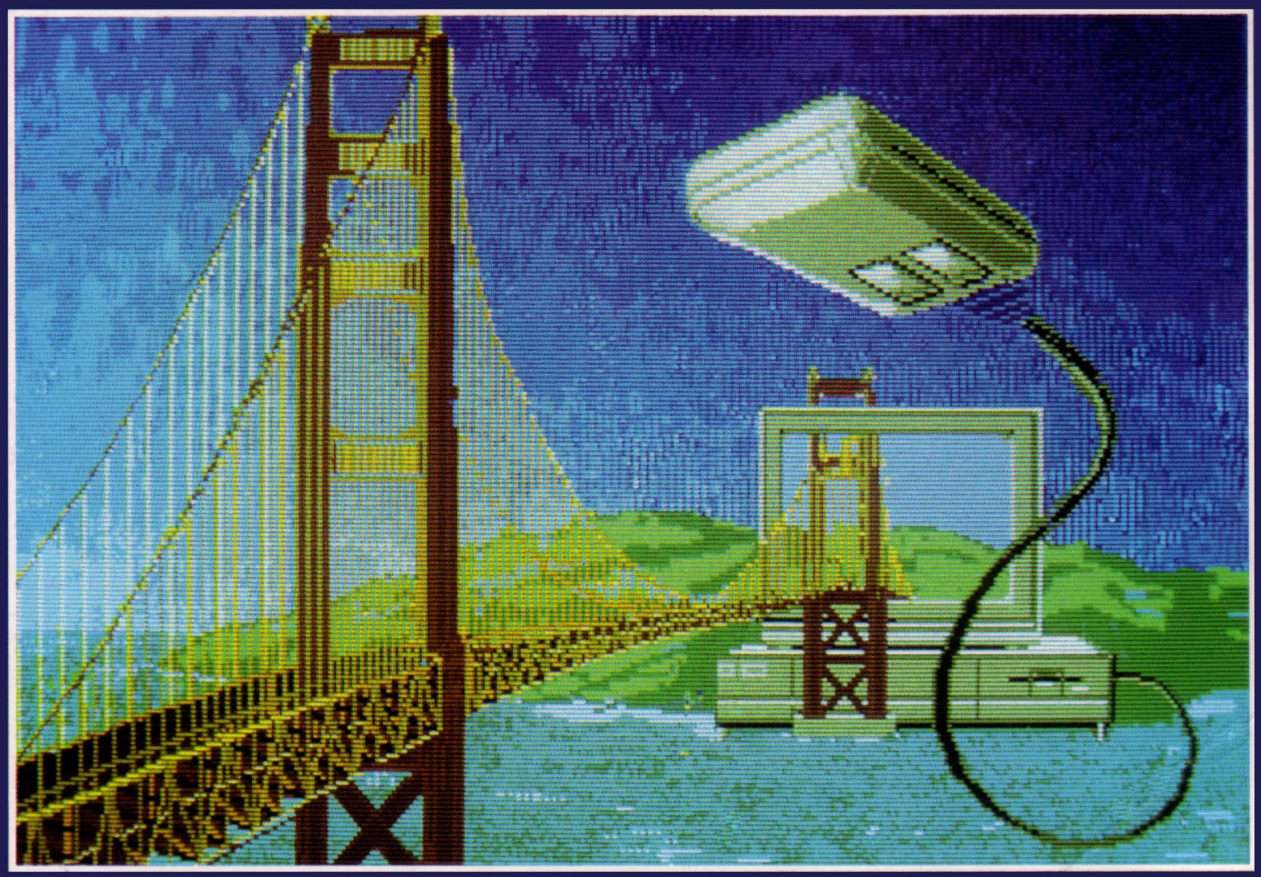
PROGRAMMARE L'AMIGA

Eugene P. Mortimore

VOLUME I



IHT
GRUPPO EDITORIALE

IHT GRUPPO EDITORIALE
DIVISIONE LIBRI

COLLANA INFORMATICA

VOLUME I

PROGRAMMARE L'AMIGA

Programmare L'AMIGA

VOLUME I/PRIMA EDIZIONE

EUGENE P. MORTIMORE

Una pubblicazione
IHT Gruppo Editoriale S.r.l.
Via Monte Napoleone, 9
20121 Milano

Titolo originale dell'opera:
Amiga Programmer's Handbook, Volume I
Edizione in lingua inglese:

SYBEX Inc., Alameda, CA, USA

Direzione editoriale della collana di Massimiliano Lisa
Traduzione di Gabriele Chiesa
Revisione di Luca Giachino e Alfredo Prochet
Collaborazione alla revisione di Mauro Gaffo e Dario Tonani
Impaginazione a cura di Andrea De Michelis

Copertina a cura di Thomas Scott Nelson (realizzata con l'ausilio
di Aegis Images, © 1985 Robert Jacob Agency/DIV Next Frontier)
Grafica dell'interno a cura di Amparo Del Rio

ISBN 88-7803-004-X

Prima edizione: aprile 1990

Dedico questo libro

a mia madre, alla memoria di mio padre,

a mio figlio Scott e a mia figlia Lori,

ognuno dei quali, a suo modo,

ha aiutato a scriverlo.

RINGRAZIAMENTI

Vorrei ringraziare Robert Peck, (Descriptive Graphics Arts, Commodore-Amiga) per l'aiuto che ha fornito nella soluzione di quesiti tecnici. Peck ha partecipato al progetto Amiga fin dall'inizio e la sua conoscenza delle chiamate di funzione è stata molto utile.

Vorrei ringraziare anche Lisa A. Siracusa dell'Amiga Technical Support della CBM, per avermi procurato informazioni aggiornate.

Ringrazio anche William Volk della Aegis Development Corporation per il suo aiuto nella soluzione di problemi tecnici e per le sue previsioni sugli sviluppi futuri dell'Amiga.

Rudolph Langer, Carole Alden, Barbara Gordon e Karl Ray meritano un ringraziamento per gli utili suggerimenti e i consigli nell'organizzazione del progetto.

Ringrazio in particolar modo Valerie Robbins, redattrice, per l'ampio aiuto nella preparazione della seconda edizione. La sua precisione e l'attenzione ai dettagli hanno assicurato un risultato finale bilanciato e preciso. Grazie Valerie!

Ringraziamenti speciali vanno anche a Olivia Shinomoto per il suo aiuto nella preparazione della seconda edizione. Oltre alla sua competente elaborazione del testo, Olivia ha preparato l'indice delle funzioni.

Grazie infine a Dawn Amsberry e Cheryl Vega per la composizione; a Suzy Anger per la correzione delle bozze; a Jannie Dresser per la stesura degli indici.

CORREZIONE DEGLI ERRORI

Questo volume è un manuale tecnico, che per la sua complessità può ancora contenere degli errori nonostante l'accurato lavoro svolto per offrire un'elevata qualità di contenuti.

Nel caso i lettori identificassero degli errori, o avessero degli aggiornamenti da proporre, possono segnalare il tutto alla redazione che si occupa di questo volume. In questo modo, nelle successive edizioni il libro conterrà correzioni e aggiornamenti certamente preziosi per tutti i programmatori. Gli aggiornamenti o le correzioni di errori vanno segnalate in forma scritta in modo chiaro, completo e conciso in modo che l'errore e la sua correzione possano essere verificati rapidamente (se necessario si possono allegare anche dischi).

Spedire il materiale a:

IHT Gruppo Editoriale
Divisione Libri
Redazione Programmare l'Amiga Vol. I
Via Monte Napoleone, 9
20121 Milano

SOMMARIO

Introduzione

1 Le funzioni della libreria Exec

2 Le funzioni di disegno e di gestione video della libreria Graphics

3 Le funzioni di animazione della libreria Graphics

4 Le funzioni di gestione del testo della libreria Graphics

5 Le funzioni della libreria Layers

6 Le funzioni della libreria Intuition

Introduzione

Introduzione

Questo libro si propone di presentare in dettaglio le funzioni del ROM Kernel dell'Amiga correlate alla grafica e alla gestione del sistema. Queste funzioni sono progettate per consentire l'accesso tramite qualsiasi linguaggio che segua le convenzioni d'interfacciamento standard , ovvero l'appropriata denominazione dei simboli, il corretto uso dei registri della CPU 68000 e il formato delle strutture di dati.

Programmare l'Amiga è una serie composta da due volumi. Il secondo analizza i dispositivi di I/O dell'Amiga, i loro comandi e le funzioni della libreria Exec predisposte per accedervi. Molte delle strutture di dati previste da queste funzioni sono trattate in entrambi i volumi. Questi due libri, nel loro complesso, definiscono e descrivono la maggior parte delle funzioni (e la maggior parte delle strutture) del software sistema dell'Amiga. Per una trattazione dettagliata delle funzioni dell'AmigaDOS si consiglia invece il *Manuale dell'AmigaDOS*, pubblicato dalla IHT Gruppo Editoriale.

La serie si rivolge a due vaste categorie di lettori. Alla prima appartengono coloro che posseggono un'approfondita esperienza nella programmazione in linguaggio C o Assembly e vogliono esaminare le possibilità offerte dall'Amiga. Queste persone conoscono già gli altri computer e vogliono scoprire se l'Amiga rappresenta un valido investimento, in linea con i loro obiettivi nella produzione di software.

La seconda categoria è formata dalle persone che posseggono e usano un computer Amiga e che ora vogliono imparare a programmarlo per sfruttarne appieno le potenzialità. A loro consigliamo di leggere *Programmare l'Amiga* parallelamente a libri sulla programmazione in linguaggio C o Assembly. Tuttavia, le funzioni qui illustrate si possono impiegare con qualsiasi linguaggio di programmazione (Basic, Pascal, Modula 2...) per il quale esista un compilatore Amiga, sempre che si seguano le norme d 'interfacciamento con il software sistema del ROM Kernel. L'AmigaBASIC fa eccezione, in quanto, pur essendo interpretato, consente un ampio uso delle funzioni del ROM Kernel.

Per impiegare le funzioni del ROM Kernel nella programmazione in linguaggio C, è necessario possedere il compilatore *Lattice C* o un pacchetto equivalente (il *Manx C68K*, per esempio), insieme alla raccolta completa dei file INCLUDE in C, che in genere appaiono sullo stesso disco del compilatore.

Per usare queste funzioni nei programmi in Assembly, si deve disporre di un compilatore Assembly e della raccolta completa dei file INCLUDE Assembly, che sono contenuti in genere nello stesso disco dell'assembler.

Il sistema Amiga

L'Amiga è uno dei personal computer più interessanti che proponga il mercato. Ciò è vero non soltanto per l'hardware ma anche per il software. L'hardware fornisce caratteristiche grazie alle quali l'utente e il programmatore possono disporre di schermi scorrevoli multipli, creare complesse animazioni e visualizzare fino a 4096 colori contemporaneamente.

Il sistema offre sia l'alta che la bassa risoluzione, e per entrambi i modi video prevede anche l'attivazione dell'interlace, che raddoppia la risoluzione verticale. Inoltre, il sistema hardware è dotato di chip dedicati al controllo dello schermo, della grafica, e alla gestione avanzata delle immagini, capaci di effettuare operazioni di riempimento di aree in modo rapidissimo, e senza impegnare la CPU. Tutte queste realizzazioni hardware conducono a una raffinatissima gestione della grafica.

Il software sistema consente di accedere a queste risorse hardware a diversi livelli, tramite raccolte di funzioni che vengono chiamate librerie. Il sistema è dotato di una completa biblioteca di librerie predefinite, in grado di coprire il più ampio spettro di esigenze, ma è anche aperto ad accogliere quelle create ex novo dai programmatori. I due generi di libreria seguono le stesse convenzioni per l'interazione con i task, e le varie procedure di gestione.

Per via della particolare architettura software dell'Amiga, nel sistema esiste un unico indirizzo di memoria assoluto (l'indirizzo 0x00000004, denominato AbsExecBase) al quale fanno poi riferimento tutti gli altri indirizzi impiegati nell'interazione con il sistema. In questa locazione di memoria si trova un puntatore che contiene l'indirizzo di una struttura di tipo Exec, unico e fondamentale punto di riferimento per l'intero sistema operativo. Nella struttura sono contenute tutte le informazioni per localizzare nel sistema qualsiasi altra risorsa, come le librerie e i dispositivi. Questo significa che le varie routine, le librerie e i dispositivi che risiedono su disco non devono essere caricati in zone di memoria prefissate; è il sistema che provvede a disporli dove più è comodo nello spazio di otto megabyte di memoria indirizzabile dalla CPU.

Quest'assenza di locazioni e indirizzi assoluti non si applica soltanto ai codici, ma anche ai dati e in particolare a quelli che vengono elaborati dai chip dedicati. Contrariamente ad altri computer, per esempio, non è necessario che la bitmap di un'immagine si trovi in un preciso segmento di memoria. Le bitmap possono essere disposte in qualsiasi punto del primo megabyte di RAM, un settore di memoria che viene chiamato "chip RAM" proprio perché rappresenta lo spazio indirizzabile dai chip dedicati (si noti che nella macchine non ancora dotate di ECS, Enhanced Chip Set, il campo d'indirizzamento dei chip dedicati è ristretto ai primi 512K della RAM). La necessità di codici e dati indipendenti dalle locazioni di memoria è dovuta alla struttura multitasking del sistema operativo, che per sua natura dev'essere in grado di considerare la memoria come una risorsa da ripartire dinamicamente fra tutti i task in esecuzione, a seconda delle situazioni contingenti.

Le funzioni del ROM Kernel dell'Amiga sono organizzate gerarchicamente. Alla sommità della gerarchia si trovano le funzioni della libreria Icon, grazie alle quali si possono sviluppare programmi applicativi capaci di usare in modo

standard le icone e il sistema dei file dell'AmigaDOS. Subito dopo seguono le funzioni della libreria Intuition. Il programma *Workbench*, per esempio, oltre a impiegare la libreria Icon, usa la libreria Intuition per produrre le sue schermate e per interagire con il sistema di gestione dei file. Intuition, d'altra parte, impiega le funzioni delle librerie Graphics e Layers per gestire il suo output su schermo. Le funzioni delle librerie Intuition e Icon forniscono un' interfaccia di alto livello per sviluppare programmi che prevedono un'interazione standard con l'utente.

Contenuto dei capitoli

Questo libro è scritto sotto forma di manuale di riferimento. Di ogni funzione sono presentate e discusse la sintassi, l'impiego e gli effetti.

Il capitolo 1 tratta le funzioni appartenenti alla libreria Exec (o, più semplicemente, all'Exec), che forniscono il meccanismo per gestire i task, i dispositivi, le librerie, le liste di sistema, la memoria e altri aspetti del sistema Amiga. L'Exec presiede anche e soprattutto lo scambio di controllo fra i task (task-switching) usando le loro rispettive priorità d'esecuzione e il sistema degli interrupt.

Il capitolo 2 esamina le funzioni di disegno e gestione del video contenute nella libreria Graphics. Queste funzioni forniscono il meccanismo per creare una bitmap, utilizzarla per disegnare e visualizzarla sullo schermo. Il loro obiettivo specifico consiste nella completa definizione di playfield, o view, sullo schermo dell'Amiga. Intuition, per esempio, utilizza la libreria Graphics per creare i cosiddetti "schermi di Intuition", che in realtà non sono altro che viewport create dalle funzioni della libreria Graphics su diretta richiesta di Intuition. Queste funzioni, insieme a opportune macro nei linguaggi C e Assembly, permettono di utilizzare i cinque modi grafici dell'Amiga per realizzare schermate anche molto complesse. Costituiscono il primo dei tre gruppi in cui vengono suddivise le funzioni della libreria Graphics.

Il capitolo 3 tratta le funzioni d'animazione della libreria Graphics, che consentono di definire e usare sprite hardware, sprite virtuali e bob (Blitter object, ovvero oggetti visualizzati e gestiti dal coprocessore Blitter), per creare oggetti mobili sullo schermo video dell'Amiga. I bob possono essere raggruppati in oggetti e componenti d'animazione più complessi, al fine di produrre effetti di movimento molto sofisticati. Questi elementi mobili possono essere combinati con i playfield statici prodotti con le funzioni esaminate nel capitolo 2. Le funzioni di animazione e le relative macro formano il secondo dei tre gruppi di funzioni della libreria Graphics.

Il capitolo 4 tratta le funzioni della libreria Graphics che consentono di creare e gestire fonti-carattere e testi sullo schermo. Oltre a creare nuove fonti-carattere, permettono di salvarle su disco e richiamarle in memoria in qualunque momento per visualizzare un testo sullo schermo. Le funzioni di gestione testi e le relative macro costituiscono il terzo dei tre gruppi di funzioni della libreria Graphics.

Il capitolo 5 tratta le funzioni della libreria Layers, che permettono di

strutturare il contenuto dello schermo in più "strati", e di utilizzare questi strati indipendenti e sovrapponibili per creare finestre. Queste funzioni, inoltre, offrono a una pluralità di task la possibilità d'intervenire sulla stessa bitmap.

Il capitolo 6 tratta le funzioni di Intuition che consentono di creare programmi applicativi dotati di un'interfaccia utente user-friendly e standard. La libreria Intuition offre funzioni molto semplici per definire schermi e finestre che oltre ad avere la dote di essere standard (una finestra di Intuition ha sempre la stessa struttura di base, qualunque sia l'applicazione che la apre), in più evitano l'uso delle più complesse funzioni della libreria Graphics. Comunque è anche possibile utilizzare le funzioni di Intuition in combinazione con alcune funzioni della libreria Graphics per produrre programmi applicativi dotati d'interfacce grafiche particolarmente elaborate e parzialmente standard.

Il capitolo 7 tratta le funzioni contenute nella libreria Icon, comunemente chiamate funzioni Workbench per indicare che è principalmente il programma *Workbench* a farne uso, anche se ovviamente sono disponibili per qualunque applicazione. Queste funzioni permettono di lavorare con icone di programmi applicativi sullo schermo Workbench, lo schermo standard aperto da Intuition per default (si noti che questo schermo ha lo stesso nome del programma *Workbench*, ma è da esso indipendente), e di associare icone ai propri file. Se il *Workbench* è attivo, l'utente potrà poi selezionare queste icone, ottenendo diversi risultati a seconda del tipo di selezione.

In aggiunta a questi sette capitoli, il libro comprende due appendici. L'appendice A è un glossario dei termini usati nel corso della trattazione. L'appendice B contiene un sommario dei quattro modi video addizionali caratteristici dell'Amiga e propone per ognuno di essi un sintetico programma di gestione.

Infine, l'indice analitico fornisce i riferimenti alle spiegazioni delle funzioni. Il sommario elenca tutte le funzioni e le macro ordinate per libreria, mentre il "sommario delle funzioni" le elenca di nuovo in ordine alfabetico.

Le librerie

Le librerie sono così importanti nel sistema Amiga che è giocoforza dedicare loro alcune considerazioni generali. Per prima cosa è importante capire che le cinque librerie trattate in questo libro rispettano in toto la definizione generale di libreria che viene data nel capitolo 1, con l'unica peculiarità di essere parte integrante del sistema operativo dell'Amiga, e quindi disponibili a tutte le applicazioni.

Le librerie si differenziano in residenti (inserite nella memoria ROM della macchina), e non-residenti (ovvero residenti su disco). Le prime sono sempre disponibili, mentre le seconde, per essere considerate disponibili, devono essere caricate dal disco e aperte da un task. Le librerie non-residenti vengono caricate in RAM solo all'occorrenza.

Il sistema di gestione delle librerie è abbastanza flessibile da permettere ai programmatori di creare e aggiungere al sistema librerie di funzioni dedicate a particolari incarichi non previsti dal sistema operativo. Per creare una libreria

personalizzata è necessario chiamare la funzione MakeLibrary (si veda il capitolo 1). Una volta che una libreria è stata creata, dev'essere salvata su disco, ed essere aggiunta alle altre librerie non-residenti presenti sul disco sistema. Compiute queste operazioni, la libreria può essere aperta dai programmi tramite la funzione OpenLibrary, che nel caso di una libreria non-residente provvede anche a caricarla da disco in maniera trasparente all'applicazione.

Le istruzioni che seguono mostrano un esempio di codici in linguaggio C che aprono la libreria Graphics e ne ricavano l'indirizzo base memorizzandolo nella variabile GfxBase.

```
LONG GfxBase;
GfxBase = OpenLibrary ("graphics.library", ØL);
if (GfxBase == ØL)
exit (LIBRERIA_GRAPHICS_NON_TROVATA);
```

In questa sequenza, la dichiarazione LONG definisce una variabile, GfxBase, di tipo long, ovvero alloca quattro byte per contenere il valore assunto dalla variabile. L'istruzione successiva chiama la funzione OpenLibrary per aprire la libreria Graphics, e memorizza nella variabile GfxBase l'indirizzo base che OpenLibrary restituisce. Questo indirizzo individua in memoria l'inizio della struttura GfxBase che definisce la libreria. Questa struttura viene creata dal sistema e possiede come primo elemento una struttura Library, la struttura di gestione standard delle librerie. Ciascuna libreria del ROM Kernel possiede una struttura di gestione personalizzata che ha come primo elemento una struttura Library: ExecBase per la libreria Exec, GfxBase per la libreria Graphics, LayersBase per la libreria Layers, DiskfontBase per la libreria Diskfont, IntuitionBase per la libreria Intuition e IconBase per la libreria Icon.

Una diversa organizzazione formale delle istruzioni C necessarie per aprire la libreria Graphics è la seguente:

```
struct GfxBase *GfxBase;
GfxBase = (struct GfxBase *) OpenLibrary ("graphics.library", ØL);
if (GfxBase == ØL)
exit (LIBRERIA_GRAPHICS_NON_TROVATA);
```

Di diverso dalla precedente c'è solo il formalismo, dal momento che con la compilazione si ottengono le stesse identiche istruzioni Assembly. In entrambi gli esempi, si assume che la costante LIBRERIA_GRAPHICS_NON_TROVATA sia stata già definita dal nostro programma in linguaggio C. Se la libreria non può essere aperta a causa di una condizione d'errore (per esempio, la libreria è non-residente e non è presente sul disco, o la memoria disponibile non è sufficiente), la procedura di controllo dell'errore causa l'uscita dal programma.

I dispositivi di I/O sono ancora librerie, ma di tipo diverso; per aprirli, si deve ricorrere alla funzione OpenDevice:

errore = OpenDevice (DEVICENAME, unitnumber, iORequest, flag)

Con OpenDevice, nella variabile "errore" si riceve una condizione d'errore anziché l'indirizzo base della libreria associata al dispositivo. Quindi, non si deve dichiarare una locazione di memoria; la funzione OpenDevice memorizza infatti l'indirizzo base all'interno della struttura IORequest indicata come argomento. Dopo che il dispositivo è stato aperto, questo indirizzo viene impiegato dal sistema per interfacciare le routine del dispositivo con quelle del sistema. La struttura con cui viene definito e gestito un dispositivo in memoria si chiama Device, ed è sostanzialmente identica alla struttura Library.

Le strutture

Accanto alle funzioni e alle macro, le strutture sono il principale meccanismo di programmazione dell'Amiga. Con queste strutture si può lavorare in Assembly, in C o in qualsiasi altro linguaggio che preveda la possibilità di comunicare direttamente con il software sistema. Ogni libreria prevede l'uso di alcune strutture che servono principalmente per mantenere i dati ai quali accedono le sue funzioni. Le precise definizioni di queste strutture e i loro parametri si trovano nei file INCLUDE di programmazione.

Quasi tutte le strutture contengono puntatori, tramite i quali si possono creare serie di strutture concatenate. Questo concetto è applicato ampiamente nel software sistema dell'Amiga. Inoltre, molte strutture contengono al loro interno altre strutture (che assumono il nome e il ruolo di sotto-strutture). Un esempio è la struttura Gadget contenuta nella struttura WBObject; all'interno della struttura WBObject, infatti, è inserita una struttura Gadget completa, e non un semplice puntatore.

In generale, le strutture appartengono a tre categorie: strutture che non sono collegate a nessun'altra struttura, strutture predisposte per essere inserite in liste semplici (liste di strutture concatenate che si possono scorrere solo in un senso) e strutture predisposte per essere inserite in liste doppie, o a doppia concatenazione (liste di strutture concatenate che si possono scorrere in entrambi i sensi). Le strutture predisposte per le liste semplici contengono un solo puntatore che viene impiegato per individuare l'elemento successivo della lista. Le strutture predisposte per le liste doppie contengono invece due puntatori, uno che individua l'elemento successivo (detto successore) e uno che individua l'elemento precedente (detto predecessore).

La Figura I.1 mostra un insieme di strutture, Struttura1, Struttura2 e via di seguito fino a StrutturaN, collegate fra loro secondo lo schema della doppia concatenazione: così disposte costituiscono una lista doppia che può essere esaminata scorrendone i nodi (nome generico per gli elementi di una lista) nei due sensi. Struttura1 è la prima struttura della lista; Struttura2 è la seconda e StrutturaN è l'ultima. Tutte queste strutture sono dello stesso tipo. Per esempio, potrebbero essere di tipo Bob.

Nel sistema operativo dell'Amiga l'uso delle liste è frequentissimo, e conoscerne la filosofia è assolutamente necessario. Due peculiarità delle liste

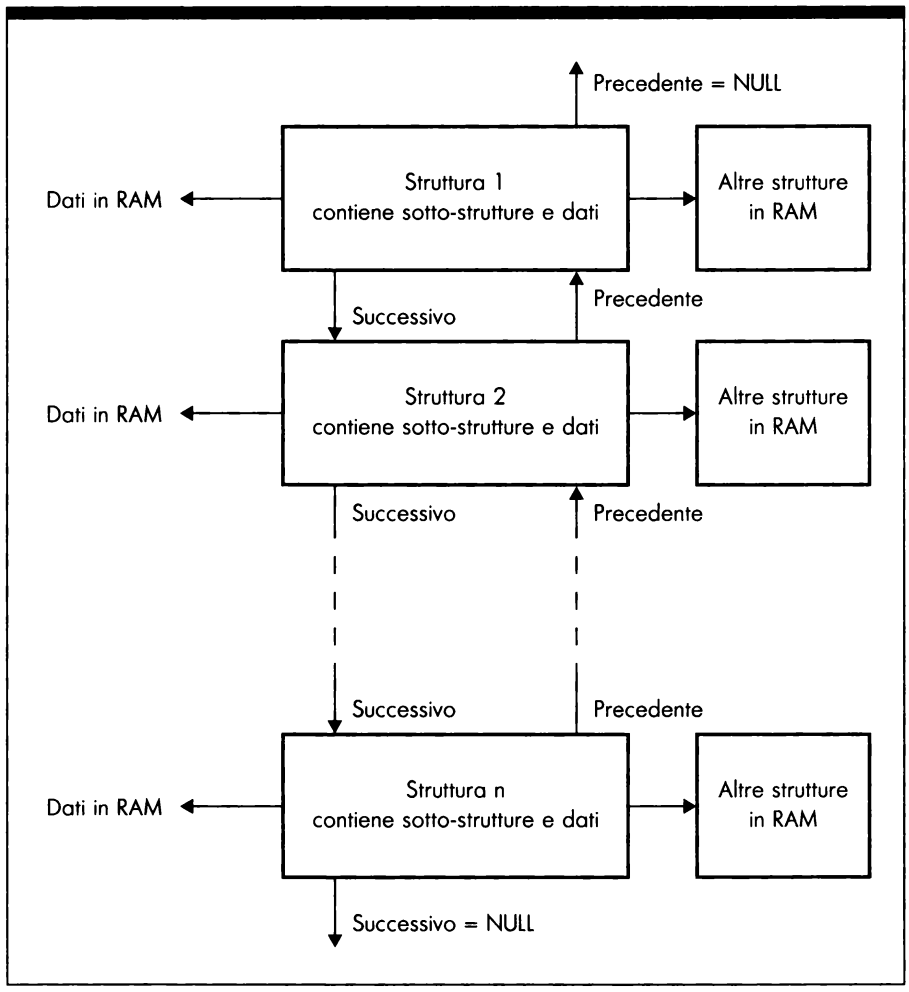


Figura I.1: *Lista a doppia concatenazione*

di dati è che possono crescere e regredire dinamicamente, e che i loro elementi possono risiedere dovunque in memoria, dal momento che sono i puntatori a instaurare i necessari legami.

Nella figura le frecce rappresentano i parametri puntatori delle strutture, cioè i parametri predisposti per contenere gli indirizzi di memoria di altre strutture. Queste frecce, identificate come Successivo e Precedente puntano ad altre strutture dello stesso tipo. Sottolineamo ancora che grazie ai puntatori, che permettono di risalire agli indirizzi di tutte le strutture del concatenamento, le strutture di una lista concatenata non hanno l'obbligo di essere adiacenti in RAM.

Nelle strutture esistono tre categorie di parametri: parametri di sistema, parametri comuni e parametri privati. I parametri di sistema possono essere

alterati soltanto dalle routine di sistema. Un esempio è il puntatore NextVSprite nella struttura VSprite. I parametri comuni possono essere alterati sia dal sistema sia dai task delle applicazioni. Un esempio è il parametro Flags nella struttura VSprite. I parametri privati sono invece accessibili in scrittura solo da un task, generalmente quello che ha creato la struttura o che se n'è appropriato. Un esempio è costituito dal parametro Height della struttura VSprite.

Ciascun parametro, inoltre, è caratterizzato dal tipo di dati che deve contenere. Il tipo sancisce quanti byte di memoria sono richiesti per allocare il parametro in memoria e come i suoi byte e bit devono essere interpretati dal sistema e dalle routine dei task. Per esempio, un parametro di tipo **struct GfxBase *** essendo un puntatore occupa in memoria quattro byte; un parametro di tipo **short** occupa due byte, mentre un parametro di tipo **struct Library** occupa 34 byte (quanti ce ne vogliono per contenere una struttura Library).

Il C prevede una nutrita serie di variabili, tuttavia nella programmazione dell'Amiga si è preferito ridefinirne i tipi e aggiungerne altri che fossero più facilmente memorizzabili. Così, i programmatori in C che creano applicazioni per l'Amiga possono scegliere fra i tipi di variabili standard previsti dal C e quelli più esplicativi previsti dall'ambiente di programmazione dell'Amiga. Nella nostra trattazione useremo sempre i tipi ridefiniti per l'Amiga, anche perché sono quelli che vengono impiegati nelle definizioni di struttura dei file INCLUDE. Si consultino i file INCLUDE exec/types.h e types.i per vedere a quali tipi standard corrispondono i tipi ridefiniti per l'Amiga (BYTE, UBYTE, WORD, UWORD, LONG, ULONG, SHORT, USHORT, APTR, STRPTR...).

I parametri delle strutture che intervengono nella programmazione possono essere modificati in due modi: dal sistema o dal task. Alcuni parametri sono di esclusiva competenza del sistema, anche quando la struttura è stata creata dal task; gli altri sono il veicolo tramite il quale il task può modificare il comportamento del sistema. Nell'Amiga, le informazioni sono sempre contenute in una struttura e collegate ad altre informazioni. Queste strutture diventano in pratica le informazioni che i task e il sistema si scambiano per cooperare. Per aprire una finestra, per esempio, il task deve allocare una struttura NewWindow e inizializzarne alcuni parametri. Quando poi il task chiama la funzione OpenWindow, il sistema alloca in memoria una struttura Window e ne inizializza i parametri secondo quanto indicato dal task nella struttura NewWindow. Da quel momento in poi la finestra appare sul video, e sistema e task utilizzano la relativa struttura Window per gestirla.

file INCLUDE

I file INCLUDE associati con ciascuna libreria dell'Amiga sono ricchi d'informazioni su ciò che si deve sapere per utilizzare le funzioni di libreria. In particolare, i file INCLUDE definiscono le strutture correlate alle librerie. Le definizioni di struttura incluse in questo libro vengono presentate nella versione in linguaggio C.

Ciascun file INCLUDE contiene due tipi di direttive condizionali per la compilazione. Le direttive del primo tipo servono per stabilire se un particolare

file INCLUDE è già stato incluso durante l'elaborazione preliminare; se la condizione non si verifica, viene eseguita un'opportuna direttiva INCLUDE per includerlo. Generalmente questo accade quando un file INCLUDE richiede che il compilatore abbia già compilato un altro file INCLUDE gerarchicamente più in alto. Per esempio, il file INCLUDE gfxmacros.h prima di procedere con le definizioni delle macro-istruzioni grafiche accerta ed eventualmente pretende l'inclusione dei file INCLUDE graphics/rastport.h e graphics/gfx.h. Il secondo tipo di direttiva serve per accertare la definizione di costanti di compilazione, ed eventualmente effettuarla.

Alcuni file INCLUDE contengono inoltre le dichiarazioni dei tipi di dati restituiti da alcune funzioni, in modo che ne venga tenuto conto durante la compilazione (più precisamente nel corso del controllo della compatibilità di tipo fra i dati restituiti e le variabili destinate ad accoglierli). Un esempio di queste dichiarazioni si trova nel file INCLUDE workbench/icon.h.

In certe versioni dei file INCLUDE, le definizioni di struttura sono anche commentate; questi commenti sono utili per capire la funzione e il comportamento di una struttura. Anche il tipo dei parametri (BYTE, WORD, puntatore...) indicato per ogni elemento della struttura contribuisce a chiarirne il significato. Osservando i tipi e conoscendo il numero di byte richiesto da ognuno si può ricavare la grandezza della struttura, cioè il numero di byte di memoria che richiede per essere allocata. Questa informazione, comunque, si ottiene più facilmente utilizzando l'operatore sizeof del C.

I file INCLUDE contengono inoltre le definizioni di tutte le costanti hardware e di tutti i flag a esse associati. Le costanti talvolta devono essere indicate come argomenti nelle chiamate di funzione. Tramite i file INCLUDE è anche possibile rilevare quale bit rappresenta un dato flag, anche se questa è un'informazione poco utile in quanto conviene sempre ricorrere alle costanti simboliche. Non c'è infatti alcun bit di flag utilizzabile dai task per il quale non sia prevista almeno una costante simbolica. Inoltre, l'uso delle costanti simboliche garantisce la corretta compilazione di un sorgente anche quando il valore di una costante dovesse cambiare da una versione all'altra del sistema operativo (una situazione che comunque non dovrebbe mai verificarsi, dal momento che comprometterebbe la compatibilità verso il basso).

Nei file INCLUDE risiedono anche le definizioni di tutte le macro-istruzioni disponibili per rendere meno intricata la programmazione. Per esempio, il file INCLUDE gfxmacros.h. contiene l'esatta definizione delle macro CINIT, CEND, CMOVE e CWAIT, utilissime con le funzioni della libreria Graphics. Spesso la procedura per chiamare queste macro è identica a quella per chiamare una funzione, e come per le funzioni sono richiesti specifici argomenti.

Alcuni file INCLUDE relativi alla libreria Exec sono particolarmente importanti. Il file exec/alerts.h contiene la definizione di tutti gli alert di sistema, compresi quelli recuperabili. Gli alert relativi a situazioni non recuperabili vengono trattati nel capitolo 6; si tratta di codici numerici che visualizzati attraverso opportuni alert di Intuition forniscono all'utente informazioni sul motivo del problema. Questi avvertimenti precedono il totale blocco del sistema. Il programmatore può utilizzarli per effettuare il debug dei propri programmi.

Il file errors.h contiene una lista di quattro codici d'errore che si possono

ricevere nelle operazioni di I/O con i dispositivi dell'Amiga: IOERR_OPENFAIL, IOERR_ABORTED, IOERR_NOCMD e IOERR_BADLENGTH. Queste costanti valgono rispettivamente -1L, -2L, -3L e -4L.

Il file execbase.h contiene la definizione della struttura ExecBase, la principale struttura del sistema, che viene allocata e inizializzata durante il boot. Il suo primo elemento è una struttura Library, nella quale durante l'attivazione del sistema vengono inserite e poi mantenute tutte le informazioni di gestione della libreria Exec. La struttura Library è poi seguita da un folto insieme di parametri che permettono alle routine della libreria di tener sott'occhio quello che accade nel sistema.

Per esempio, questi parametri contengono le informazioni necessarie al sistema per effettuare lo scambio di controllo fra i task, per tener conto del tempo di CPU sfruttato dal task in esecuzione, per individuare in memoria le varie routine di trap e di exception del task in esecuzione, per tenere il conto del livello di nidificazione degli interrupt e così via. Altri parametri contengono l'intervallo dopo il quale deve verificarsi uno scambio di controllo fra task dotati della stessa priorità d'esecuzione; la quantità di memoria di cui il sistema può usufruire; la versione del sistema operativo. La struttura ExecBase contiene inoltre un insieme di dieci sotto-strutture List che permettono al sistema Exec di tenere sotto controllo i segmenti di memoria ancora disponibili, i task che aspettano di ricevere il controllo, i task che sono in stato di attesa e altre categorie di elementi del sistema. Tutte queste informazioni, contenute in apposite strutture, vengono continuamente tenute sotto controllo dal sistema.

Convenzioni usate nel sistema

La programmazione dell'Amiga, per quanto assai flessibile, impone la conoscenza e l'osservanza di alcune norme, o convenzioni, che valgono per il sistema e per i task che in esso vengono mandati in esecuzione. Rispettando queste convenzioni si ha la certezza che i propri task si interfacceranno con il sistema in modo corretto.

Uso dei registri

Tutte le funzioni del sistema sono predisposte per ricevere valori in input e restituire valori in output, impiegando i registri della CPU. I valori di input vengono indicati dal task negli "argomenti" delle chiamate alle funzioni, mentre i valori di output vengono restituiti dalle funzioni al termine della loro esecuzione. La conoscenza delle regole d'uso dei registri è fondamentale se si desidera programmare in linguaggio Assembly, eventualmente anche all'interno di sorgenti in C. In linguaggio C, invece, il programmatore non deve occuparsi dei registri: sono le librerie di compilazione (linked) a interfacciare correttamente la sintassi di chiamata delle funzioni prevista dal C (passaggio di parametri sullo stack) con la sintassi di chiamata prevista dalle funzioni del

sistema operativo (passaggio di parametri nei registri della CPU).

Gli argomenti usati nelle chiamate di funzione ricadono in due categorie: argomenti puntatore e argomenti dato. Gli argomenti puntatore si suddividono ulteriormente in argomenti che contengono indirizzi di strutture, e argomenti che contengono indirizzi di blocchi di memoria.

I registri d'indirizzamento A0/A7 sono sempre usati per gli argomenti puntatore. Nelle chiamate alle funzioni, tutti gli argomenti che contengono indirizzi di memoria devono essere memorizzati in questi registri, a cominciare dall'argomento più a sinistra e spostandosi verso gli argomenti successivi della sintassi di chiamata della funzione. Il primo argomento puntatore dev'essere memorizzato in A0, il successivo in A1 e così via. Questa regola è scrupolosamente rispettata dalle funzioni della libreria Intuition, ma sottolineamo subito che altre librerie se ne discostano significativamente. È quindi necessario esaminare individualmente le chiamate di funzione per determinare quali registri d'indirizzamento vengono usati da ciascuna funzione.

I registri dato sono trattati in modo simile ai registri d'indirizzamento. Nelle chiamate alle funzioni, tutti gli argomenti che contengono dati devono essere memorizzati nei registri dato della CPU (D0/D7), a cominciare dall'argomento più a sinistra e spostandosi verso gli argomenti successivi nella sintassi di chiamata della funzione. Il primo argomento dato dev'essere memorizzato in D0, il successivo in D1 e così via. Questa regola è scrupolosamente rispettata dalle funzioni della libreria Intuition, ma anche in questo caso altre librerie se ne discostano significativamente. Ancora una volta, è meglio controllare la definizione della sintassi di chiamata di ciascuna funzione per verificare fino a che punto la regola è rispettata.

Ci sono alcune norme addizionali a proposito dell'uso dei registri. Il registro A6, per esempio, ha una funzione speciale: deve sempre contenere l'indirizzo base della libreria della quale si chiamano le funzioni. Quando una libreria viene aperta tramite la funzione OpenLibrary, il task riceve l'indirizzo della relativa struttura Library. Quando poi deve chiamare una funzione della libreria, prima di effettuare la chiamata deve memorizzare nel registro A6 l'indirizzo base della libreria. Gli indirizzi base costituiscono i riferimenti assoluti tramite i quali, grazie a opportuni offset, è possibile individuare tutte le funzioni della libreria. Programmando in C, è il compilatore a occuparsi di generare gli opportuni codici in Assembly; c'è un'unica regola: si devono memorizzare gli indirizzi base delle librerie in opportune variabili che il linker si aspetta di trovare definite. Per esempio, se nel corso di un programma in C occorre aprire la libreria Intuition, l'indirizzo base restituito dalla funzione OpenLibrary dev'essere memorizzato in una variabile che deve rispondere al nome di IntuitionBase, perché solo così il linker potrà individuarla. Nel corso del libro verranno indicati i nomi delle variabili nelle quali memorizzare l'indirizzo base di ogni libreria.

I registri D0, D1, A0 e A1 sono considerati registri alterabili, cioè modificabili in ogni momento. Sulla base di questa convenzione, le funzioni di sistema li utilizzano senza preoccuparsi di salvaguardarne preventivamente il contenuto. Tutti gli altri registri della CPU devono invece essere salvaguardati. Se qualcuno di questi registri viene usato da una funzione, il suo contenuto dev'essere salvato e opportunamente ripristinato, e questo vale anche per le

funzioni create in Assembly dai programmatori.

Il risultato principale a cui giunge una funzione dev'essere sempre restituito nel registro D0. Tuttavia, molto spesso le funzioni restituiscono in opportune strutture anche altre informazioni, magari modificando lo stato di certi parametri. Conoscendo queste strutture e i relativi parametri che una funzione modifica, è possibile rilevare gli interventi effettuati dalla funzione. Un caso tipico sono le funzioni che inizializzano una struttura, come le funzioni InitArea, InitBitMap, InitRastPort, InitTmpRas, InitView e InitVPort della libreria Graphics. Un altro esempio sono le funzioni che modificano particolari parametri di struttura, come SetAPen e SetBPen, sempre della libreria Graphics, che variano due parametri della struttura RastPort.

Valori restituiti dalle funzioni

Alcune funzioni di libreria dell'Amiga non sono predisposte per restituire valori. Altre invece restituiscono nel registro D0 uno dei seguenti tipi di valore: un indirizzo, in genere di una struttura o di un'area di memoria, o un dato numerico, molto spesso un codice d'errore. Eguagliando la chiamata a una variabile del proprio programma, il valore restituito nel registro D0 viene automaticamente trasferito in quella variabile. In C, talvolta, il risultato restituito da una funzione non viene memorizzato in alcuna variabile, e viene direttamente impiegato per effettuare un controllo condizionale.

La Tavola I.1 riassume i tipi di valori che le funzioni possono restituire, insieme allo spazio di memoria che richiedono ed esempi delle funzioni di sistema che li restituiscono.

Dati espressi su quattro byte sono di tipo ULONG, o di tipo LONGBITS quando ogni bit è significativo. Dati espressi su due byte sono del tipo WORD o di tipo WORDBITS quando ogni bit è significativo. Dati espressi su un byte sono di tipo BYTE o BYTEBITS quando ogni bit è significativo. Dati espressi su quattro bit sono di tipo BYTEBITS, con ogni bit del nibble significativo.

Tavola I.1: *Valori restituiti dalle funzioni*

Valore restituito	Dimensione	Esempio
Indirizzo di una struttura	4 byte	InitRastPort
Indirizzo di un blocco di memoria	4 byte	Allocate
Dato	32 bit	AllocSignal
Dato	16 bit	AllocTrap
Dato	1 byte	SetTaskPri
Dato	4 bit	SetIntVector

I parametri di flag

I parametri di flag delle strutture giocano un ruolo importante nella programmazione dell'Amiga. Vengono così definiti perché servono come flag, ovvero "bandierine di segnalazione" per il sistema e per i task (ricordiamo che ogni flag è rappresentato da un bit). Sia il sistema sia i task rilevano e impostano lo stato di questi flag rispettivamente per prelevare o inserire informazioni. Vi sono parametri di flag quasi in ogni struttura. Un buon esempio è il parametro tc_Flags nella struttura Task (la struttura di gestione di un task). Questo parametro è di tipo UBYTE (unsigned byte), e può contenere una qualsiasi combinazione di flag impostati a 1 e a 0. I flag TB_PROCTIME, TB_STACKCHK, TB_SWITCH e TB_LAUNCH sono sotto il dominio del task, mentre il flag TB_EXCEPT è sotto il controllo del sistema operativo. Avremmo potuto indicare questi flag tramite il loro numero di bit, ma l'uso delle costanti simboliche è molto più esplicativo.

Si noti che, come regola generale, i nomi delle costanti con le quali si attivano e si disattivano i flag appaiono interamente in maiuscolo. Come si può osservare dai loro nomi, ciascuno dei flag citati ha qualcosa a che fare con la gestione del task a cui appartengono. Per esempio, se il sistema rileva che nella struttura Task di un task è impostato a 1 il flag TB_PROCTIME del parametro tc_Flags, in un apposito parametro verrà tenuto conto del tempo di CPU che il sistema elargisce a questo task.

Esistono due modi per definire lo stato dei flag. È possibile predisporre istruzioni in C che impostino a 1 o a 0 i bit dei parametri, oppure affidarsi a funzioni che richiedano come argomento il flag su cui agire. Il primo metodo è, di gran lunga, il più comune.

Un esempio del secondo metodo è la funzione CreateBehindLayer della libreria Layers, che prevede come settimo argomento un campo flag. Quando questa funzione restituisce il controllo, il valore dell'argomento flag indicato nella chiamata viene copiato nel parametro Flags della struttura Layer creata dalla funzione; i flag in esso impostati determinano le caratteristiche del layer. I flag che si possono impostare nell'argomento flag della funzione (LAYERSIMPLE, LAYERSMART o LAYERSUPER) permettono di scegliere fra tre tipi di layer mutuamente esclusivi.

Convenzioni sulle notazioni adottate in questo libro

A causa della complessità del soggetto trattato, anche la notazione usata in questo libro è complessa. Diciamo subito che la notazione aderisce completamente a quella usata nei file INCLUDE forniti dalla Commodore-Amiga. Ecco comunque alcune indicazioni che saranno d'aiuto.

Primo, i termini generali sono scritti in lettere minuscole (come per esempio task, message port, bitmap...). Alcuni sono stati tradotti in italiano, come message (messaggio), device (dispositivo), mentre altri sono stati lasciati in inglese per evitare possibilità di confusioni o fraintendimenti.

Secondo, i nomi delle strutture hanno sempre l'iniziale maiuscola e

proseguono in minuscolo, eccetto quando sono costituiti da parole composte. In questo caso, anche la seconda parola inizia con la lettera maiuscola (come per esempio struttura MsgPort e struttura BitMap).

Terzo, le costanti, come quelle dei codici d'errore, dei valori di default di alcuni parametri e dei flag, sono state scritte tutte in lettere maiuscole (come per esempio le costanti MEMF_CHIP e MEMF_CLEAR, due flag che nella chiamata della funzione AllocMem specificano se l'area di memoria dev'essere allocata nella chip RAM oppure azzerata).

Quarto, quando occorreva scegliere un nome per una variabile puntatore adibita a contenere l'indirizzo di una struttura, si è deciso di utilizzare lo stesso nome della struttura. Esempi di ciò sono il nome msgPort per la variabile puntatore che individua in memoria strutture MsgPort, e il nome interrupt per le variabili che individuano strutture Interrupt. Si noti che il nome delle variabili puntatore inizia sempre con la lettera minuscola. Occasionalmente però, il nome del puntatore non corrisponde esattamente al nome della struttura. Un esempio è la variabile taskCS, che abbiamo utilizzato per individuare in memoria strutture Task. In questo caso per il nome della variabile è stato scelto un nome un po' più descrittivo (taskCS, Struttura di Controllo del task). Un'altra eccezione a questa regola sono le variabili adibite a contenere l'indirizzo base delle librerie, per le quali è imperativo impiegare nomi convenzionali, come IntuitionBase per l'indirizzo base della libreria Intuition. In ogni caso, il legame tra la variabile puntatore e la struttura sarà sempre evidente.

Quinto, i nomi delle variabili di tipo struttura seguono la stessa convenzione adottata per le variabili puntatore a struttura.

Sesto, tutti i nomi di funzione iniziano con lettera maiuscola e proseguono in minuscolo, eccetto uno o più cambiamenti in maiuscolo per i nomi composti. Esempi di questa notazione sono le funzioni AddDevice, AddTask, Cause e Insert nel capitolo 1 e le funzioni AreaDraw, AreaMove, Draw e Move nel capitolo 2.

Settimo, la maggior parte delle funzioni prevede uno o più argomenti nella chiamata, mentre altre, come ColdReset, SuperState, OwnBlitter e DisownBlitter, non ne richiedono affatto. La maggior parte degli argomenti di funzione, come i puntatori alle strutture e i campi flag, sono scritti in lettere minuscole, a eccezione dell'iniziale della seconda parola che compone il nome.

In linea generale, le parole interamente in maiuscolo sono usate nelle seguenti circostanze:

- per i nomi dei nodi inseribili in una lista. Per esempio NT_INTERRUPT e NT_DEVICE, che rappresentano tipi di nodi previsti dalle funzioni dell'Exec di gestione delle liste (AddHead, AddTail...).

- Per i nomi dei vettori di salto alle funzioni di una libreria o di un dispositivo. Esempi di questo tipo sono i vettori di salto LIB_OPEN e LIB_CLOSE, previsti da ogni libreria, trattati nel capitolo 1, che sono in pratica gli offset, relativi all'indirizzo base delle funzioni di apertura e chiusura della libreria.

- Per i nomi dei flag. Esempi sono NT_MEMORY e NT_UNKNOWN, usati nella definizione di un nodo. Lo stato dei flag viene rilevato e usato dalle funzioni di libreria dell'Amiga per decidere l'esatto comportamento da tenere. Altri esempi sono i flag HIRES e LACE previsti dalle strutture ViewPort e View della libreria Graphics.

- Per i nomi delle macro-istruzioni Assembly (ma con qualche eccezione, per rispettare la grafia adottata nei file INCLUDE). Esempi sono le macro ADDHEAD e ADDTAIL di gestione delle liste e le macro istruzioni in C ON_DISPLAY e OFF_DISPLAY di accesso ai registri hardware, generalmente usate dal sistema per attivare e disattivare la gestione del monitor.

- In alcuni casi, i caratteri maiuscoli sono usati per i nomi di macro-istruzioni associate ad alcune librerie di sistema. Per la libreria Graphics ritroviamo quattro esempi di questo tipo: CINIT, CEND, CMOVE e CWAIT. Queste quattro macro-istruzioni sono in maiuscolo perché sono strettamente collegate alle istruzioni di basso livello del coprocessore Copper.

- In accordo con quanto definito nei file INCLUDE exec/types.h ed exec/types.i, i tipi di variabile in linguaggio C e Assembly definiti appositamente per l'Amiga, come UBYTE e UWORD, sono interamente in caratteri maiuscoli. Nella programmazione dell'Amiga sono di uso frequentissimo, specialmente nella definizione delle strutture all'interno dei file INCLUDE.

Ricordiamo infine che i file INCLUDE si trovano sul disco del compilatore che avete o che intendete adottare, in genere all'interno della directory INCLUDE. Avendo il disco, è possibile stampare i contenuti dei file INCLUDE associati a ciascuna libreria. L'analisi di questi file aiuta a comprendere il meccanismo delle liste di sistema, le strutture, i parametri di struttura, i collegamenti fra le strutture, le macro-istruzioni, i tipi di variabili e altri elementi usati dalle macro e dalle funzioni di libreria dell'Amiga.

Le funzioni della libreria Exec

Introduzione

Questo capitolo illustra le funzioni della libreria Exec. Si tratta di strumenti che permettono di gestire i task, i dispositivi, le librerie, le liste e gli altri componenti del sistema Amiga. Le funzioni dell'Exec appartengono a una delle seguenti 14 categorie.

- Le funzioni di gestione delle librerie: OpenLibrary, AddLibrary, CloseLibrary, MakeLibrary, RemLibrary, SumLibrary, SetFunction e MakeFunctions; per creare e gestire sia le librerie di sistema sia quelle aggiunte dai programmatori.

- Le funzioni di gestione dei task: AddTask, FindTask, RemTask e SetTaskPri; per aggiungere, individuare, rimuovere e cambiare la priorità dei task nel sistema multitasking dell'Amiga.

- Le funzioni di gestione della memoria: AddMemList, Allocate, AllocEntry, AllocMem, AllocAbs, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem; per allocare, liberare e gestire la RAM del sistema.

- Le funzioni di gestione dei dispositivi: AddDevice, CloseDevice, OpenDevice e RemDevice; per aggiungere, chiudere, aprire e rimuovere i dispositivi di I/O.

- Le funzioni di accesso ai dispositivi di I/O: CheckIO, DoIO, SendIO e WaitIO; per controllare l'invio delle richieste di I/O ai dispositivi, la loro elaborazione e restituzione da parte dei dispositivi.

- Le funzioni di gestione delle message port: AddPort, FindPort, RemPort e WaitPort; per aggiungere, individuare, rimuovere le message port, e per attendere l'arrivo di un messaggio a una message port.

- Le funzioni per la gestione dei messaggi in transito nel sistema: GetMsg, PutMsg e ReplyMsg; per accertare l'arrivo di un messaggio alla propria message port, inviare e restituire un messaggio alla reply port di un task.

- Le funzioni di gestione dei segnali: AllocSignal, SetExcept, FreeSignal, SetSignal, Signal e Wait; per allocare, gestire e liberare segnali di comunicazione.

- Le funzioni di gestione dei semafori di segnalazione: AddSemaphore, AttemptSemaphore, FindSemaphore, InitSemaphore, ObtainSemaphore, ObtainSemaphoreList, ReleaseSemaphore, ReleaseSemaphoreList e RemSemaphore; per aggiungere, gestire e rimuovere semafori di segnalazione dal sistema.

- Le funzioni relative ai semafori basati sui messaggi: Procure e Vacate.
- Le funzioni di gestione delle liste e dei loro nodi: AddHead, AddTail, Enqueue, FindName, Insert, Remove, RemHead e RemTail; per creare e gestire le liste di strutture dati necessarie ai programmi.
- Le funzioni di gestione dei moduli residenti: FindResident, InitCode e InitResident.
- Le funzioni che permettono di copiare blocchi di memoria: CopyMem e CopyMemQuick.
- Le funzioni di gestione delle routine di interrupt dei task: AddIntServer, Cause, RemIntServer e SetIntVector; per aggiungere e controllare le routine di interrupt che devono essere associate ai task.
- Le funzioni di gestione dei vettori di trap dei task: AllocTrap e FreeTrap; per specificare le routine di gestione delle trap della CPU che devono essere usate con i task.
- Le funzioni di gestione delle risorse: AddResource, OpenResource e RemResource; per aggiungere, gestire e rimuovere risorse dal sistema.
- Le funzioni di gestione diretta del microprocessore 68000: ColdReset, GetCC, SetSR, SuperState e UserState; per impartire il reset al sistema, impostare a una specifica configurazione i bit del registro di stato della CPU, attivare i modi supervisor o user della CPU.
- La funzione d'inizializzazione delle strutture dati InitStruct, che consente d'inizializzare il contenuto di una struttura dati in RAM. Questa funzione è il complemento di altre funzioni (AddTask, OpenDevice, MakeLibrary e così via...) che svolgono operazioni analoghe su specifiche strutture di dati.

risultati delle funzioni Exec

Mentre tutte le funzioni dell'Exec producono ovviamente un risultato, non tutte restituiscono un valore nel registro D0, e quindi nelle variabili alle quali è possibile eguagliarle quando si programma in C. Un esempio di queste funzioni di tipo "void" è AddDevice, che aggiunge la struttura Device di un dispositivo alla lista di sistema dei dispositivi (in questo modo il dispositivo diventa disponibile), ma non restituisce alcun valore significativo nel registro D0. Quando si usano funzioni di questo tipo, è necessario evitare di eguagliarle a variabili. Per esempio:

AddDevice (device);

Le funzioni che invece restituiscono un valore nel registro D0, in linguaggio C devono essere eguagliate a una variabile affinché quel valore non venga perso. Per esempio:

```
memBlock = Allocate (memHeader, byteSize);
```

Allocate, AllocEntry e AllocMem appartengono a questo tipo di funzioni, e restituiscono un indirizzo di memoria. Anche le funzioni AllocTrap e AllocSignal operano in questo modo, ma restituiscono rispettivamente un numero di trap e un numero di segnale.

In molti casi il valore restituito da queste funzioni è l'indirizzo di memoria di una struttura di dati. Talvolta si tratta di una struttura preesistente, come nel caso della funzione FindPort, mentre altre volte si tratta di una struttura allocata ex novo e magari anche inizializzata, come nel caso della funzione OpenWindow, che alloca e inizializza una struttura Window. Quando si tenta di capire come lavora una funzione è opportuno tener presente il significato del valore restituito. Spesso, infatti, le funzioni producono cambiamenti più sottili di quanto sembri. Studiare in modo approfondito le strutture previste dal sistema (per esempio la struttura Task e la struttura IORequest) è un buon metodo per capire a fondo come operano le funzioni che restituiscono indirizzi a strutture di dati.

In alcuni casi, i valori restituiti dalle funzioni dell'Exec costituiscono anche un codice d'errore. Per esempio, se la funzione FindPort, che deve cercare una particolare message port pubblica e restituire l'indirizzo della relativa struttura MsgPort, non riesce a trovare la message port indicata dal task, restituisce un indirizzo nullo, e questo dev'essere interpretato come una condizione d'errore. Un altro esempio è la funzione OpenDevice, che restituisce un codice d'errore diverso da zero se per qualche ragione l'apertura del dispositivo indicato non ha avuto successo.

Infine è importante tener presente che molto spesso il valore restituito da una funzione può diventare l'argomento di un'altra funzione magari complementare. Per esempio, l'indirizzo restituito dalla funzione OpenWindow è quello che dev'essere impiegato come argomento della funzione CloseWindow per chiudere la finestra. Allo stesso modo, l'indirizzo base di una libreria restituito dalla funzione OpenLibrary diventa l'argomento della funzione CloseLibrary quando il task non ha più bisogno di quella libreria e provvede a chiuderla.

Le strutture del sistema Exec

Le funzioni dell'Exec lavorano con diverse importanti strutture. Le due più importanti sono la struttura Task e la struttura IORequest. Queste due strutture sono anche denominate Task Control Block e I/O Request Block, anche se in questo libro ci riferiremo a esse utilizzando esclusivamente i loro nomi originali, quelli da impiegare nella programmazione.

Entrambe vengono definite "dinamiche", in quanto dopo l'inizializzazione

i valori di alcuni parametri vengono di volta in volta modificati per riflettere lo stato del task e della richiesta di I/O nel sistema.

La struttura Task

La definizione in C della struttura Task appare nel file INCLUDE exec/tasks.h, mentre quella in Assembly appare nel file INCLUDE exec/tasks.i. Ogni task nel sistema è definito e gestito attraverso una struttura di tipo Task. Se il task è stato creato dal sistema (come per esempio il task di input del dispositivo Input) è il sistema stesso a creare la necessaria struttura Task, mentre se il task viene aggiunto al sistema da un altro task, è quest'ultimo che deve preoccuparsi di allocare e inizializzare opportunamente la struttura Task. Si veda anche la spiegazione della funzione AddTask.

La struttura IORequest

La struttura IORequest è un particolare tipo di messaggio preconfezionato (il suo primo elemento è una struttura Message) che i task devono impiegare per inviare richieste di I/O ai dispositivi dell'Amiga. Alcuni parametri di questa struttura, che il task deve allocare per proprio conto, vengono inizializzati dalla funzione OpenDevice al momento dell'apertura del particolare dispositivo (infatti, la funzione OpenDevice richiede fra i suoi argomenti l'indirizzo di una struttura di tipo IORequest). Quando OpenDevice ha aperto il dispositivo indicato e restituisce il controllo, i parametri io_Device e io_Unit contengono gli indirizzi della struttura Device di gestione del dispositivo, e della struttura Unit di gestione dell'unità aperta. In particolare, l'indirizzo memorizzato dalla funzione nel parametro io_Device è l'indirizzo base della libreria che costituisce il dispositivo.

Una volta che il dispositivo è stato aperto e la struttura IORequest è stata parzialmente inizializzata dalla funzione OpenDevice, il task può impostare gli altri parametri per formulare una richiesta di I/O e inoltrarla al dispositivo. A sua volta, il dispositivo elabora la richiesta, ne modifica alcuni parametri per formulare una risposta, e la restituisce al mittente (generalmente il task). Come si può notare, alcuni parametri della struttura IORequest sono destinati a contenere valori in continua evoluzione.

La definizione della struttura IORequest appare nei file INCLUDE exec/io.h (per la programmazione in linguaggio C) ed exec/io.i (per la programmazione in linguaggio Assembly). Se si desiderano conoscere i dettagli della struttura IORequest, è opportuno stampare questi file ed esaminarli. Si veda anche la spiegazione della funzione OpenDevice.

La struttura List e la struttura Node

La struttura Node permette di legare insieme più strutture e costituire organizzazioni che vengono chiamate "liste". Tutte le strutture di sistema

predisposte per appartenere a una lista possiedono come primo elemento una struttura Node, la quale finisce per essere sempre la sotto-struttura di qualche altra struttura. Per via di questa funzione, e per il fatto che nel sistema Exec l'uso delle liste come organizzazione di dati è frequentissimo, la struttura Node è considerata fondamentale.

Per esempio, se si esamina la struttura Task si vedrà che essa contiene una sotto-struttura Node come primo elemento. La stessa cosa vale per le strutture List, Interrupt, Library, MemHeader, MemList, MsgPort, Message, tutte strutture predisposte per essere organizzate all'interno di liste.

Vediamo un esempio che mostri l'utilità della struttura Node nella gestione delle liste. PortList, una delle liste di sistema che si diramano dalla struttura ExecBase, è devoluta a mantenere un elenco di strutture MsgPort, cioè delle message port che sono state allocate e rese pubbliche. La funzione FindPort (che ha il compito di restituire l'indirizzo di una particolare struttura MsgPort pubblica) esamina le strutture Node concatenate nella lista PortList assumendo che ogni struttura Node sia la sotto-struttura di una struttura MsgPort. Quando identifica il nodo cercato, ne restituisce l'indirizzo. Il task riceve l'indirizzo di una struttura Node, ma essendo questa il primo elemento di una struttura MsgPort, quell'indirizzo individua in pratica una struttura di tipo MsgPort, la message port desiderata.

La Tavola 1.1 riporta i nomi delle strutture contenenti come primo elemento una struttura Node, e i file INCLUDE nei quali sono definiti.

La struttura Node permette a ciascuna struttura che la contiene di avere un nome proprio, una priorità e di essere mantenuta in una lista di strutture della stessa categoria. Per esempio, nel caso dei task il parametro ln_Pri, che indica la priorità della struttura Node a sua volta contenuta nella struttura Task, indica in effetti la priorità attribuita al task. Si tratta di un numero che può variare tra -128 e +127 e serve all'Exec per stabilire il grado di urgenza con

Nomi dei file INCLUDE dell'Exec

Struttura	Linguaggio C	Linguaggio Assembly
Node	nodes.h	nodes.i
Task	tasks.h	tasks.i
List	lists.h	lists.i
Interrupt	interrupts.h	interrupts.i
Library	libraries.h	libraries.i
MemHeader	memory.h	memory.i
MemList	memory.h	memory.i
MsgPort	ports.h	ports.i
Message	ports.h	ports.i

Tavola 1.1: *Strutture di sistema e relativi file INCLUDE*

cui il task deve ricevere il controllo della CPU nella gestione multitasking del sistema. La struttura Node viene approfondita nella spiegazione della funzione AddHead.

La struttura List viene usata per definire una lista di strutture presente in memoria. I suoi parametri definiscono l'inizio della lista (l'indirizzo della prima struttura Node), la fine della lista (l'indirizzo dell'ultima struttura Node) e il tipo di lista. Per una descrizione della struttura List si veda la spiegazione della funzione AddTail.

Ogni volta che in questo capitolo si leggono le parole nodo e lista, si pensi alle strutture Node e List e al modo in cui queste due strutture sono usate per concatenare altre strutture in una lista.

La struttura Interrupt è usata per definire e concatenare in una lista le routine di servizio di particolari interrupt. Il primo elemento della struttura Interrupt è una sotto-struttura Node, la quale consente al sistema di mantenere le strutture Interrupt all'interno di una lista, disponendole secondo le loro priorità.

La struttura Library serve per definire le librerie. Anche in questa struttura il primo elemento è una sotto-struttura Node che consente al sistema di riunire in una lista le definizioni delle librerie disponibili nel sistema. Ogni volta che si aggiunge una libreria al sistema (tramite la funzione AddLibrary), questa lista viene estesa con l'aggiunta di un nuovo nodo. Questo discorso vale anche per le strutture Device e per le strutture di gestione delle risorse di sistema, in quanto tutte possiedono come primo elemento una struttura Library, e quindi una struttura Node.

La struttura MemHeader serve per concatenare i riferimenti ad allocazioni di blocchi di memoria quando si vuole predisporre una lista di memoria libera che sarà gestita dal programma anziché dal sistema. Allo stesso scopo viene impiegata anche la struttura MemList.

La struttura MsgPort viene impiegata per allocare nel sistema le message port.

La struttura Message viene impiegata per definire un messaggio. Più precisamente, costituisce l'intestazione di qualunque messaggio venga scambiato nel sistema. Dal momento che lo scambio dei messaggi avviene tramite liste nelle quali vengono accodati tutti i messaggi inviati a ogni message port, la struttura Message contiene anch'essa una sotto-struttura Node come primo parametro.

Altre strutture

Ci sono diverse altre strutture previste dal sistema Exec delle quali vale la pena di stampare e studiare i corrispondenti file INCLUDE. Si tratta di strutture delle quali i task devono spesso servirsi quando desiderano gestire la memoria, gli interrupt, i dispositivi, e interagire con la libreria Exec. Queste strutture e i nomi dei relativi file INCLUDE sono elencate nella Tavola 1.2.

Nomi dei file INCLUDE dell'Exec

Struttura	Linguaggio C	Linguaggio Assembly
ExecBase	execbase.h	execbase.i
IntVector	interrupts.h	interrupts.i
SoftIntList	interrupts.h	interrupts.i
MemChunk	memory.h	memory.i
MemEntry	memory.h	memory.i
Unit	devices.h	devices.i
IORequest	io.h	io.i
IOStdReq	io.h	io.i
Resident	resident.h	resident.i

Tavola 1.2: *Altre strutture dell'Exec e i file INCLUDE che le definiscono*

Task, message port, messaggi e segnali

Mettere i task in grado di comunicare tra loro può sembrare complesso. Tuttavia, se si spezzetta il processo in piccoli passi, si capisce facilmente in che modo message port, messaggi e segnali vengono usati per passare informazioni da un task all'altro e per consentire a un task di dirigere le azioni dell'altro.

Prima di tutto è importante capire che le message port e i messaggi non sono altro che strutture del sistema Amiga. Inoltre, i segnali non sono altro che bit in una long word (due word per un totale di 32 bit). Per mettere a fuoco queste idee, vale la pena di esaminare per prima cosa le strutture MsgPort e Message, che giocano un ruolo rilevante nella comunicazione inter-task e task-dispositivo. La definizione della struttura MsgPort è la seguente:

```
struct MsgPort {
    struct Node mp_Node;
    UBYTE mp_Flags;
    UBYTE mp_SigBit;
    struct Task *mp_SigTask;
    struct List mp_MsgList;
};
```

Il significato dei parametri nella struttura MsgPort è il seguente:

- il parametro mp_Node è una sotto-struttura Node che viene usata per aggiungere una message port alla lista delle message port pubbliche.

- Il parametro mp_Flags contiene i flag della struttura MsgPort. Si può impostare questo parametro con uno dei seguenti tre valori: PA_SIGNAL, PA_SOFTINT o PA_IGNORE. Impostandolo a PA_SIGNAL si richiede che il task venga avvisato con un segnale dell'arrivo di un messaggio alla message port. Il task può poi reagire come preferisce. Impostandolo a PA_SOFTINT, all'arrivo del messaggio viene inviato un segnale al task che ha creato la message port e viene causato un interrupt software, con conseguente esecuzione della relativa routine di servizio (il task perde temporaneamente il controllo della CPU). Infine, impostandolo a PA_IGNORE, al task non viene inviato alcun segnale e non viene compiuta alcuna azione. È importante comprendere che queste attività (o assenze di attività), si verificano quando giunge un messaggio alla message port, indipendentemente dalla sua origine.

- Il parametro mp_SigBit deve contenere il numero del segnale da inviare al task quando giunge un messaggio alla message port. In pratica, contiene il numero del bit di segnale che all'arrivo del messaggio dev'essere impostato a 1 (invio del segnale) nel parametro tc_SigRecvd della struttura Task, il cui indirizzo è indicato nel parametro mp_SigTask della struttura MsgPort. È importante capire che ogni struttura MsgPort può essere associata a un solo bit di segnale, cioè può provocare l'invio di un segnale a un solo task. Infine, a prescindere dal contenuto di questo parametro, l'azione intrapresa dal sistema quando nella message port giunge un messaggio è sempre e solo determinata dal contenuto del parametro mp_Flags.

- Il parametro mp_SigTask deve contenere l'indirizzo della struttura Task relativa al task che deve ricevere il segnale indicato dal parametro mp_SigBit quando giunge un messaggio alla message port. Questo task può essere qualsiasi task del sistema, compreso quello che ha creato la message port.

- Il parametro mp_MsgList è una sotto-struttura List che viene usata dal sistema per accodare in una lista tutti i messaggi in arrivo a questa message port. Si tratta di una lista gestita in modo FIFO (First In, First Out) nella quale i messaggi pervenuti risalgono a mano a mano che i messaggi in cima vengono estratti. Generalmente, lo svuotamento di una coda a una message port viene condotto dal task che ha creato quella message port.

La struttura Message è definita come segue:

```
struct Message {
    struct Node mn_Node;
    struct MsgPort *mn_ReplyPort;
    UWORD mn_Length;
};
```

Il significato dei parametri nella struttura Message è il seguente:

- il parametro mn_Node è la sotto-struttura Node usata per mantenere il messaggio all'interno delle liste, cioè all'interno delle code alle message port presenti nel sistema.

- Il parametro mn_ReplyPort contiene l'indirizzo della message port alla quale il messaggio dev'essere restituito una volta giunto a destinazione. Questa message port è il "mittente" del messaggio, ed è indispensabile che ogni messaggio ricevuto sia restituito al mittente. Una volta che il task ricevente ha elaborato il messaggio, deve chiamare la funzione ReplyMsg, la quale restituisce il messaggio al mittente indicato nel parametro mn_ReplyPort. Generalmente i task che inoltrano messaggi nel sistema predispongono alcune message port adibite a ricevere i messaggi rispediti al mittente. Queste message port vengono chiamate reply port, e generalmente vengono create e gestite dai task nei quali i messaggi hanno avuto origine; tuttavia, la reply port indicata in un messaggio può essere qualsiasi message port presente nel sistema, anche appartenente a un task diverso da quello che ha inviato il messaggio.

- Il parametro mn_Length contiene la lunghezza in byte del messaggio intestato dalla struttura Message, cioè la quantità di byte che segue la struttura in memoria. Questi byte costituiscono il messaggio vero e proprio, e generalmente sono i parametri di una struttura il cui primo elemento è una struttura Message, come avviene per la struttura IORequest.

Se si studia la definizione dei parametri in queste strutture si inizia a comprendere come vengono scambiati i messaggi tra i task. Ecco i punti da tenere presenti:

- il sistema può gestire qualsiasi numero di task. Ciascun task è definito da una struttura Task (si veda la spiegazione della funzione AddTask).

- Ciascun task ha a disposizione un massimo di 32 segnali, cioè può allocare fino a 32 segnali. Ciascun segnale è rappresentato da un bit di segnale del parametro tc_SigAlloc della struttura Task: i bit di segnale impostati sono quelli relativi ai segnali allocati dal task. Si possono impiegare 16 di questi segnali (i bit di segnale da 16 a 31) per mettere il task in condizione di dialogare con un altro task. I rimanenti 16 (i bit di segnale da 0 a 15) sono riservati al sistema.

- Qualsiasi task può creare e utilizzare un qualsiasi numero di message port, mentre può servirsi soltanto di 16 message port predisposte per l'invio di segnali (un numero pari ai bit di segnali di cui dispone); comunque, uno stesso bit di segnale può anche essere assegnato a diverse message port. Ogni volta che il task alloca una struttura

MsgPort in memoria, in pratica crea una message port. Il task ha la facoltà di scegliere se rendere pubblica o meno la message port. Renderla pubblica significa mettere il sistema a conoscenza della sua esistenza, dotarla di un nome e vederla elencata all'interno della lista di sistema PortList. Diventando pubblica, la message port può essere sempre individuata tramite il suo nome da qualsiasi task nel sistema. Ogni message port deve sempre appartenere a uno e un solo task, quello che l'ha creata.

- Ciascuna message port può causare l'invio di segnali a un solo particolare task, dal momento che a essa è associato un solo bit di segnale nella struttura Task del task. È importante capire che questo può essere il task a cui appartiene la message port, ma può anche essere qualsiasi altro task definito nel sistema. Il task che riceve il segnale è definito attraverso il parametro mp_SigTask della struttura MsgPort.

- Ogni volta che una message port predisposta per l'invio di segnali riceve un messaggio nella sua coda (inoltrato dalla funzione PutMsg o ReplyMsg), invia al task sempre lo stesso segnale, cioè causa l'impostazione a 1 sempre dello stesso bit nel parametro tc_SigRecvd della struttura Task. Il task avvertito vede sempre lo stesso segnale, qualunque sia l'origine del messaggio. Le strutture Message non contengono di per sé alcuna informazione per definire i segnali; è l'arrivo dei messaggi alle code delle message port che provoca i segnali. Questo è un concetto importante. Un task, quando riottiene il controllo (dopo essere entrato in attesa) può rilevare il numero di segnale pervenuto, e stabilirne la provenienza. Stabilire cioè quale delle sue message port ha ricevuto il messaggio.

- I messaggi vengono collocati nelle code alle message port dalla funzione PutMsg (o ReplyMsg). Si dice che PutMsg "invia un messaggio a un destinatario", e questo corrisponde a inserire quel messaggio nella coda alla message port indicata come argomento della funzione. Allo stesso modo, ReplyMsg "restituisce un messaggio al mittente", e questo corrisponde a inserire quel messaggio nella coda alla reply port il cui indirizzo è indicato nel parametro mn_ReplyPort della struttura Message che intesta il messaggio. Si noti che il destinatario di un messaggio viene indicato semplicemente come argomento della funzione PutMsg, mentre il mittente è indicato nell'intestazione del messaggio.

- Lo scambio dei messaggi avviene inserendoli all'interno delle code alle message port. Questo inserimento comporta solo l'aggiornamento di due puntatori all'interno della sotto-struttura Node della struttura Message affinché venga instaurata la concatenazione con gli altri nodi della coda. La struttura Message non viene mai copiata o spostata dalla RAM in cui è stata originariamente allocata, anche se si parla di "spedizione" dei messaggi, "transito" dei messaggi, e "ascesa" dei

messaggi all'interno delle code. Consapevoli del fatto che in realtà nessuna struttura dell'Amiga, e neanche i messaggi, vengono spostati, adotteremo anche noi queste espressioni nel corso del libro.

- L'iter tipico di un messaggio all'interno del sistema prevede le seguenti fasi. Il mittente invia il messaggio al destinatario tramite la funzione PutMsg. A questo punto il messaggio non gli appartiene più. Il destinatario viene avvisato con un segnale che un nuovo messaggio è giunto alla coda della sua message port, e che deve provvedere a estrarlo tramite la funzione GetMsg o Remove. Accede al contenuto del messaggio, eventualmente modificandolo per formulare una risposta, e lo restituisce al mittente tramite la funzione ReplyMsg. D'ora in poi deve considerare che il messaggio non è più di sua proprietà. Il mittente riceve nella sua reply port il messaggio che aveva originariamente inviato, e solo in quel momento ha la conferma che il messaggio è giunto a destinazione e il destinatario l'ha esaminato. In tutto questo "viaggio" all'interno del sistema, il messaggio non si è mai mosso dalle locazioni RAM nelle quali era stato originariamente allocato.

- I messaggi vengono estratti dalle code attraverso la funzione GetMsg, la quale si aspetta come argomento l'indirizzo di una message port. La funzione rimuove il messaggio in cima alla coda e ne restituisce l'indirizzo; se la coda è vuota restituisce un indirizzo nullo. Se si dispone dell'indirizzo del messaggio prima ancora che pervenga (perché siamo noi ad averlo mandato), quando giunge un segnale alla propria reply port si può chiamare la funzione CheckIO, la quale verifica che il messaggio indicato come argomento sia stato restituito. Se l'esito è positivo, il task può estrarlo dalla coda alla reply port senza attendere che giunga in cima, chiamando la funzione Remove. Questa funzione, infatti, si aspetta come argomento l'indirizzo di un nodo, e lo rimuove dalla coda qualsiasi sia la sua posizione.

- Ogni volta che una message port di un task predisposta per l'invio di segnali riceve un messaggio, causa l'invio di un segnale al task indicato dal parametro mp_SigTask della struttura MsgPort. Se questo task era entrato in attesa del segnale tramite la funzione Wait o la funzione WaitPort, ricevendo il segnale riottiene il controllo. È importante ricordare che il segnale viene inviato quando il messaggio arriva nella coda alla message port, e non quando giunge alla sua sommità.

Due task, due message port, due reply port e quattro segnali

La Figura 1.1 mostra una situazione abbastanza frequente, nella quale lo scambio di messaggi avviene fra due task, Task1 e Task2. Ciascun task possiede una message port, una reply port, e due segnali assegnati (uno per ciascuna message port). Questi task rappresentano una qualunque coppia di task nel sistema. Per esempio, quando si interagisce con Intuition, le message

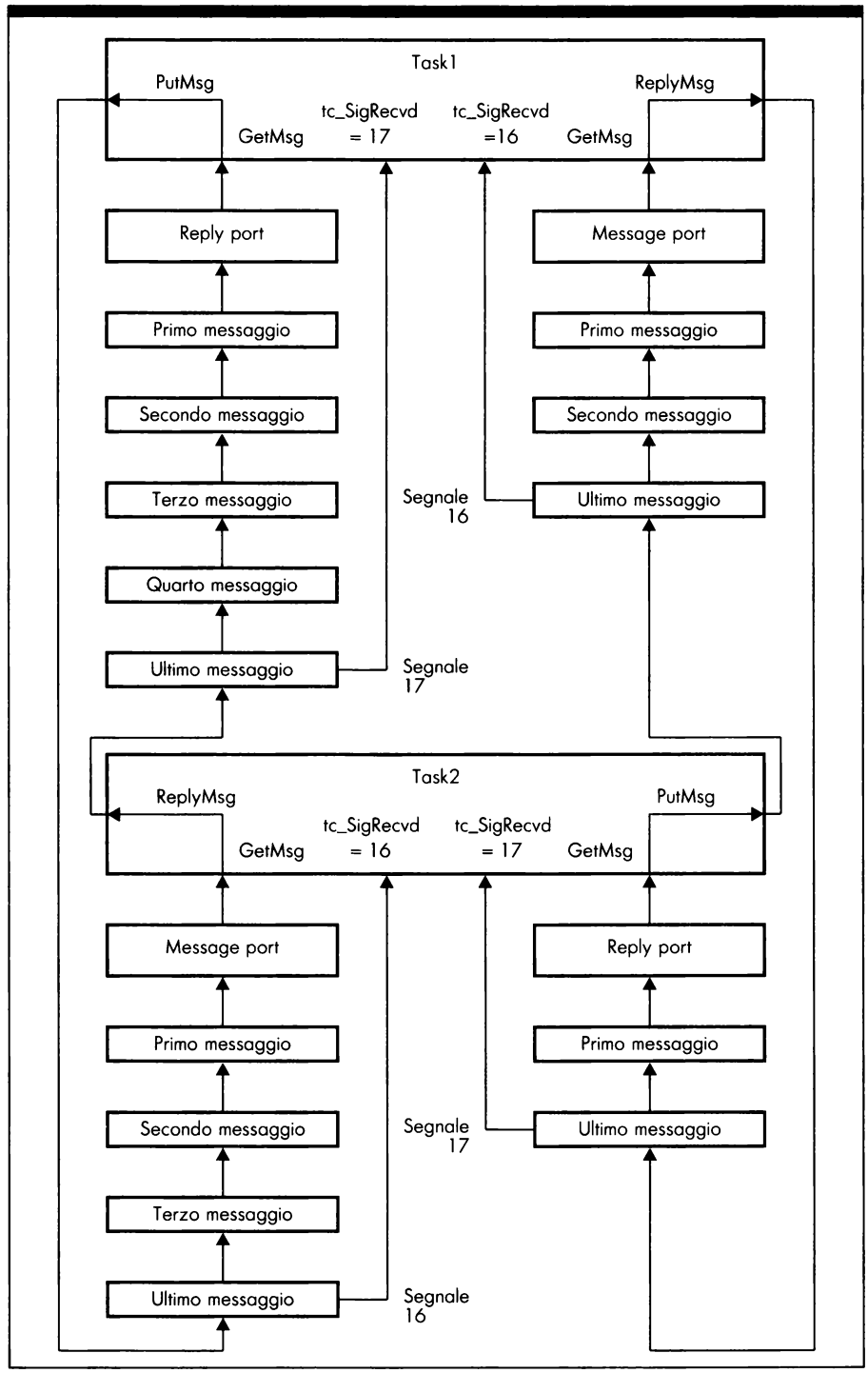


Figura 1.1: *Task, message port, messaggi e segnali*

port associate alle finestre definite dall'utente vengono chiamate user port. Sono le message port nelle quali i task si aspettano di ricevere messaggi da Intuition. Le message port di Intuition sono chiamate window port. Queste sono le message port nelle quali Intuition si aspetta di ricevere messaggi dalla particolare finestra (si veda il capitolo 6).

Nella Figura 1.1, le funzioni PutMsg, ReplyMsg e GetMsg vengono usate per muovere i messaggi dentro e fuori le liste mantenute per ciascuna delle code alle message port.

In aggiunta, ciascun task ha assegnato due segnali, uno alla message port e uno alla reply port. Per entrambi i task sono stati usati i bit di segnale 16 e 17, ma questo non crea conflitti dal momento che ogni task può disporre dei suoi bit di segnale senza interferire con quelli di altri task.

Ecco un tipico scambio: Task1 invia un messaggio a Task2, e quest'ultimo lo restituisce al mittente, cioè a Task1.

1. Un task, Task1, deve inoltrare un messaggio a Task2. Per farlo deve prima di tutto creare il messaggio in memoria, allocando una quantità di byte pari alla dimensione della struttura Message più il messaggio vero e proprio (per rendere questa operazione di allocazione più semplice, il programmatore può definire una struttura nella quale il primo parametro sia una sotto-struttura Message). Successivamente deve memorizzare nel parametro mn_ReplyPort della struttura Message l'indirizzo della sua reply port, cioè la message port nella quale si aspetta di ricevere il messaggio quando il destinatario l'ha elaborato e lo restituisce al mittente.

2. Task1 deve conoscere l'indirizzo della message port di Task2, il destinatario del messaggio. Per ottenerlo, generalmente chiama la funzione FindPort indicando come argomento il nome della message port di Task2. Poi, Task1 può chiamare la funzione PutMsg, indicando come argomenti l'indirizzo del destinatario (la message port di Task2) e l'indirizzo del messaggio da inviare. Questa operazione corrisponde all'invio del messaggio. La funzione lo inserisce in fondo alla coda alla message port di Task2, e questo comporta l'invio di un segnale a Task2. Da questo momento, Task1 non deve più accedere al messaggio, sebbene ne detenga ancora l'indirizzo. Il messaggio è infatti diventato di proprietà del destinatario.

3. Task2, che presumibilmente è in attesa di quel segnale dalla sua message port, viene avvisato e chiama la funzione GetMsg (rimuove il messaggio in cima alla coda). Supponendo che prima la coda fosse vuota, l'indirizzo che ottiene da GetMsg è quello del messaggio inviato da Task1, cioè quello che ha causato l'invio del segnale. Ora può copiarne i parametri nelle sue variabili interne, eventualmente modificarlo per formulare la risposta, e chiamare la funzione ReplyMsg indicando come argomento l'indirizzo del messaggio ricevuto. Questa operazione corrisponde alla restituzione del messaggio al mittente. ReplyMsg accede al parametro mn_ReplyPort della struttura Message,

ne estrae l'indirizzo della reply port alla quale il messaggio dev'essere restituito, e inserisce il messaggio in fondo alla coda a quella reply port (si noti che se questo indirizzo è nullo il messaggio non viene restituito). Da questo momento Task2 non deve più accedere al messaggio. Il messaggio è infatti tornato di proprietà del mittente.

4. Task1 riceve un segnale e svuota la coda alla sua reply port riottenendo l'indirizzo del messaggio che aveva inviato. Solo in quel momento ha la certezza che il messaggio sia giunto a destinazione e può riutilizzarne

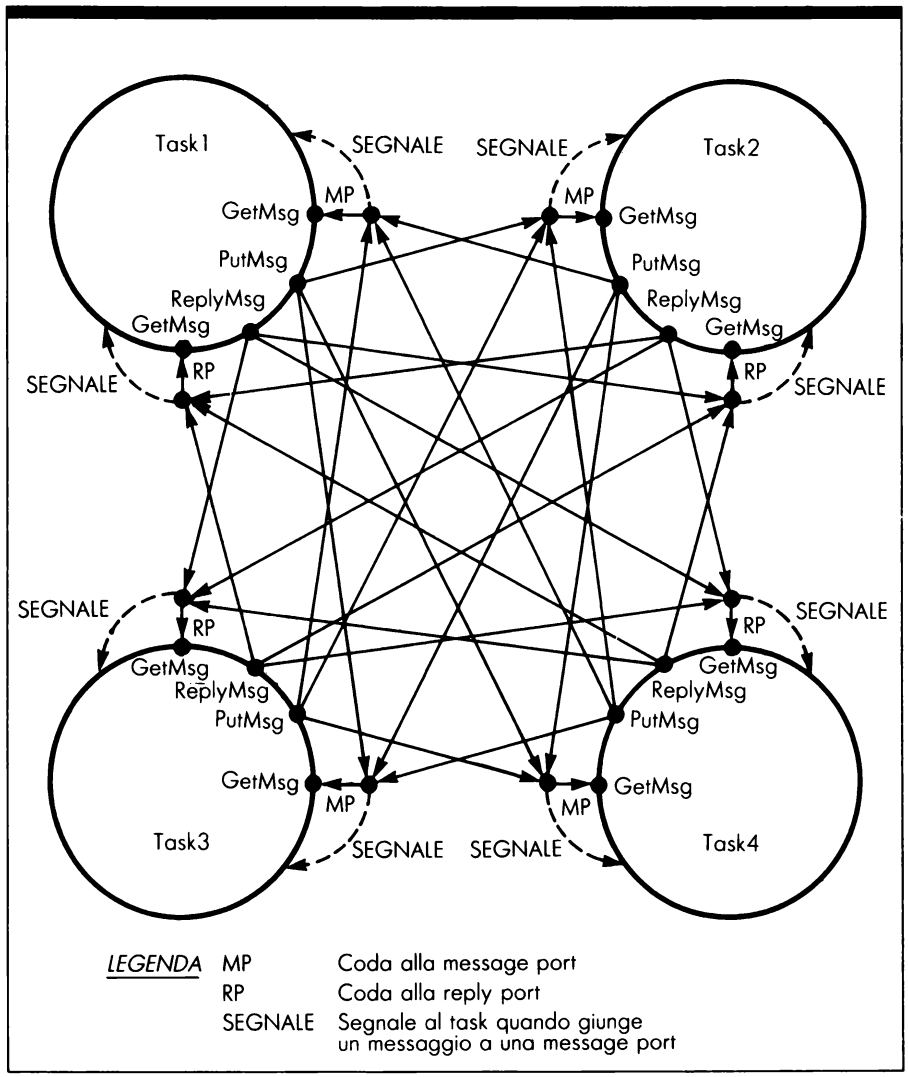


Figura 1.2: *Diversi task, message port, messaggi, e segnali*

lo spazio di memoria, liberandolo o formulando un altro messaggio.

Questa stessa situazione può essere anche rovesciata, in modo che sia Task2 a inviare un messaggio a Task1, e sia quest'ultimo a restituirlo al mittente, dal momento che anche Task2 possiede una reply port, e anche Task1 possiede una message port.

Una situazione più complessa

La Figura 1.2 illustra tutte le possibili connessioni in una situazione a quattro task. Ciascun task ha una message port e una reply port. Ciascun task può inviare messaggi a ognuna delle altre tre message port. Inoltre, ciascun task può inviare messaggi di risposta verso le reply port dei mittenti. Come si può vedere, ciò comporta molte possibili linee di comunicazione.

Ciascuna message port è predisposta per inviare un segnale al task che l'ha creata. Questo vale anche quando la message port è adibita a reply port.

Ciascuno dei quattro task può condividere gli stessi bit di segnale. I messaggi in ciascuna message port (oppure reply port) formano una coda FIFO e si spostano progressivamente verso la cima a mano a mano che vengono prelevati.

La Figura 1.2 rappresenta una situazione assai frequente. Comunque è probabile che nella maggior parte dei casi non saranno necessarie tutte le connessioni (e le istruzioni di programmazione associate) rappresentate da questo diagramma.

AbortIO

Sintassi di chiamata della funzione

```
error = AbortIO (iORequest)
D0              A1
```

Scopo della funzione

Questa funzione ordina a un dispositivo di eliminare una richiesta di I/O. Il dispositivo esegue la richiesta ricorrendo all'omonima funzione interna che, come BeginIO, è comune a tutti i dispositivi. Se la funzione ha successo, il dispositivo interrompe l'elaborazione della richiesta di I/O e la restituisce immediatamente alla reply port del task.

Argomenti della funzione

iORequest Puntatore alla struttura IORequest relativa alla richiesta di I/O da abortire.

Discussione

A differenza di BeginIO, che è una funzione interna di tutti i dispositivi ma che può essere chiamata direttamente dai task, la funzione AbortIO è una funzione dell'Exec; si limita però a chiamare l'omonima funzione interna del dispositivo, che a sua volta si occupa di abortire una specifica richiesta di I/O.

Si noti che AbortIO restituisce la struttura IORequest alla reply port del task, ma non la rimuove. Inoltre, non mette automaticamente il task in attesa: è quindi il task che deve controllare la restituzione della richiesta nella sua reply port, prima di poter riutilizzare la struttura IORequest.

Se al momento della chiamata ad AbortIO la richiesta di I/O era già stata elaborata, non viene compiuta nessuna azione. Se per qualche ragione (dipendente dallo stato del dispositivo) la richiesta di I/O non può essere abortita, la funzione AbortIO restituisce in D0 un codice d'errore; in caso di successo, invece, restituisce uno 0.

Si vedano anche le spiegazioni relative alle funzioni WaitIO, DoIO, SendIO e CheckIO.

AddDevice

Sintassi di chiamata della funzione

AddDevice (device)
A1

Scopo della funzione

Questa funzione aggiunge un nuovo dispositivo alla lista di sistema mantenuta dall'Exec, la lista DeviceList. Da questo momento il dispositivo è disponibile per ogni task che voglia aprirlo, cioè servirsene (con il termine "disponibile" intendiamo che il dispositivo si trova in memoria ed è pronto per essere impiegato, ma non è detto che qualcuno lo stia impiegando). Si noti che questa funzione non fa altro che aggiungere un nodo alla lista di sistema dei

dispositivi; questo nodo è la sotto-struttura Node presente come primo elemento della struttura Library, a sua volta unico elemento della struttura Device. Prima di chiamare AddDevice, occorre che il dispositivo sia già in memoria e pronto per diventare operativo. In genere AddDevice viene impiegata dai codici d'installazione dei dispositivi, e non dai task. Questi ultimi, infatti, hanno a loro disposizione la funzione OpenDevice, che in maniera trasparente rende disponibile il dispositivo se non lo è già (dispositivo residente su disco) e lo apre.

Argomenti della funzione

device Indirizzo di una struttura Device opportunamente inizializzata per definire il dispositivo da rendere disponibile.

Discussione

La libreria Exec dispone di due funzioni per rendere disponibile e rimuovere dal sistema un dispositivo: AddDevice e RemDevice.

La lista di sistema DeviceList

Il sistema, tramite la struttura ExecBase, mantiene una lista di tutti i dispositivi disponibili nel sistema, la lista DeviceList. Ogni volta che si effettua una chiamata alla funzione AddDevice, questa lista viene aggiornata per riflettere la situazione creata dall'aggiunta di un nuovo dispositivo al sistema.

Ogni dispositivo aggiunto impiega certe risorse del sistema, in particolare la memoria. Un dispositivo richiede generalmente un certo quantitativo di memoria, a seconda delle esigenze di buffer delle sue unità. Perciò, a meno che un dispositivo non sia essenziale per la prosecuzione di un certo compito, è sempre meglio chiuderlo con una chiamata alla funzione CloseDevice quando non lo si deve più impiegare. La funzione CloseDevice, che serve per chiudere una particolare unità di un dispositivo, restituisce al sistema quanta più memoria è possibile. La quantità di memoria che viene liberata dipende dallo stato in cui si trova il dispositivo. Se dispone di una sola unità e questa è stata aperta solo da un task, chiamando la funzione CloseDevice quel task causa la completa chiusura del dispositivo. In questo caso viene liberata la maggior quantità di memoria possibile, ma il dispositivo continua a rimanere disponibile nel sistema; la memoria che occupa con la sua libreria non viene liberata, e la sua struttura Device continua a far parte della lista di sistema DeviceList. Per liberare anche la memoria occupata dalla libreria del dispositivo, cioè rimuovere il dispositivo dal sistema, occorre chiamare la funzione RemDevice, la quale rimuove l'intero dispositivo se non risulta aperto

da nessun task. Questa operazione di rimozione può essere compiuta solo sui dispositivi non residenti, cioè quelli che risiedono nel disco sistema. Una volta rimosso, il dispositivo dev'essere nuovamente caricato da disco per tornare disponibile, operazione che viene svolta dalla funzione OpenDevice.

Sequenza di gestione del dispositivo

La normale gestione di un dispositivo procede seguendo queste linee.

1. Se il dispositivo è di quelli residenti nella ROM della macchina, è già stato reso disponibile durante il boot, e compare sempre nella lista di sistema DeviceList. Se invece è un dispositivo residente su disco, viene caricato in memoria quando qualcuno richiede l'accesso chiamando la funzione OpenDevice. In questo caso, quando la libreria del dispositivo è stata caricata in memoria, il sistema chiama la funzione AddDevice per renderlo disponibile. Questa funzione si comporta come AddLibrary, AddResource e altre funzioni dell'Exec che aggiungono alle liste di sistema le risorse software in questione, rendendole disponibili. A questo punto, la memoria occupata dal dispositivo è solo quella occupata dalla sua libreria. Una volta che il dispositivo è stato aggiunto, può essere utilizzato da ciascun task del sistema multitasking dell'Amiga, tanto dai task specifici dell'utente, quanto dai task di sistema.

2. Ciascun task che voglia usare questo dispositivo deve aprirlo con una chiamata alla funzione OpenDevice. Lo scopo della funzione OpenDevice è simile a quello di OpenLibrary e OpenResource della libreria Exec, che servono per ottenere l'accesso alle librerie e alle risorse del sistema. Dato che il sistema è multitasking, diversi task possono usare lo stesso dispositivo, ciascuno chiamando la funzione OpenDevice. La funzione OpenDevice è assai efficiente, dal momento che se il dispositivo non risulta disponibile si preoccupa di caricarlo da disco senza bisogno che il task lo richieda esplicitamente.

In qualsiasi momento ci possono essere diversi task nel sistema che hanno chiamato OpenDevice per aprire lo stesso dispositivo. In ogni momento, però, è sempre uno e solo uno il task in esecuzione. Ogni chiamata alla funzione OpenDevice permette a un task di accedere al dispositivo, ed è il dispositivo a tener conto del numero di task che hanno accesso alla sua libreria e alle sue unità. Questo dispositivo diventa così una risorsa *condivisa* nel sistema: i dati giungono e partono da esso nella direzione di un solo task (quello in esecuzione), ma anche altri task possono in seguito usare lo stesso dispositivo per inviare e ricevere dati.

La libreria del dispositivo

L'organizzazione della libreria di gestione di un dispositivo è molto simile a quella delle librerie software dell'Amiga, come Intuition e Graphics. In effetti, potremmo dire che i dispositivi sono particolari librerie software, e come tali vengono gestiti. Al pari delle librerie, occorre aprirli, richiuderli, e devono essere resi disponibili quando non lo sono. Come qualsiasi altra, la libreria del dispositivo possiede una serie di vettori di salto (la tavola dei vettori di salto) che precede in memoria l'indirizzo base della libreria, seguita dalla struttura Device di gestione del dispositivo (che è uguale alla struttura Library delle librerie software), e dai codici che costituiscono le funzioni del dispositivo vere e proprie. Nella tavola dei vettori di salto i primi sono quelli che individuano le funzioni tipiche di ogni libreria: le funzioni OPEN, CLOSE, EXPUNGE ed EXTFUNC. Oltre a questi vettori ve ne sono sempre altri quattro riservati a usi futuri.

La libreria di un dispositivo contiene inoltre altri vettori di salto specifici dei dispositivi, che individuano le routine BeginIO, AbortIO, Read, Write, Reset, Update, Clear, Flush, Stop e Start. Infine, ciascun dispositivo può avere un certo numero di vettori di salto addizionali per individuare le sue routine interne di gestione.

I programmatori possono aggiungere al sistema dispositivi di I/O di propria creazione. Per farlo, devono creare file su disco conformi a un particolare tipo di struttura. Si veda la descrizione della funzione MakeLibrary per i dettagli sullo sviluppo della libreria di un dispositivo.

Uso dei dispositivi

Una volta chiamata la funzione OpenDevice, l'interazione con il dispositivo avviene inviando quelle che vengono chiamate richieste di I/O. Si tratta di particolari messaggi con una struttura predefinita che varia a seconda del dispositivo. Alcuni parametri di questa struttura vengono inizializzati dalla funzione OpenDevice quando apre il dispositivo e la relativa unità. Per quanto la struttura delle richieste di I/O possa differire da dispositivo a dispositivo, inizia sempre con la sotto-struttura IORequest, che rappresenta quindi il messaggio base standard per tutti i dispositivi. Tramite le richieste di I/O, i task comunicano con il dispositivo. Le richieste possono essere molto diverse fra loro, ma devono adattarsi al dispositivo. Possono essere richieste di invio dati, lettura dati, aggiornamento dei buffer... Generalmente, il dispositivo deve restituire la richiesta di I/O al task quando ha finito di elaborarla, magari dopo averne modificato alcuni parametri per formulare una risposta (più avanti vedremo che esiste un'eccezione a questo comportamento).

Il task che ha inviato la richiesta aspetta che il dispositivo la restituisca in risposta, esattamente come farebbe inviando un messaggio a un altro task nel sistema. Rispetto allo scambio di messaggi tra task però, le comunicazioni con i dispositivi permettono anche di accelerare le cose richiedendo al dispositivo di trattare la richiesta nella modalità QuickIO. Come vedremo, se il dispositivo accoglie la richiesta e può trattarla nel modo "quick", non la restituisce.

I parametri delle richieste di I/O

Diversi parametri della struttura IORequest meritano una discussione.

Il parametro io_Device deve contenere l'indirizzo base del dispositivo, cioè l'indirizzo della sua struttura Device. Solo grazie a questa indicazione il sistema è in grado di sapere a quale dispositivo è diretta la richiesta di I/O. Questo parametro è sempre inizializzato dalla funzione OpenDevice.

Il parametro io_Unit generalmente deve contenere un'indicazione dell'unità del dispositivo alla quale è indirizzata la richiesta di I/O. Può trattarsi dell'indirizzo della struttura Unit di gestione dell'unità, di un valore numerico... o può anche restare inutilizzato (con i dispositivi che possiedono una sola unità). Anche questo parametro viene inizializzato dalla funzione OpenDevice.

Il parametro io_Command dev'essere inizializzato dal task con il codice del comando che il dispositivo deve eseguire. Questo parametro determina lo scopo della richiesta di I/O. Ogni richiesta deve indicare un comando da eseguire, e il comando dev'essere ovviamente compreso fra quelli che il dispositivo è in grado di eseguire. Com'è ampiamente illustrato in *Programmare l'Amiga Volume II*, i dispositivi prevedono comandi standard e comandi specifici.

Il parametro io_Flags è composto da un insieme di flag. Alcuni sono da considerare "privati" e vengono utilizzati dal dispositivo per le sue operazioni interne, mentre altri sono sotto il controllo del task. Il più importante è il flag IOF_QUICK, che se viene impostato prima di una chiamata a BeginIO (DoIO lo imposta automaticamente mentre SendIO lo azzera automaticamente) informa il dispositivo che se le condizioni lo permettono questa richiesta dev'essere elaborata nella modalità di accesso veloce (QuickIO). Allora il dispositivo (se può) elabora subito la richiesta e non la restituisce al mittente, ma lascia il flag IOF_QUICK impostato; se invece non può accordare il QuickIO, azzera il flag IOF_QUICK ed elabora la richiesta come una qualunque richiesta accodata. Quando il task riottiene il controllo da BeginIO, può rendersi conto se il QuickIO ha avuto successo controllando lo stato del flag IOF_QUICK della richiesta di I/O.

Il parametro io_Error viene sempre azzerato dal dispositivo se non sorgono condizioni d'errore, altrimenti viene inizializzato con l'appropriato codice d'errore. Se il task rileva che io_Error contiene un valore diverso da zero, deve desumerne che si è verificata una condizione d'errore. Per esempio manca il disco nel disk drive, l'informazione nel buffer del task non ha senso o il codice del comando da eseguire non è fra quelli che il dispositivo può eseguire. Ogni errore dev'essere trattato in modo diverso. È possibile costruire routine di controllo degli errori e chiamarle individualmente, basandosi sul codice numerico dell'errore, per avvisare l'utente dell'accaduto e spiegargli come rimediare (inserire un disco, per esempio). Si noti che in alcuni casi la condizione di errore è soltanto apparente. I programmi devono essere in grado di discernere e di decidere di conseguenza.

Molto spesso per formulare certe richieste di I/O occorre servirsi di una struttura di I/O più estesa di IORequest: IOStdReq. In genere, questa necessità si presenta quando il comando implica un trasferimento di dati. IOStdReq aggiunge quattro parametri alla struttura IORequest.

Il parametro io_Actual viene aggiornato dal dispositivo in risposta alle

richieste che implicano trasferimenti di byte: se il trasferimento ha avuto successo, io_Actual indica il numero di byte trasferiti. Quando il task formula una richiesta di I/O per il trasferimento di una certa quantità di dati, deve memorizzare nel parametro io_Length il numero esatto di byte da trasferire. Se durante il trasferimento si verifica una condizione d'errore, il numero di byte da trasferire (io_Length) e il numero di byte effettivamente trasferiti possono differire. A volte le differenze sono causate da errori e quindi l'informazione non si trova dove dovrebbe, cioè nel buffer di memoria, sul disco e così via... In questo caso, il dispositivo restituisce un opportuno codice d'errore. Altre volte, invece, una differenza fra il contenuto dei due parametri non rappresenta una situazione critica; per esempio, se il task richiede al dispositivo Serial di ricevere 512 byte dalla porta seriale, e il dispositivo al centesimo byte rileva una condizione di EOF (End Of File), il trasferimento viene interrotto, ma evidentemente questa interruzione non è stata causata da un problema di natura elettrica sulla linea seriale, e quindi non può essere considerata una condizione d'errore. Pertanto, in un caso come questo il dispositivo non restituisce alcun codice d'errore, e il task può rendersi conto dell'accaduto confrontando il contenuto del parametro io_Length con quello del parametro io_Actual.

Il parametro io_Data dev'essere inizializzato dal task con l'indirizzo dell'area di memoria che ha predisposto come buffer. Può trattarsi di un buffer allocato per ricevere o inviare dati al dispositivo.

Infine, dev'essere impiegato il parametro io_Offset ogni qualvolta un trasferimento avviene a byte-offset, cioè per mezzo di letture o scritture successive da un buffer.

I dispositivi standard dell'Amiga

I dispositivi standard previsti dall'Amiga sono per lo più installati nella memoria ROM del sistema, e quindi sono parte integrante del sistema operativo; questo significa che sono immediatamente disponibili all'accensione della macchina. Però, per non occupare troppa memoria ROM, si è deciso che alcuni dispositivi standard, quelli d'uso meno frequente, risiedessero su disco, pronti per essere caricati in RAM all'occorrenza. Questi dispositivi non residenti sono Clipboard, Narrator, Parallel, Serial e Printer. Il fatto che risiedano su disco anziché in ROM non è comunque d'interesse per i task, dal momento che la loro particolare gestione è affidata interamente al sistema operativo.

Il dispositivo Timer è predisposto per soddisfare le necessità di temporizzazione dei task. È composto da due unità, corrispondenti a due sistemi di temporizzazione diversi. Il primo è sincronizzato con l'interrupt di vertical-blanking del video, e quindi può scandire il tempo a intervalli di 1/60 di secondo nei sistemi americani (NTSC) e di 1/50 di secondo nei sistemi europei (PAL). Il secondo usa gli interrupt prodotti da un timer del chip 8250 CIA, che si verificano a intervalli molto inferiori.

Il dispositivo TrackDisk fornisce una serie di comandi per accedere direttamente ai disk drive dell'Amiga. Si tratta del più basso livello di accesso ai disk drive che il sistema offra, molto più basso del livello d'accesso

dell'AmigaDOS. Il dispositivo TrackDisk non sa infatti cosa siano i file, le directory, le sotto-directory... È predisposto solo per leggere e scrivere dati grezzi o codificati nei settori e nelle tracce dei dischi, per effettuare controlli di stato sui disk drive e sui dischi inseriti, e per formattare i dischi. L'AmigaDOS impiega il dispositivo TrackDisk per la gestione dei file su disco, ed è per questo che nella gerarchia del sistema si trova a un livello molto alto.

Il dispositivo Keyboard gestisce gli input provenienti dalla tastiera, convertendoli in eventi di input che i task del sistema possono rilevare e interpretare. Gli eventi di input da tastiera vengono mantenuti all'interno di un buffer di tastiera perché nessuno vada perduto.

Il dispositivo Gameport gestisce gli input provenienti dalle porte giochi, convertendoli in eventi di input che i task del sistema possono rilevare e interpretare. È predisposto per riconoscere i segnali generati dal mouse, dal joystick o da qualsiasi dispositivo a potenziometro. Come nel caso della tastiera, gli eventi di input delle porte giochi vengono mantenuti all'interno di opportuni buffer. Il sistema è in grado di riconoscere quale tipo di dispositivo hardware è collegato, e a quale porta giochi, e quali sono le condizioni di trigger da soddisfare perché un input costituisca un evento di input.

Il dispositivo Input congloba in un unico flusso tutti gli eventi generati dai dispositivi Gameport, Keyboard e TrackDisk. Questo flusso di eventi viene poi sottoposto a una catena di routine di gestione degli eventi di input, fra le quali compaiono per esempio quella di Intuition e quella del dispositivo Console. La disposizione di queste routine di gestione è a priorità.

Il dispositivo Console funziona sia come dispositivo di input sia come dispositivo di output. Permette a un task di aprire una console di I/O sullo schermo, cioè una finestra di Intuition adibita a ricevere dati da tastiera e a visualizzare caratteri. Lavora in stretto contatto con il dispositivo Input, e quindi indirettamente con i dispositivi Keyboard e Gameport, mentre le finestre che è in grado di controllare (ogni finestra aperta è un'unità del dispositivo) sono gestite da Intuition. Quando viene trattato come dispositivo di input, filtra gli eventi generati dai dispositivi Keyboard e Gameport che interessano le sue unità. Quando viene trattato come dispositivo di output, gestisce le operazioni di output per le finestre associate alle sue unità.

Il dispositivo Audio controlla la generazione di suoni attraverso i canali audio dell'Amiga.

Il dispositivo Narrator permette di far parlare l'Amiga, cioè di effettuare la riproduzione vocale di testi, preferibilmente in lingua inglese. Si avvale del dispositivo Audio e risiede su disco. All'occorrenza viene caricato in memoria.

Il dispositivo Serial controlla le comunicazioni che avvengono con dispositivi hardware esterni attraverso la porta seriale dell'Amiga. Gestisce i buffer di I/O, la velocità di trasmissione richiesta dai task, le condizioni di errore e di quelle EOF. Questo dispositivo è residente su disco e viene caricato all'occorrenza. Una volta in memoria si configura con alcuni parametri di default, ma i task possono modificare questa configurazione.

Il dispositivo Parallel controlla le comunicazioni che avvengono con dispositivi hardware esterni attraverso la porta parallela dell'Amiga. Gestisce i buffer di I/O, le condizioni di errore e quelle di EOF. Come il dispositivo Serial, risiede su disco e viene caricato in memoria all'occorrenza. Una volta in

memoria si configura con alcuni parametri di default, ma i task possono modificare questa configurazione.

Il dispositivo Printer permette di comunicare con la stampante a un livello più alto. I task che lo usano non sono tenuti a conoscere il tipo di stampante e la porta alla quale è collegata, dal momento che è il dispositivo Printer a dirigere i dati verso la giusta porta e a convertirli nei codici riconosciuti dalla stampante collegata. Per sapere quali sono questi parametri, il dispositivo Printer si affida al tool Preferences, soprattutto per quanto riguarda il driver di stampa prescelto dall'utente. È grazie ai driver di stampa che i task possono inviare testi e immagini alla stampante senza conoscerne le caratteristiche: il dispositivo Printer effettua tutte le necessarie operazioni di filtraggio. È possibile aggiungere anche altri driver di stampa al sistema e selezionarli tramite il tool Preferences, in modo che siano a disposizione del dispositivo Printer. Proprio come il dispositivo Printer, che risiede su disco, il driver di stampa viene caricato dal disco non appena l'utente o il task impartisce un comando di stampa.

Il dispositivo Clipboard permette ai task di effettuare il cut & paste di dati per scambiarli con altri task nel sistema.

A questi dispositivi standard è possibile aggiungere altri dispositivi di I/O non standard. Un esempio è il dispositivo RAMdrive presente dalla versione 1.3 del software sistema.

AddHead

Sintassi di chiamata della funzione

AddHead (list, node)
AØ A1

Scopo della funzione

Questa funzione aggiunge una nuova struttura Node all'inizio di una lista a concatenazione doppia.

Argomenti della funzione

list	Indirizzo della struttura List che definisce la lista alla quale la funzione deve aggiungere il nodo.

node Indirizzo della struttura Node da inserire nella lista.

Discussione

L'Exec dispone di varie funzioni predisposte per operare sui nodi e sulle liste doppie costituite da nodi concatenati. Queste funzioni appartengono a diverse categorie:

- funzioni per aggiungere e rimuovere nodi nelle liste (Insert e Remove).
- Funzioni per aggiungere e rimuovere nodi di testa e di coda nelle liste (AddHead, RemHead, AddTail e RemTail).
- Una funzione per inserire nodi nella posizione determinata dalla loro priorità (Enqueue).
- Una funzione per cercare un nodo con un certo nome in una lista (FindName).

Si noti che le funzioni Insert, Remove, AddHead e AddTail non usano il campo priorità della struttura Node.

L'elemento fondamentale di una lista è il nodo, che nell'Amiga è costituito da una struttura Node. Ecco la forma in linguaggio C della struttura Node:

```
struct Node {
    struct Node *ln_Succ;
    struct Node *ln_Pred;
    UBYTE ln_Type;
    BYTE ln_Pri;
    char *ln_Name;
};
```

I parametri della struttura Node hanno i seguenti significati:

- ln_Succ (successore) deve contenere l'indirizzo della successiva struttura Node.
- ln_Pred (predecessore) deve contenere l'indirizzo della precedente struttura Node.
- ln_Type indica il tipo di nodo. Può contenere un numero compreso tra 0 e 255, e viene usato dalle funzioni dell'Exec per verificare che tutti i nodi di una particolare lista siano dello stesso tipo.
- ln_Pri (priorità) indica la priorità del nodo. Può contenere un numero compreso tra +127 e –128 che determina la priorità della struttura Node,

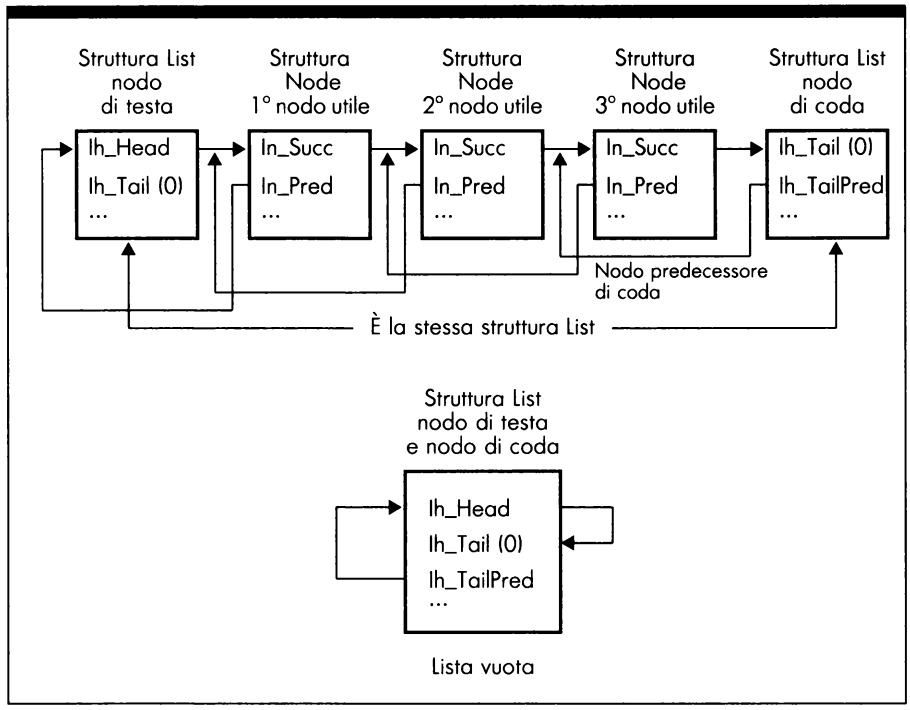


Figura 1.3: *Organizzazione di una lista doppia con tre nodi (in alto) e vuota (in basso)*

e quindi la sua posizione nelle liste organizzate secondo la priorità dei nodi. I nodi a priorità più alta si trovano in cima alla lista.

- ln_Name può contenere l'indirizzo di una stringa a terminazione nulla. Questa stringa viene impiegata per dare un nome alla struttura Node.

Le relazioni esistenti tra una lista, il nodo di testa, il nodo di coda e il nodo predecessore di coda sono illustrate nella Figura 1.3, che mostra in alto una lista composta da tre nodi e in basso una lista vuota, cioè priva di nodi.

Tutte le liste a doppia concatenazione impiegate nel sistema Amiga sono intestate da una struttura List, per la cui esposizione rimandiamo il lettore alla spiegazione della funzione AddTail. I primi tre parametri di questa struttura sono puntatori a strutture di tipo Node, il quarto indica il tipo di lista, e l'ultimo serve solo per allineare la dimensione della struttura alle word. In questa spiegazione ci interessano soltanto i primi tre parametri, che nell'ordine sono lh_Head, lh_Tail e lh_TailPred.

Una lista è una concatenazione di nodi. Viene definita a doppia concatenazione quando ogni nodo contiene un puntatore al nodo successore e un puntatore al nodo predecessore della lista. Questo doppio legame consente di leggere la lista in entrambi i sensi, ma soprattutto d'inserire e rimuovere i nodi più facilmente.

Durante la scansione in avanti dei nodi di una lista doppia, si rileva la fine

della lista quando il puntatore al nodo successore contiene uno zero. Allo stesso modo, durante la scansione all'indietro dei nodi di una lista doppia, si sa che si è giunti all'inizio quando il puntatore al nodo predecessore contiene uno zero.

Nelle liste doppie dell'Amiga, i nodi sono sempre rappresentati da strutture di tipo Node, che contengono appunto due puntatori a strutture Node come primi parametri: ln_Succ (contiene l'indirizzo del nodo successore) e ln_Pred (contiene l'indirizzo del nodo predecessore). L'unica eccezione a questa regola sono il nodo di testa e il nodo di coda, entrambi conglobati nella struttura List che definisce la lista. Osservando questa struttura, infatti, si deve pensare che la coppia di parametri lh_Head e lh_Tail rappresentino il nodo di testa, mentre la coppia lh_Tail e lh_TailPred quello di coda. In quest'ottica si nota che i puntatori al successore e al predecessore di questi due nodi fittizi si sovrappongono nella struttura List, e che questi due nodi, a differenza di tutti gli altri, non possono contenere dati, dal momento che in realtà non sono costituiti da strutture Node: devono essere quindi considerati semplici nodi di servizio che contribuiscono alla gestione della lista.

Da quanto esposto, è evidente che il modo migliore per comprendere l'organizzazione delle liste doppie nell'Amiga è considerare il nodo di testa e il nodo di coda come se fossero definiti da due strutture Node nelle quali i parametri dal terzo in poi non devono essere né considerati, né tantomeno alterati.

Ora analizziamo una generica lista dell'Amiga composta da tre nodi, cioè da tre strutture Node fra loro concatenate. In memoria risiede una struttura List, nella quale i primi due parametri vengono considerati dal sistema come i primi due parametri della struttura Node di testa della lista; pertanto, il parametro lh_Head contiene l'indirizzo del nodo successore (l'effettivo primo nodo della lista costituito da una struttura Node), mentre il parametro lh_Tail contiene un valore nullo, dal momento che il nodo di testa non può essere per definizione preceduto da altri nodi.

Il primo nodo della lista, individuato da lh_Head, identifica il primo gruppo di dati della lista. Si tratta di una struttura Node, nella quale il parametro ln_Succ contiene l'indirizzo della seconda struttura Node della lista, mentre il parametro ln_Pred contiene l'indirizzo della struttura List, nodo di testa della lista. A sua volta la seconda struttura Node è concatenata alla prima e alla terza.

Nella terza struttura Node della lista, l'ultima utile, il parametro ln_Pred contiene l'indirizzo della seconda struttura Node, mentre il parametro ln_Succ contiene l'indirizzo del nodo di coda, cioè del parametro lh_Tail contenuto nella struttura List che intesta la lista. Come dicevamo, questo parametro insieme al seguente parametro lh_TailPred costituisce il nodo fittizio di coda della lista. Pertanto, in questo particolare nodo il puntatore al nodo successore (lh_Tail) contiene un valore nullo, mentre il puntatore al nodo predecessore (lh_TailPred) contiene l'indirizzo della terza struttura Node, l'ultima utile della lista e denominata nodo predecessore di coda. Quindi, il parametro lh_Tail funge sia da puntatore al nodo predecessore del nodo di testa, sia da puntatore al nodo successore nel nodo di coda. In entrambi i casi deve sempre e comunque contenere il valore zero perché la gestione della lista, così come è stata descritta, funzioni.

Da questa descrizione emerge che nelle liste doppie utilizzate all'interno

dell'Amiga, l'inizio e la fine di una lista fanno capo alla stessa struttura List che la definisce, e che i nodi utili per memorizzare i dati sono solo quelli compresi fra il nodo di testa e il nodo di coda. Questa organizzazione impone nei cicli di scansione delle liste doppie due particolari condizioni di uscita: si è raggiunto l'ultimo nodo utile quando il parametro ln_Succ del nodo successore contiene il valore nullo, e si è raggiunto il primo nodo utile quando il parametro ln_Pred del nodo precedente contiene il valore nullo.

Nella parte bassa della Figura 1.3 è raffigurata una lista doppia vuota. In essa il parametro lh_Head della struttura List contiene l'indirizzo del nodo di coda, cioè l'indirizzo del parametro lh_Tail della stessa struttura, e il parametro lh_TailPred contiene l'indirizzo della struttura List in cui è contenuto.

Ci sono diverse macro-istruzioni in linguaggio Assembly che possono essere usate per semplificare la gestione delle liste nella programmazione in Assembly, ottenendo in tal modo una sua ottimizzazione. Queste macro sono definite nel file INCLUDE lists.i, e comprendono:

ADDHEAD	aggiunge una struttura Node alla testa di una lista;
ADDTAIL	aggiunge una struttura Node alla coda di una lista;
REMOVE	rimuove una struttura Node da una lista;
REMHEAD	rimuove una struttura Node dalla testa di una lista;
REMTAIL	rimuove una struttura Node dalla coda di una lista;
NEWLIST	inizializza una struttura List.

Per esempio, le istruzioni seguenti appartengono alla macro REMOVE:

```
MOVEA.L (A1),AØ          ; preleva l'indirizzo del nodo successore
MOVEA.L LN_PRED(A1),A1   ; preleva l'indirizzo del nodo predecessore
MOVE.L AØ,(A1)           ; aggiorna il puntatore ln_Succ del nodo predecessore
MOVE.L A1,LN_PRED(AØ)    ; aggiorna il puntatore ln_Pred del nodo successore
```

Si può anche determinare quando una lista è vuota. Per farlo si controlla se il parametro lh_TailPred della struttura List contiene l'indirizzo della struttura stessa. Se ciò accade significa che la struttura List intesta una lista vuota. In Assembly questo controllo si esegue con le seguenti istruzioni:

```
CMP.L LH_TAILPRED(AØ), AØ
BEQ listavuota
```

Per esaminare una lista è necessario scorrerla fino a che viene raggiunto il nodo di coda. Ancora meglio, si può mantenere un registro puntatore per muoversi lungo la lista, impiegando le seguenti istruzioni:

```
        MOVE.L (A1), D1      ; primo nodo
loopScansione:
        MOVEA.L D1, A1       ; imposta il registro indice
        MOVE.L (A1), D1      ; indirizzo contenuto nel parametro successore del
                               nodo seguente
        BEQ.S fineScansione  ; fine della lista
        <corpo>              ; usa A1 come puntatore al nodo
        BRA.S loopScansione
fineScansione:
```

Queste istruzioni in linguaggio Assembly conservano nel registro A1 l'indirizzo del nodo in uso della lista. Questo consente al programma di guardare a ogni struttura Node prima di usarla. In C il loop di scansione di una lista potrebbe essere il seguente:

```
struct List *lista;
struct Node *nodo;
...
for (nodo = lista->lh_Head; nodo->ln_Succ; nodo = nodo->ln_Succ)
{ /* il puntatore nodo contiene l'indirizzo dell'attuale nodo */
}
```

Per concludere, *tutte* le strutture standard di sistema possiedono una struttura Node come primo elemento. L'Exec prevede i seguenti tipi di struttura Node, i cui nomi sono piuttosto esplicitativi:

NT_DEVICE	NT_MEMORY	NT_SEMAPHORE
NT_FONT	NT_MESSAGE	NT_SOFTINT
NT_FREEMSG	NT_PROCESS	NT_TASK
NT_INTERRUPT	NT_REPLYMSG	NT_UNKNOWN
NT_LIBRARY	NT_RESOURCE	

I file INCLUDE che contengono le definizioni di queste costanti sono exec/nodes.h (per l'uso con il linguaggio C) e exec/nodes.i (per l'uso con il linguaggio Assembly).

AddIntServer

Sintassi di chiamata della funzione

AddIntServer (intNumber, interrupt)
DØ: Ø-4 A1

Scopo della funzione

Questa funzione aggiunge una nuova routine di interrupt a una catena di routine di servizio (detta anche server chain). Questa routine di servizio viene collocata nella catena in una posizione determinata dalla sua priorità; gli interrupt per i quali le relative routine di servizio possiedono una più alta priorità vengono soddisfatti per primi. Ogni routine di servizio della catena può essere interrotta nel corso della sua esecuzione, se si verifica un interrupt a priorità più alta.

Argomenti della funzione

intNumber — Bit di interrupt del chip Paula (chip dedicato alle periferiche e al suono).

interrupt — Puntatore alla struttura Interrupt che contiene il punto d'ingresso per la routine di interrupt corrispondente al bit di interrupt indicato.

Discussione

Ci sono quattro funzioni che gestiscono le routine di servizio: AddIntServer, Cause, RemIntServer e SetIntVector. Si vedano anche le spiegazioni relative a queste funzioni.

Le routine di servizio degli interrupt vengono chiamate applicando le seguenti convenzioni sui registri:

D0 — Alterabile.

D1 — Alterabile.

A0 — Alterabile.

A1 — In ingresso, puntatore al segmento dati della routine di servizio; alterabile una volta all'interno della routine.

A5 — In ingresso, vettore di salto; alterabile una volta all'interno della routine.

A6 — In ingresso, puntatore all'indirizzo base di sistema, cioè alla struttura ExecBase; alterabile una volta all'interno della routine.

Tutti gli altri registri debbono essere preservati. Il termine *alterabile* significa che il registro può essere usato e modificato per manipolare dati una volta che ci si ritrova all'interno della routine di servizio dell'interrupt.

Ci sono due tipi di routine usate per servire gli interrupt nel sistema Amiga.

1. Gli handler, routine di gestione degli interrupt che controllano direttamente le specifiche richieste di elaborazione di un interrupt associato a un determinato bit di interrupt del chip Paula. Quando l'interrupt si verifica, l'handler associato è l'unica routine a ricevere il controllo.

2. I server, routine di servizio degli interrupt che sono sempre organizzate in catene associate a particolari interrupt. Quando uno di questi interrupt si verifica, le routine di servizio elencate nella corrispondente catena vengono eseguite una dopo l'altra, nell'ordine indicato dalle loro priorità d'esecuzione. Questa priorità dev'essere memorizzata nel parametro ln_Pri della struttura Node che costituisce il primo elemento della struttura Interrupt. Non è detto che tutte le routine presenti in una catena di server riescano a ottenere il controllo quando l'interrupt si verifica.

Nell'Amiga ci sono due tipi di interrupt: software e hardware. Solitamente tutti gli interrupt hardware sono elaborati dalle routine dell'Exec prima di procedere alla chiamata delle relative routine di interrupt. Ecco come gli interrupt dell'Amiga vengono identificati: i sette livelli di interrupt della CPU sono decodificati nei 15 vettori di interrupt dell'Amiga.

Per il modo in cui le cose sono organizzate, al verificarsi di uno stesso interrupt molto spesso sono diverse le routine di servizio che devono ricevere il controllo. Per esempio, diverse routine di servizio devono ricevere il controllo quando si verifica l'interrupt di vertical-blanking (interrupt dell'Amiga numero 5); ciò accade ogni 1/60 di secondo per le macchine in standard americano (NTSC) e 1/50 di secondo per le macchine in standard europeo (PAL). Per questa ragione sono state predisposte le catene di routine di servizio degli interrupt, nelle quali ogni routine, rappresentata da una struttura Interrupt, possiede un suo livello di priorità, cioè una sua priorità d'esecuzione.

Come si può immaginare, le routine a più alta priorità vengono eseguite per prime. Ciò assicura il rapido svolgimento di certi compiti che devono essere eseguiti prontamente quando si verifica l'interrupt.

Oltre agli interrupt hardware, il sistema permette di causare interrupt software e quindi l'esecuzione delle relative routine di interrupt di basso livello. Questo tipo di routine di interrupt non è in relazione con alcun particolare dispositivo hardware. L'istruzione TRAP del 68000 è uno dei modi usati per implementare gli interrupt software.

Gli interrupt riguardano anche lo scambio di controllo fra i task. La maggior parte delle volte, uno scambio di controllo si verifica come risultato di un interrupt, tanto software quanto hardware. Ne segue che il sistema degli interrupt costituisce il cuore del meccanismo che sovraintende allo scambio di

controllo fra i task dell'Amiga.

Si veda la spiegazione della funzione Cause per un elenco delle possibili sorgenti di interrupt. Ciascun interrupt, quando si verifica, imposta un bit nel registro di richiesta di interrupt del chip Paula. Ci sono 15 bit di interrupt ma esistono solo sei livelli di interrupt (oltre al settimo livello previsto dal 68000) effettivamente utilizzati.

All'interno di ciascun livello di routine di servizio degli interrupt c'è una priorità software determinata dall'ordine nel quale vengono esaminati i vari bit di interrupt. Si veda la spiegazione della funzione Cause per conoscere il meccanismo di assegnazione della priorità nei vari livelli. Ci sono sei livelli di assegnazione delle priorità di interrupt: il livello 6 è quello a priorità più alta mentre il livello 1 è quello a priorità più bassa.

Livelli di priorità degli interrupt

Il coprocessore di controllo video Copper possiede il più alto livello di priorità di interrupt, il livello 6. L'azione del Copper è sincronizzata con il movimento del pennello elettronico lungo lo schermo dell'Amiga, dall'alto verso il basso procedendo linea dopo linea. Se il Copper genera un interrupt, in genere significa che non appena il pennello elettronico oltrepassa una certa posizione sulla linea di scansione dello schermo dev'essere eseguito qualche compito.

L'interrupt della porta d'espansione possiede anch'esso il livello di priorità 6, ma essendo meno importante può aspettare fino a quando l'interrupt generato dal Copper non è stato completamente servito.

Il disco e l'UART (Universal Asynchronous Receiver Transmitter, trasmettitore-ricevitore asincrono universale) sono posti al livello 5. Il tempo critico di risposta è molto breve per il disco (approssimativamente 16 microsecondi). La risposta dell'UART è meno esigente (approssimativamente 250 microsecondi).

Tutti e quattro i canali audio sono posti al livello 4. I canali audio 0 e 2 possono anche essere usati per modulare l'uscita dei canali audio 1 e 3. Normalmente i canali audio 1 e 3 vengono usati per riprodurre forme d'onda. In questo caso l'overhead dell'interrupt per i canali 1 e 3 viene minimizzato ponendoli alla testa della relativa catena di routine di servizio. Il Blitter, il Copper (a volte) e gli interrupt di vertical-blanking sono posti al livello 3. Le routine grafiche tengono il Blitter occupato il più possibile, ed è questo il motivo per cui i grafici e le animazioni sono così veloci. Ecco perché in questo livello il canale DMA del Blitter viene soddisfatto per primo. D'altra parte, se il coprocessore Copper dovesse avere una priorità più alta, potrebbe scrivere nel bit 15 del registro di richiesta di interrupt, invece che nel bit 5. Ciò causerebbe un livello 6 di interrupt della CPU, anziché un livello 3. Il vertical-blanking si verifica a ritmi lenti se confrontati con quelli di altre sorgenti di interrupt. Per questo motivo, la routine di servizio dell'interrupt di vertical-blanking è in fondo alla catena di server.

Il livello 2 riguarda le porte di I/O, la tastiera e i timer. Questi sono dispositivi a media velocità, e quindi l'esecuzione delle loro routine di interrupt

è richiesta a un ritmo medio.

Gli interrupt relativi alla condizione di buffer di trasmissione pieno della porta seriale e alla condizione di blocco del disco trasferito sono posti al livello 1. Il primo è servito da una routine di interrupt che capeggia la catena di server associata al livello 1, seguita dalla routine di servizio dell'interrupt di trasferimento del blocco del disco. L'ultima routine della catena è quella che serve gli interrupt software; questi interrupt sono quelli meno critici per quanto riguarda l'intervallo fra il loro verificarsi e la loro elaborazione.

Catene di server definite dal sistema

Le routine dell'Exec costruiscono automaticamente una catena di server per sei tipi di interrupt:

- l'interrupt del livello 3 relativo al vertical-blanking.
- L'interrupt del livello 2 causato dai dispositivi collegati alla porta d'espansione dell'Amiga.
- L'interrupt del livello 6 causato dai dispositivi connessi alla porta d'espansione dell'Amiga.
- L'interrupt del livello 3 causato dal coprocessore Copper.
- Gli interrupt dei livelli 2 e 6 relativo alle porte di I/O.
- L'interrupt del livello 3 relativo al Blitter.

Attualmente non esiste un punto di entrata nel codice del sistema per creare una catena di server come gestore di uno o più degli altri bit di interrupt. Tuttavia è possibile creare propri handler se si usa lo stesso modo di procedere impiegato per queste sei catene di server.

AddLibrary

Sintassi di chiamata della funzione

AddLibrary (library)
A1

Scopo della funzione

Questa funzione aggiunge un nuova libreria al sistema e la rende disponibile per ogni task che desideri chiamarne le funzioni. La struttura Library di gestione della libreria viene aggiunta alla lista di sistema LibList. Prima che questa funzione venga chiamata, la libreria dev'essere già in memoria e pronta per essere aperta. Questa funzione calcola inoltre il checksum sulla libreria.

Argomenti della funzione

library Indirizzo della struttura Library opportunamente definita.

Discussione

Ci sono otto funzioni dell'Exec che si occupano delle librerie nel sistema Amiga: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Le librerie sono identificate tramite i loro nomi e numeri di versione, e possono contenere le routine più frequentemente usate dal programmatore. Cambiando il numero di versione di una libreria si può disporre di librerie più strettamente correlate all'applicazione.

Si può tenere su disco una collezione di diverse librerie. Queste librerie sono classificate come librerie di sistema e librerie utente. Se una libreria è in ROM si dice che è *residente*; se è su disco si dice che è *non-residente*. I task usano le librerie per evitare di contenere al loro interno i codici per lo svolgimento di compiti già previsti dalle librerie. Nel sistema possono risiedere molti task contemporaneamente. Ciò significa che diversi task possono trovarsi nella necessità di usare la stessa libreria.

La Figura 1.4 illustra la struttura generale di una libreria. Se si crea una libreria con MakeLibrary, è necessario conformarla a questo formato. Esprimendosi formalmente, il rettangolo grande rappresenta la libreria. I rettangoli piccoli, verso l'alto della figura, rappresentano le funzioni della libreria, che non fanno effettivamente parte della sua definizione. Soltanto i vettori di salto a queste funzioni sono davvero "nella libreria". La figura mostra i tre componenti della libreria: la struttura Library, seguita in memoria dal suo segmento dati, e preceduta dalla tavola dei vettori di salto.

Il primo elemento della struttura Library è la sotto-struttura Node lib_Node, che ne permette l'inserimento nella lista di sistema LibList.

Per convenzione, un vettore di salto a una funzione occupa sei byte, due occupati dalla codifica dell'istruzione JMP e quattro dall'indirizzo della funzione. I vettori di salto a sei byte sono usati per individuare segmenti di

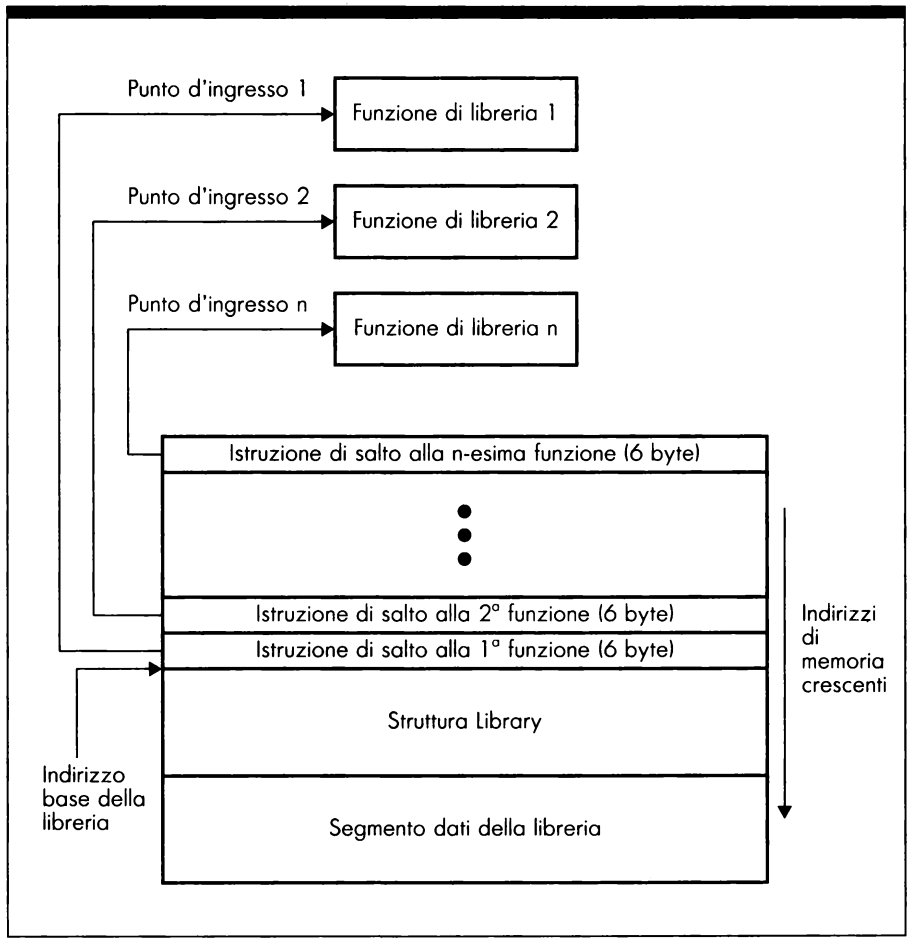


Figura 1.4: *Relazione fra una libreria e le sue funzioni*

codice in qualunque punto dello spazio indirizzabile della CPU. L'indirizzo della struttura Library è l'indirizzo base della libreria, il fulcro per individuare gli altri elementi. I vettori di salto sono collocati a offset negativi rispetto all'indirizzo base, mentre le aree dati sono collocate a offset positivi rispetto all'indirizzo base.

Se i codici delle funzioni della libreria sono entro 32K dalla struttura Library, si possono usare vettori di salto da quattro byte nella definizione della libreria. Se sono ancora più vicini, si possono usare vettori di salto da due byte.

Il segmento dati è di misura variabile. Il suo contenuto e le sue dimensioni dipendono dagli usi e dalle necessità previste per la libreria. Le funzioni della libreria accedono alle locazioni del segmento dati attraverso opportuni offset positivi dalla struttura Library. Differenti linguaggi di programmazione hanno differenti esigenze d'interfacciamento con le librerie. Quando si progettano librerie proprie, si possiede un completo arbitrio sull'organizzazione dei

segmenti di dati.

Lo scopo della funzione AddLibrary è di aggiungere una nuova libreria alla lista di librerie del sistema. Essa si limita ad aggiungere la libreria alla lista, e non provvede ad aprirla. La libreria dev'essere appropriatamente definita prima di chiamare la funzione AddLibrary. Si può definire una libreria utilizzando la funzione MakeLibrary.

Per accedere a una libreria già definita è necessario eseguire i due seguenti passi.

1. Per prima cosa si deve aprire la libreria con la funzione OpenLibrary. Se la libreria non risulta disponibile, cioè già in memoria, la funzione provvede a caricarla da disco e a renderla disponibile in modo del tutto trasparente.

2. In secondo luogo, si deve accedere alle funzioni e ai dati della libreria specificando un offset, sia negativo (per i vettori di salto) che positivo (per i dati), a partire dall'indirizzo base della struttura Library restituito dalla funzione OpenLibrary.

Osservando questo schema di accesso si può vedere che i propri task non sono tenuti a conoscere nessun indirizzo della libreria alla quale devono accedere: è sufficiente che conoscano gli offset alle funzioni e li utilizzino insieme all'indirizzo base che ottengono aprendo la libreria con la funzione OpenLibrary. Con questa organizzazione, le librerie possono risiedere in qualsiasi area dello spazio indirizzabile, e i task possono funzionare anche su modelli di Amiga differenti.

La struttura Library

Una libreria ha lo scopo di offrire un insieme organizzato di funzioni correlate che possono essere chiamate secondo adeguate modalità. La struttura Library associata a una libreria contiene tutte le informazioni necessarie al sistema per gestire quella libreria. Nella notazione in linguaggio C la struttura è definita come segue:

```
struct Library {
    struct Node lib_Node;
    UBYTE lib_Flags;
    UBYTE lib_pad;
    UWORD lib_NegSize;
    UWORD lib_PosSize;
    UWORD lib_Version;
    UWORD lib_Revision;
    APTR lib_IDString;
    ULONG lib_Sum;
    UWORD lib_OpenCnt;
};
```

Questi sono i parametri della struttura Library:

- il parametro lib_Node è una sotto-struttura Node, tramite la quale la struttura Library può essere mantenuta all'interno della lista di sistema LibList. Il parametro ln_Name nella struttura Node deve contenere l'indirizzo della stringa di testo che costituisce il nome della libreria, permettendo perciò di riferirsi alla libreria tramite il suo nome. Il parametro ln_Pri della struttura Node determina dove si trova la libreria nella lista delle librerie. Prima di aggiungere una libreria nel sistema si dovrebbero sempre predefinire questi due parametri. Si dovrebbe anche impostare il parametro ln_Type nella struttura Node a NT_LIBRARY per indicare che la struttura Node appartiene a una libreria.

- Il parametro lib_Flags contiene un insieme di flag di gestione della libreria. Questi flag sono spiegati dettagliatamente in seguito.

- Il parametro lib_pad è un parametro di riempimento da 1 byte, necessario per mantenere l'appropriato allineamento della struttura alle word.

- Il parametro lib_NegSize contiene il numero di byte della libreria che precedono in memoria la struttura Library stessa. Questa area è la tavola dei vettori di salto alle funzioni della libreria.

- Il parametro lib_PosSize contiene il numero di byte della libreria che seguono in memoria la struttura Library stessa. Questa area contiene i dati della libreria.

- Il parametro lib_Version contiene il numero di versione della libreria.

- Il parametro lib_Revision contiene il numero di revisione della libreria.

- Il parametro lib_IDString contiene l'indirizzo di una stringa di testo a terminazione nulla adibita a identificare la libreria.

- Il parametro lib_Sum contiene il valore del checksum per la libreria. Ogni volta che il contenuto della libreria viene cambiato questo parametro dovrebbe essere aggiornato. Vedere le spiegazioni della funzione SetFunction.

- Il parametro lib_OpenCnt tiene conto del numero di task che hanno aperto la libreria e non l'hanno ancora richiusa. Ogni volta che un task chiama la funzione OpenLibrary per aprire la libreria associata a questa struttura Library, questo parametro viene incrementato. Ogni volta che un task chiama la funzione CloseLibrary chiudere questa libreria, il parametro viene decrementato.

AddMemList

Sintassi di chiamata della funzione

```
error = AddMemList (size, attributes, priority, basePointer, name)
D0                  D0    D1        D2        A0          A1
```

Scopo della funzione

Questa funzione aggiunge un nuovo blocco di memoria a un pool di memoria libera dotato di nome. Questo blocco consiste di uno o più sotto-blocchi. La prima sezione di byte di questo blocco conterrà la struttura MemHeader per la definizione del blocco di memoria e di ciascuno dei suoi sotto-blocchi. Non è necessario che i sotto-blocchi siano contigui in memoria; le strutture MemHeader e MemChunk prevedono i puntatori e le correlazioni per ciascun sotto-blocco. Il resto del blocco (altri sotto-blocchi) diventa così disponibile al sistema e a ogni altro task che vorrà impiegarlo.

Argomenti della funzione

size — Numero totale dei byte dell'intero blocco di memoria che devono essere aggiunti.

attributes — Insieme degli attributi che il blocco di memoria dovrà avere. Vedere la spiegazione della funzione AllocMem per la discussione di questi attributi.

priority — È la priorità per questo blocco di memoria. La chip RAM ha una priorità pari a -10 e la fast RAM ha una priorità pari a 0. La priorità di un blocco di memoria è contenuta nel parametro ln_Pri della sotto-struttura Node all'interno della struttura MemHeader.

basePointer — L'indirizzo della locazione base del nuovo blocco di memoria libera.

name — È l'indirizzo del nome del pool di memoria libera. Questo nome sarà usato nella struttura MemHeader; si usi un valore nullo se non dev'essere assegnato alcun nome a questo pool di memoria. Il nome non

viene copiato; perciò deve rimanere valido finché la struttura MemHeader è presente nel sistema.

Discussione

Prima della versione 1.2 non esisteva un modo efficiente per aggiungere un insieme di blocchi di memoria a un pool di memoria libera identificato da un nome. AddMemList fornisce un modo conveniente per ottenere questo risultato.

La struttura MemHeader serve al sistema per definire esplicitamente un pool di memoria libera identificato da un nome; questo argomento viene approfondito nella spiegazione della funzione Allocate. È opportuno prendere nota delle seguenti osservazioni in relazione alla funzione AddMemList:

- ciascuna struttura MemHeader contiene una sotto-struttura Node, la quale contiene i parametri del nome (ln_Name) e della priorità (ln_Pri) per la struttura MemHeader. La funzione AddMemList usa gli argomenti della chiamata per definire questi parametri nella struttura MemHeader.

- Ciascuna struttura MemHeader contiene un parametro per gli attributi (mh_attributes), che la funzione aggiorna con il valore indicato dal task nell'argomento attributes della chiamata.

- Ciascuna struttura MemHeader indica un limite verso il basso e verso l'alto del blocco di memoria che definisce. In aggiunta, ciascuna struttura MemHeader indica il numero totale di byte liberi (la misura totale del pool di memoria libera) presenti nell'intero blocco di memoria controllato dalla struttura MemHeader.

- Ciascuna struttura MemHeader contiene un puntatore alla prima di una o più strutture MemChunk, le quali costituiscono una lista concatenata di blocchi di memoria, il completo pool di memoria libera. Ciascuna struttura MemChunk contiene un puntatore al blocco seguente di memoria della lista.

Si dovrebbero studiare le strutture MemHeader e MemChunk nel file INCLUDE memory.h per comprendere più a fondo queste relazioni.

AddPort

Sintassi di chiamata della funzione

AddPort (msgPort)
A1

Scopo della funzione

Questa funzione aggiunge un nuova message port alla lista di message port del sistema e la rende disponibile per ogni task del sistema. Quando una message port viene aggiunta al sistema si dice che è *pubblica*. Il nome e i campi priorità della sotto-struttura Node della struttura MsgPort devono essere inizializzati prima di chiamare questa funzione. Se il programmatore non ha bisogno del nome e dei campi di priorità dovrà impostare questi parametri a zero. Come avviene per il campo nome nelle altre liste di sistema, il nome è comunque indispensabile se si desidera che la message port sia pubblica e quindi rintracciabile da qualsiasi task che ne conosca il nome (tramite la funzione FindPort).

Argomenti della funzione

msgPort Indirizzo della struttura MsgPort.

Discussione

Ci sono quattro routine dell'Exec che riguardano esplicitamente le message port nel sistema Amiga: AddPort, FindPort, RemPort e WaitPort. AddPort, FindPort e RemPort si occupano esclusivamente delle message port pubbliche che vengono aggiunte alla lista di sistema con la funzione AddPort. WaitPort, invece, riguarda sia le message port pubbliche che quelle private. Queste routine gestiscono le message port tramite le quali i task possono scambiarsi vicendevolmente messaggi. I messaggi sono lo strumento con il quale i task cooperano nel sistema multitasking dell'Amiga.

Strettamente correlate alle routine di gestione delle message port sono le routine di gestione dei messaggi: PutMsg, GetMsg e ReplyMsg. Per comprendere le operazioni svolte dalle funzioni di gestione delle message port si dovrebbero studiare anche le operazioni svolte dalle funzioni di gestione dei messaggi.

Le message port e le reply port

In generale, le message port sono classificate in due categorie: message port e reply port. Ogni volta che un task crea una message port (usando la funzione AddPort), significa che intende ricevere messaggi dal sistema (in generale da qualunque task possieda l'indirizzo di quella message port). Quando invece desidera inviare messaggi, per esempio a un dispositivo o ad altri task, deve crearsi una reply port, che non ha generalmente bisogno di essere pubblica. Il task deve memorizzare l'indirizzo della sua reply port nel parametro mn_ReplyPort delle strutture Message che definiscono i messaggi: rappresenta l'indirizzo del mittente del messaggio, cioè la message port alla quale il messaggio dev'essere restituito dal destinatario.

Per definizione, è attivo un solo task alla volta; si dice che tutti gli altri task sono "in riposo". I messaggi inoltrati dal task mittente vengono inseriti nelle code alle message port destinatarie. Al successivo scambio di controllo un task destinatario può acquisire il controllo del sistema ed esaminare i messaggi giunti alla sua message port. All'arrivo di un messaggio, tutti i segnali generati dalle message port di un task devono essere esaminati per rilevare in quale message port è arrivato.

Il task mittente di un messaggio ha bisogno di sapere quando un messaggio è stato elaborato dal ricevente. Una volta che il task destinatario di un messaggio lo ha ricevuto e lo ha elaborato, deve quindi restituirlo al mittente tramite la funzione ReplyMsg; questa funzione accoda il messaggio alla reply port indicata nel messaggio stesso. Quando il task mittente riceve il controllo della CPU, può sondare la propria reply port per verificare quali dei messaggi inviati sono stati restituiti. La sequenza delle azioni successive dipende da questa verifica.

Le caratteristiche delle message port

Mentre possono esserci molti task nel sistema, uno solo di essi può essere attivo in ogni momento. Quale sia il task attivo è determinato in parte dalla sua priorità relativa. Tutti i task, sia quello attivo che quelli in riposo, possono disporre di un qualsiasi numero di message port, ognuna delle quali può essere stata resa pubblica tramite la funzione AddPort. I messaggi contenuti in una message port possono essere di qualsiasi lunghezza.

Quando un task crea una message port pubblica, questa message port viene aggiunta al sistema e diventa un elemento della lista di sistema delle message port. Il sistema mantiene automaticamente la lista delle message port pubbliche.

Le message port possono essere pubbliche o private. Una message port è *pubblica* quando è disponibile per l'intero sistema, cioè quando tutti i task possono inviarle messaggi. Si deve usare la funzione AddPort per rendere pubblica una message port. A una message port pubblica dev'essere assegnato un nome. Una message port *privata* viene usata soltanto da sezioni di codici che lavorano in stretta cooperazione. Il sistema stesso mantiene e impiega diverse message port private per le sue necessità.

AddResource

Sintassi di chiamata della funzione

AddResource (resource)
A1

Scopo della funzione

Questa funzione aggiunge un nuova risorsa alla lista di risorse del sistema, e la rende disponibile per tutti i task del sistema. Si noti che la risorsa viene semplicemente collocata in una lista di sistema. Prima di chiamare la funzione la risorsa dev'essere anche preparata all'impiego.

Argomenti della funzione

resource	Indirizzo di una struttura Resource opportunamente definita.

Discussione

Ci sono tre funzioni dell'Exec direttamente correlate alla gestione delle risorse dell'Amiga: AddResource, OpenResource e RemResource. I loro nomi suggeriscono il particolare compito che svolgono. Si noti che non esiste una funzione CloseResource.

La lista delle risorse di sistema

Il sistema mantiene in RAM una lista di tutte le risorse di sistema. Ogni volta che si effettua una chiamata alla funzione AddResource, questa lista viene aggiornata automaticamente. La funzione OpenResource alloca anche la memoria necessaria per la gestione della risorsa.

La sequenza di gestione delle risorse

La sequenza generale di gestione delle risorse procede secondo queste linee.

1. La risorsa viene aggiunta alla lista di risorse del sistema tramite la funzione AddResource. Questa funzione opera come le funzioni AddLibrary, AddDevice e altre funzioni dell'Exec che aggiungono risorse software al sistema. Una volta che la risorsa è stata aggiunta, può essere utilizzata da ogni task del sistema, tanto da quelli creati dal programmatore, quanto da quelli di sistema. Fino a questo punto è stata usata soltanto la memoria necessaria per aggiungere la risorsa alla lista di sistema.

2. Ciascun task che vuole usare la risorsa deve aprirla con una chiamata alla funzione OpenResource. Lo scopo della funzione OpenResource è simile a quello delle funzioni OpenLibrary, OpenDevice e le altre funzioni "Open" dell'Exec: offrire a chi la chiama l'accesso alla risorsa. Siccome l'Amiga è un sistema multitasking, diversi task possono accedere alla stessa risorsa; ognuno di essi può effettuare una chiamata alla funzione OpenResource quando diventa attivo.

Può quindi accadere che diversi task chiamino la funzione OpenResource per aprire la stessa risorsa, anche se può essere attivo solo un task alla volta. Ogni chiamata alla funzione OpenResource assegna questa risorsa a un nuovo task. La risorsa diventa così "condivisa". Questo assicura che i dati andranno e torneranno dalla risorsa a un solo task (quello in esecuzione), anche se diversi altri task possono in seguito usare la stessa risorsa per inviare e ricevere dati.

La libreria di una risorsa

Una risorsa è rappresentata nel sistema da una libreria di funzioni. Queste funzioni sono conservate nello stesso formato utilizzato per tutte le altre librerie standard dell'Amiga. Si può usare la funzione MakeLibrary per creare la libreria di una risorsa. Come accade per le altre librerie, la libreria di una risorsa ha diversi vettori di salto che precedono la struttura Library.

Creazione e uso delle risorse

Per creare una risorsa da inserire nel sistema, occorre definirla come libreria. Questa trasformazione in libreria dev'essere effettuata dopo che i codici della risorsa sono stati opportunamente creati e verificati. Si porranno le specifiche funzioni della risorsa nella libreria collocandole a vari offset dalla struttura Library. Una volta che si conoscono gli offset si possono specificare i vettori di salto per raggiungere le funzioni. Fra questi sono indispensabili quelli relativi alle funzioni obbligatorie della libreria e quelli relativi alle sue aree dati.

Una volta che tutto questo è stato fatto, si è pronti a costruire la libreria della risorsa chiamando la funzione MakeLibrary. Se tutto va a buon fine, la definizione della libreria sarà accettata e si potrà usare la funzione AddResource per aggiungere la risorsa alla lista di sistema, in modo che qualsiasi task possa aprirla e accedere alle sue funzioni.

Quando nessun task ha più bisogno della risorsa, si può rimuoverla con RemResource. Questa funzione libera tutta la memoria assegnata alla risorsa. Se il sistema è dotato di molta memoria RAM, non è necessario rimuovere continuamente le risorse aperte.

Il sistema Amiga possiede cinque risorse. Si veda anche la descrizione della funzione OpenResource.

AddSemaphore

Sintassi di chiamata della funzione

AddSemaphore (signalSemaphore)
A1

Scopo della funzione

Questa funzione aggiunge una struttura SignalSemaphore alla lista di sistema dei semafori di segnalazione. I parametri nome (ln_Name) e priorità (ln_Pri) della sotto-struttura Node della struttura SignalSemaphore devono essere impostati prima di chiamare AddSemaphore. Se non si desidera che altri task abbiano rapporti con questo semaforo (non abbiano cioè accesso alla sua struttura SignalSemaphore), deve invece essere usata la funzione InitSemaphore, che non richiede di definire il parametro ln_Name della sotto-struttura Node della struttura SignalSemaphore.

Argomenti della funzione

signalSemaphore Indirizzo di una struttura SignalSemaphore adeguatamente definita. Sia il parametro del nome che quello della priorità della sotto-struttura Node della struttura SignalSemaphore devono essere impostati prima di cedere il controllo alla funzione AddSemaphore.

Discussione

Ci sono sette funzioni che riguardano i semafori di segnalazione: AddSemaphore, AttemptSemaphore, FindSemaphore, InitSemaphore, ObtainSemaphore, ReleaseSemaphore e RemSemaphore. Ciascuna di esse interagisce con una struttura SignalSemaphore. Oltre a queste funzioni, esistono due funzioni che riguardano la lista dei semafori di segnalazione (ObtainSemaphoreList e ReleaseSemaphoreList) e due funzioni che hanno a che fare con i semafori basati su messaggi (Procure e Vacate). Si legga la spiegazione di ciascuna di queste funzioni per comprendere come e perché vengono usati i semafori nel software sistema dell'Amiga.

Le prime nove funzioni lavorano direttamente con la struttura SignalSemaphore e con i suoi parametri.

La struttura SignalSemaphore

La struttura SignalSemaphore ha nel file INCLUDE semaphores.h la seguente definizione:

```
struct SignalSemaphore {
    struct Node ss_Link;
    SHORT ss_NestCount;
    struct MinList ss_WaitQueue;
    struct SemaphoreRequest ss_MultipleLink;
    struct task *ss_Owner;
    SHORT ss_QueueCount;
};
```

Questi sono i parametri nella struttura SignalSemaphore:

- il parametro ss_Link è una sotto-struttura Node, nella quale il parametro nome (ln_Name), il parametro priorità (ln_Pri) e altri parametri consentono alla struttura SignalSemaphore di essere inserita in una lista di semafori di segnalazione e di essere identificata tramite un nome.

- Il parametro ss_NestCount è un contatore che tiene conto del numero di volte che questo particolare semaforo è stato utilizzato (con una chiamata a ObtainSemaphore) e rilasciato (con una chiamata a ReleaseSemaphore) dal task che lo detiene. Solo quando questo parametro vale zero, un altro task può accedere a questa struttura SignalSemaphore.

- Il parametro ss_WaitQueue è una sotto-struttura MinList che definisce la lista dei task che stanno attendendo di accedere a questa struttura SignalSemaphore. La struttura MinList è una forma ridotta della

struttura List, ed è definita nel file INCLUDE lists.h. Quando il parametro ss_NestCount giunge a zero, il sistema consulta questa lista per determinare quali task sono in coda per usare questa struttura SignalSemaphore.

- Il parametro ss_MultipleLink è una sotto-struttura SemaphoreRequest contenuta nella struttura SignalSemaphore. La struttura SemaphoreRequest viene allocata dalla funzione ObtainSemaphore quando un task tenta di accedere a una struttura SignalSemaphore non utilizzabile. In tal caso si forma un legame multiplo fra tutti i task che stanno tentando di usare il semaforo di segnalazione rappresentato da questa struttura SignalSemaphore.

- Il parametro ss_Owner è un puntatore alla struttura Task del task che ha accesso alla struttura SignalSemaphore. Quando un nuovo task ottiene l'accesso alla stessa struttura SignalSemaphore, il sistema provvede ad aggiornare il parametro.

- Il parametro ss_QueueCount è il contatore del numero dei task che sono in coda per usare una struttura SignalSemaphore. Esso informa il sistema di quanto è lunga la lista ss_WaitQueue. Ogni volta che un altro task tenta di accedere alla struttura SignalSemaphore, questo parametro viene incrementato.

AddTail

Sintassi di chiamata della funzione

AddTail (list, node)
A0 A1

Scopo della funzione

Questa funzione aggiunge una nuova struttura Node alla coda di una lista a doppia concatenazione.

Argomenti della funzione

list Indirizzo della struttura List la cui lista è destinata a contenere questo nodo.

node Indirizzo della struttura Node da inserire in coda alla lista.

Discussione

La libreria Exec dell'Amiga ha diverse funzioni destinate a operare con i nodi e le liste a doppia concatenazione. Per un'esposizione completa si veda la spiegazione della funzione AddHead.

Grazie all'organizzazione a doppia concatenazione, si possono inserire e cancellare nodi in ogni punto della lista senza doverla leggere dall'inizio. Ciascuna lista è definita da una struttura List che in linguaggio C è definita come segue:

```
struct List {
        struct Node *lh_Head;
        struct Node *lh_Tail:
        struct Node *lh_TailPred;
        UBYTE lh_Type;
        UBYTE l_pad;
};
```

I campi lh_Head e lh_Tail formano insieme il primo nodo della lista, chiamato nodo di testa. I campi lh_Tail e lh_TailPred formano l'ultimo nodo della lista, detto nodo di coda. Questi due nodi sono fittizi, nel senso che non sono in realtà costituiti da strutture Node come tutti gli altri, e quindi non possono contenere dati; servono unicamente per la gestione della lista e si sovrappongono all'interno della struttura List. Il parametro lh_Type serve per verificare i singoli nodi della lista (non è detto che venga sempre usato). Ci sono 14 tipi di nodo, e sono stati definiti nel corso della trattazione della funzione AddHead. Il campo l_pad è un byte di allineamento della struttura alle word, e pertanto non viene impiegato.

La struttura List è collegata al resto della lista secondo queste modalità:

1. Il parametro lh_Head della struttura List punta alla prima struttura Node della lista.

2. Il parametro lh_TailPred della struttura List punta all'ultima struttura Node della lista, quella che costituisce il nodo predecessore di coda.

3. La prima struttura Node della lista punta indietro al parametro lh_Head

della struttura List, cioè al nodo di testa. In questo nodo fittizio, il parametro che punterebbe al nodo predecessore (lh _Tail) contiene un valore nullo.

4. L'ultima struttura Node punta avanti al parametro lh_Tail della struttura List, cioè al nodo di coda. In questo nodo fittizio, il parametro che punterebbe al nodo successore (lh _Tail) contiene un valore nullo.

Questi legami sono illustrati nella Figura 1.3 e descritti più ampiamente nella spiegazione della funzione AddHead.

Occorre inizializzare opportunamente la struttura List prima di utilizzarla. Il parametro lh_Tail deve sempre e comunque contenere il valore zero perché la lista doppia possa essere gestita. Per quanto riguarda gli altri due puntatori a strutture Node, se la lista che si vuole intestare deve risultare inizialmente vuota, si devono seguire i seguenti passi:

1. Memorizzare nel parametro lh_Head l'indirizzo del parametro lh_Tail.

2. Memorizzare nel parametro lh_TailPred l'indirizzo del parametro lh_Head, cioè l'indirizzo della struttura List stessa.

3. Memorizzare il tipo di lista nel parametro lh_Type.

Ecco un esempio di definizione della struttura List in linguaggio Assembly:

```
MOVE.L AØ, (AØ)
ADDQ.L #LH_TAIL, (AØ)
CLR.L LH_TAIL(AØ)
MOVE.L AØ, LH_TAILPRED(AØ)
```

Per dare qualche idea di come vengono usate queste funzioni di gestione delle liste, si considerino i tipi di liste che le funzioni dell'Exec utilizzano per gestire i task di sistema e quelli previsti dal programmatore. Ecco una descrizione dei principali tipi di liste.

La lista di sistema TaskReady

È una lista di sistema nella quale compaiono tutti i task che sono stati aggiunti al sistema usando la funzione AddTask e che sono pronti per ricevere il controllo della CPU. Questa lista è ordinata secondo la priorità dei task. In ogni istante un solo task è in esecuzione; tutti gli altri possono trovarsi in due possibili stati: pronti per ricevere il controllo della CPU o in attesa di un evento. Gli eventi che possono accadere perché un task venga inserito nella lista TaskReady sono tre: può arrivare a una delle sue message port un messaggio del quale era entrato in attesa, può verificarsi un interrupt che lo risveglia, può essergli assegnata una più alta priorità tramite la funzione SetTaskPri.

La lista di sistema TaskReady aumenta quando nuovi task diventano pronti a ricevere il controllo della CPU, e diminuisce quando i task entrano in stato di attesa.

La lista di sistema TaskWait

È una lista di sistema nella quale compaiono tutti i task che sono stati aggiunti al sistema usando la funzione AddTask e che sono in attesa di un evento. I task che si trovano in questo stato non ricevono cicli della CPU fino a quando non tornano attivi, cioè fino a quando non si verificano gli eventi che stanno aspettando. In questa lista di sistema, contrariamente alla lista TaskReady, le strutture Task dei task non sono ordinate secondo le priorità. Un task può passare dalla lista TaskReady alla lista TaskWait in diversi modi; per esempio, quando entra in attesa di un segnale o quando chiama una funzione di accesso al video come printf del linguaggio C standard.

La lista di sistema LibList

Questa lista elenca tutte le librerie disponibili (cioè presenti in memoria) e pronte per essere aperte. Cresce a mano a mano che vengono aggiunte librerie al sistema attraverso la funzione AddLibrary; decresce invece quando le librerie vengono rimosse dal sistema con la funzione RemLibrary.

La lista di sistema DeviceList

Questa lista elenca tutti i dispositivi disponibili (cioè presenti in memoria) e pronti per essere aperti. Cresce a mano a mano che vengono aggiunti dispositivi al sistema attraverso la funzione AddDevice; decresce invece quando i dispositivi vengono rimossi dal sistema con la funzione RemDevice.

La lista di sistema MemList

Questa lista elenca tutti i blocchi di memoria libera del sistema suddividendoli per tipi. Cresce a mano a mano che vengono liberati blocchi di memoria tramite le funzioni FreeEntry e FreeMem. Si noti che il sistema non tiene una lista delle aree di memoria in uso, ma viene mantenuta soltanto la lista dei blocchi di memoria disponibili (liberi).

La lista di sistema PortList

Questa lista elenca tutte le message port dichiarate pubbliche. Cresce a mano a mano che nuove message port vengono dichiarate pubbliche tramite la funzione AddPort, decresce quando le message port pubbliche vengono rimosse con la funzione RemPort.

La lista di sistema IntrList

Questa lista elenca tutte le routine di interrupt inserite nel sistema. Cresce a mano a mano che vengono aggiunte routine di interrupt con la funzione AddIntServer, e decresce a mano a mano che le routine di interrupt vengono rimosse con la funzione RemIntServer.

La lista di sistema ResourceList

Questa lista elenca tutte le risorse disponibili nel sistema. Cresce a mano a mano che vengono aggiunte risorse al sistema tramite la funzione AddResource, e decresce a mano a mano che le risorse vengono rimosse con la funzione RemResource.

AddTask

Sintassi di chiamata della funzione

```
AddTask (taskCS, initialPC, finalPC)
          A1      A2        A3
```

Scopo della funzione

Questa funzione aggiunge un nuovo task al sistema. Quando un task desidera aggiungere un altro task al sistema, prima di chiamare AddTask deve inizializzare alcuni parametri della struttura Task e deve aver allocato uno stack di dimensioni opportune. Generalmente è consigliabile non scendere sotto i cento byte.

AddTask usa un po' di spazio nell'area stack del nuovo task per memorizzarvi lo stato iniziale dei registri; in questo modo, appena prima che il task riceva il controllo della CPU questi dati vengono automaticamente copiati nei registri della CPU per creare una situazione di default. Negli scambi di controllo successivi, ogni volta che il task perde il controllo, i valori dei registri della CPU vengono sempre salvati nello stack del task per essere poi ripristinati quando il task riottiene il controllo. AddTask memorizza i valori di default dei registri a partire dalla posizione nello stack indicata dal parametro SPReg della struttura Task (attenzione, non da SPUpper). Questo significa che prima di chiamare AddTask è possibile inserire nello stack dati statici, in modo che il task possa leggerli facilmente quando riottiene il controllo. AddTask tiene conto di quest'eventualità e non parte dal presupposto che lo stack del

nuovo task sia vuoto. Questo risulta molto utile per predisporre variabili globali prestabilite che possono poi essere utilizzate dal task per soddisfare alcune delle sue necessità di elaborazione dati. In aggiunta, alcuni task possono usare questo spazio per passare al nuovo task i suoi argomenti iniziali. I valori iniziali dei registri della CPU sono impostati a zero, con l'esclusione del PC (program counter) del 68000.

Argomenti della funzione

taskCS Indirizzo della struttura Task che definisce il task da aggiungere nel sistema.

initialPC Indirizzo del punto d'ingresso dei codici del task.

finalPC Indirizzo dei codici del task adibiti alla chiusura del task. Se vale zero, quando verrà richiesta la rimozione del task il sistema userà automaticamente una propria routine generale di rimozione. Questo indirizzo viene posto nello stack del task nella posizione più bassa, in modo che rappresenti il più remoto indirizzo di ritorno per il task.

Discussione

Ci sono quattro funzioni dell'Exec preposte alla gestione dei task nel sistema Amiga: AddTask, FindTask, RemTask e SetTaskPri. Si vedano anche le spiegazioni di queste funzioni.

Il ROM Kernel dell'Amiga è un ambiente che opera in tempo reale, basato su messaggi e gestione multitasking delle risorse. Tuttavia è importante comprendere che le funzioni Exec non costituiscono un sistema operativo nel senso usuale del termine. Più vicino alla definizione di sistema operativo è invece l'AmigaDOS.

Essendo predisposte per funzionare in tempo reale, le funzioni dell'Exec sono in grado di rispondere agli eventi non appena questi si verificano, dal momento che sono progettate per completarsi nel minor tempo possibile. Inoltre, gli speciali chip dedicati dell'Amiga assistono le operazioni che coinvolgono i task e le rendono particolarmente veloci. In particolare, il canale DMA (direct memory access, memoria ad accesso diretto) del Blitter può trasferire dati a velocità molto elevata. Le operazioni di sistema risultano anche accelerate dalla presenza di altri canali DMA (fino a un massimo di 24).

Essendo multitasking, le funzioni dell'Exec rendono possibile lo svolgimento concorrente di diversi programmi, senza che ciascuno debba tener conto degli altri. Per esempio, un task può segnalare a un altro di aver bisogno di informazioni da disco. Quando questo sotto-task di accesso al disco inizia a

lavorare, ci sarà nel sistema un task correlato al disco. Mentre il task di accesso al disco attende che la testina di lettura/scrittura del disk drive giunga sulla traccia richiesta, un altro task può svolgere parte del suo lavoro. Ciò evita la perdita di tempo che si verifica mentre la testina di lettura/scrittura si sta muovendo, e mantiene il sistema sempre occupato a svolgere operazioni utili, senza inefficienti loop di attesa.

Quando si inserisce un task nel sistema, occorre attribuirgli una priorità d'esecuzione, che dev'essere un numero compreso tra +127 (priorità massima) e -128 (priorità minima). La priorità di un task dev'essere memorizzata nel parametro ln_Pri della sotto-struttura Node (di nome tc_Node) che costituisce il primo elemento della struttura Task. Nello scambio di controllo fra i task condotto dall'Exec, task eseguibili dotati della stessa priorità ricevono porzioni di tempo uguali della CPU.

Le funzioni dell'Exec sono basate sullo scambio di messaggi; i task possono comunicare fra loro attraverso i messaggi. Ciascun task può avere una o più message port dove vengono inviati i messaggi. I messaggi che giungono a una message port vengono messi in coda ed elaborati in ordine FIFO. Quando arriva un messaggio, il relativo task può ricevere un segnale di cui era in attesa e quindi riottenere il controllo della CPU, oppure può essere mandata in esecuzione una speciale routine di exception: il modo in cui si manifesta l'evento corrispondente all'arrivo di un messaggio dipende dal task che ha creato la message port.

Per inviare un messaggio si usa la funzione PutMsg, la quale lo accoda alla message port destinataria. Quando il task che possiede la message port ottiene il controllo, può chiamare la funzione GetMsg per prelevare il messaggio. PutMsg può svolgere una delle seguenti due azioni:

- può attivare il task impostando uno dei suoi bit di segnale, quello associato alla message port destinataria del messaggio. Se il task era in attesa di quel segnale, riottiene il controllo.

- Può attivare il task e causare un interrupt software. Ciò forza l'esecuzione della routine di exception del task. Tuttavia questo accade soltanto se il segnale associato alla message port è stato predisposto per causare exception. Quando l'interrupt è terminato, il task riprende l'esecuzione dal punto a cui era arrivato prima che si verificasse l'interrupt.

Per definire e aggiungere un task al sistema, si definisce per prima cosa la relativa struttura Task, la quale viene poi usata dal sistema per gestire il task. I parametri da definire sono quelli relativi allo stack del task, alla routine di exception, alla routine di trap, all'allocazione della memoria e a tutti gli altri parametri necessari per definire il task.

Quando questo lavoro è stato svolto, si procede ad aggiungere il task al sistema chiamando la funzione AddTask. Da quel momento in poi, il task condivide le risorse del sistema con tutti gli altri task già presenti. Nella creazione di un task è necessario rispondere alle seguenti domande:

1. Dove si trova la struttura Task relativa al task considerato? La funzione AddTask richiede come primo argomento l'indirizzo della struttura Task, che dev'essere allocata in memoria e inizializzata prima di chiamare la funzione AddTask.

2. Qual è l'indirizzo d'inizio dei codici del task? La funzione AddTask richiede questo indirizzo come secondo argomento. I codici del task possono trovarsi in qualsiasi area RAM, inclusa la chip RAM.

3. Qual è l'indirizzo d'inizio della routine di rimozione che questo task dovrebbe usare quando ha terminato il suo lavoro? La funzione AddTask richiede questo indirizzo come terzo argomento.

La struttura dati che definisce la struttura Task appare nei file INCLUDE tasks.h (per il linguaggio C) e tasks.i (per il linguaggio Assembly). Per allocarla in memoria si possono impiegare le seguenti istruzioni:

```
struct Task *mioTask;
mioTask = (struct Task *) AllocMem ((LONG) sizeof (struct Task), MEMF_PUBLIC
| MEMF_CLEAR);
```

La variabile puntatore mioTask viene così a contenere l'indirizzo della struttura Task appena allocata. Ora occorre impostarne alcuni parametri prima di poter chiamare la funzione AddTask per aggiungere il task al sistema. Eccone l'elenco.

1. Parametri di definizione dello stack:

tc_SPUpper	deve contenere l'indirizzo del byte superiore dello stack;
tc_SPLower	deve contenere l'indirizzo del byte inferiore dello stack;
tc_SPReg	deve contenere il valore iniziale del registro stack pointer che il sistema usa per gestire lo stack del task e individuarne gli elementi.

2. Parametri di definizione della routine di exception:

tc_ExceptCode	deve contenere l'indirizzo della routine di exception di questo task; deve valere zero se non sono previste routine di exception;

tc_ExceptData — deve contenere l'indirizzo dell'area dati della routine di gestione dell'exception per il task; deve valere zero se non sono previste routine di exception;

tc_SigExcept — maschera di bit che informa il sistema su quali segnali sono destinati a causare una exception; dev'essere azzerato se non sono previste routine di exception;

tc_SigRecvd — quando il task riceve un segnale, il sistema imposta il bit corrispondente in questo parametro. La funzione Wait controlla il contenuto di questo parametro per rilevare se il segnale pervenuto è fra quelli indicati nel suo argomento, cioè se è fra quelli che devono risvegliare il task. Se il controllo ha esito positivo, provvede a copiare il contenuto del parametro nel registro D0 e ad azzerarlo; il registro D0 contiene il valore che viene restituito al task che ha chiamato la funzione Wait. Generalmente i task non hanno nessuna necessità di accedere direttamente al parametro tc_SigRecvd, che dev'essere quindi azzerato prima di chiamare AddTask;

tc_SigAlloc — è un insieme di 32 bit di segnale nel quale quelli a 1 indicano i segnali allocati; quando si inserisce il task nel sistema con la funzione AddTask, questo valore dev'essere impostato a zero.

3. Parametri di definizione della routine di trap:

tc_TrapCode — deve contenere l'indirizzo d'inizio della routine di trap per il task; dev'essere azzerato se non è prevista una routine di trap;

tc_TrapData — deve contenere l'indirizzo dell'area dati della routine di trap per il task; dev'essere azzerato se non è prevista una routine di trap;

tc_TrapAlloc — parametro che informa il sistema sulle trap già allocate;

tc_TrapAble — parametro che informa il sistema sulle trap abilitate in quel particolare momento.

4. Parametri di allocazione della memoria:

tc_MemEntry — Questo è un puntatore a una struttura List che dice al sistema quali locazioni di memoria devono essere

allocate per il nuovo task. Tali aree di memoria possono anche essere liberate più tardi, durante l'esecuzione del task. Se non è necessario che il sistema allochi memoria per il task, questo parametro dev'essere azzerato. È importante notare che se prima d'inserire il task nel sistema si indicano in questa lista i blocchi di memoria che devono essere allocati automaticamente dal sistema, quando poi il task termina, il sistema accede al parametro tc_MemEntry per liberare la memoria allocata all'avvio del task. Questo vale anche quando la lista tc_MemEntry cresce dinamicamente durante l'esecuzione del task.

5. Altri parametri: in generale, tutti gli altri parametri della struttura Task si impostano a zero prima che il task venga aggiunto al sistema con la funzione AddTask.

La struttura Task

La struttura Task è definita come segue:

```
struct Task {
        struct Node tc_Node;
        UBYTE tc_Flags;
        UBYTE tc_State;
        BYTE tc_IDNestCnt;
        BYTE tc_TDNestCnt;
        ULONG tc_SigAlloc;
        ULONG tc_SigWait;
        ULONG tc_SigRecvd;
        ULONG tc_SigExcept;
        UWORD tc_TrapAlloc;
        UWORD tc_TrapAble;
        APTR tc_ExceptData;
        APTR tc_ExceptCode;
        APTR tc_TrapData;
        APTR tc_TrapCode;
        APTR tc_SPReg;
        APTR tc_SPLower;
        APTR tc_SPUpper;
        VOID (*tc_Switch)();
        VOID (*tc_Launch)();
        struct List tc_MemEntry;
        APTR tc_UserData;
};
```

I parametri della struttura Task sono i seguenti:

- il parametro tc_Node è la sotto-struttura Node che permette alla struttura Task di essere inserita in una lista di sistema. Il parametro ln_Name della struttura Node contiene l'indirizzo della stringa che rappresenta il nome del task; i task che sono dotati di nome possono essere rintracciati tramite la funzione FindTask. Il parametro ln_Pri della struttura Node deve contenere la priorità che si desidera assegnare al task. Questi due parametri dovrebbero essere definiti prima d'inserire il task nel sistema. Si dovrebbe anche impostare il parametro ln_Type della struttura Node a NT_TASK per indicare che la struttura Node rappresenta un task.

- Il parametro tc_Flags contiene un insieme di flag per il task. Ciascun task può avere fino a cinque flag. Alcuni o tutti questi flag devono essere impostati prima che il task venga inserito nel sistema.

- Il parametro tc_State riporta continuamente lo stato del task. I task possono trovarsi in cinque stati diversi (si veda più avanti). Il sistema aggiorna automaticamente il parametro.

- Il parametro tc_IDNestCnt è il contatore del numero di volte che gli interrupt sono stati disabilitati per il task rappresentato dalla struttura Task in questione. Ogni volta che un task avvia una routine che disabilita gli interrupt, questo parametro viene incrementato. Gli interrupt vengono abilitati appena prima che la routine restituisca il controllo, decrementando in tal modo il parametro. Il parametro viene anche aggiornato direttamente dalle routine di sistema Enable e Disable.

- Il parametro tc_TDNestCnt contiene il contatore di annidamento del numero delle chiamate alle funzioni Forbid e Permit. Questo parametro viene aggiornato automaticamente dal sistema ogni volta che vengono chiamate queste routine di sistema.

- Il parametro tc_SigAlloc rappresenta l'insieme dei segnali allocati dal task. Ci sono 32 segnali (quindi bit) disponibili per ogni task, ma i 16 bit più bassi (i bit da 0 a 15) sono riservati al sistema. Il task può allocare gli altri 16 (i bit da 16 a 31). Quando il task chiama la funzione AllocSignal, questa provvede a impostare uno di questi bit per indicare che il corrispondente segnale è da considerare già allocato. Al contrario, FreeSignal provvede ad azzerare il bit corrispondente a un particolare segnale.

- Il parametro tc_SigWait rappresenta l'insieme di segnali per i quali il task è entrato in attesa, e viene impostato dalla funzione Wait con il valore del suo argomento. Tale insieme dovrebbe sempre essere un sotto-insieme dei 16 segnali disponibili ai task (i bit da 16 a 31). In genere i task non hanno necessità di accedere a questo parametro, che viene impiegato essenzialmente dal sistema.

- Il parametro tc_SigRecvd è sotto il controllo del sistema. Quando un task riceve un segnale, il sistema imposta a 1 il corrispondente bit del parametro tc_SigRecvd, in modo che la funzione Wait possa rilevare quale segnale è pervenuto. Se la funzione Wait rileva l'arrivo di uno dei segnali indicati nel suo argomento, provvede ad azzerare il parametro tc_SigRecvd prima di restituire il controllo. In genere i task non hanno necessità di accedere a questo parametro, che viene impiegato essenzialmente dal sistema.

- Il parametro tc_SigExcept rappresenta l'insieme dei segnali che devono causare un'exception. Tale insieme è rappresentato come una maschera di bit. Ci sono 32 possibili bit di segnale e 32 corrispondenti bit di exception. Ancora una volta i 16 bit bassi della long word sono riservati al sistema, mentre gli altri 16 possono essere usati dal task. Questo parametro opzionale dovrebbe essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_TrapAlloc rappresenta l'insieme delle trap allocate per il task rappresentato da questa struttura Task. Ciascuna trap dovrebbe avere una corrispondente routine per provvedere alla sua gestione. Tale parametro opzionale dovrebbe essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_TrapAble rappresenta l'insieme dei numeri di trap abilitati per il task. Tale parametro opzionale dovrebbe essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_ExceptData deve contenere l'indirizzo di un'area dati da utilizzare con la routine di exception associata al task. La routine di exception usa questi dati quando riceve il controllo, ovviamente al verificarsi di un'exception. Se si vuole che il task elabori exception che prevedono dati, questo parametro opzionale dev'essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_ExceptCode deve contenere l'indirizzo della routine di exception da associare al task. Se si desidera che il task elabori le exception, questo parametro opzionale deve'essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_TrapData è un puntatore a un insieme di dati di trap per il task. Se si desidera che il task elabori le trap attingendo a dati, questo parametro opzionale dev'essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_TrapCode è un puntatore alla routine di trap per il task. Se si vuole che il task elabori le trap, questo parametro opzionale dev'essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_SPReg contiene il valore dello stack pointer del task. Le routine di sistema aggiornano questo valore, incrementandolo o decrementandolo quando nuovi dati vengono prelevati o inseriti nello stack del task. Prima d'inserire il task nel sistema, è necessario inserire un valore in questo parametro. Se lo stack non è necessario, occorre inizializzare questo parametro con lo stesso valore memorizzato nel parametro tc_SPUpper.

- Il parametro tc_SPLower contiene il valore del confine inferiore e il parametro tc_SPUpper contiene il valore del confine superiore dello stack pointer del task. Questi valori rimangono costanti durante l'esecuzione del task. Tali parametri devono essere impostati prima d'inserire il task nel sistema.

- Il parametro tc_Switch può essere inizializzato con l'indirizzo di una routine definita dal programmatore e destinata a prendere il controllo della macchina quando il task perde il controllo della CPU. Questo parametro opzionale dev'essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_Launch può essere inizializzato con l'indirizzo di una routine definita dal programmatore e destinata a prendere il controllo della macchina quando il task sta per acquisire il controllo della CPU. Se il parametro non contiene un valore nullo, la routine riceve il controllo appena prima del task a ogni scambio di controllo della CPU. Questo parametro opzionale dev'essere impostato prima d'inserire il task nel sistema.

- Il parametro tc_MemEntry è una struttura List che dice al sistema quanta memoria dev'essere allocata e assegnata al task quando viene inserito nel sistema, e liberata automaticamente quando il task viene rimosso. Questa lista può anche crescere dinamicamente durante l'esecuzione del task.

- Il parametro tc_UserData è un puntatore a un insieme di dati dell'utente che possono essere usati esclusivamente da questo task.

Il parametro tc_Flags della struttura Task contiene i seguenti flag:

- TF_PROCTIME. Se è impostato, il sistema tiene conto del tempo totale durante il quale la CPU manda in esecuzione codici del task, cioè registra automaticamente (ogni volta che si verifica uno scambio di controllo fra i task) il tempo di CPU usato dal task.

- TF_STACKCHK. Se è impostato il sistema mantiene un continuo controllo sulla condizione dello stack del task.

- TF_EXCEPT. Il sistema imposta questo flag quando è in esecuzione la routine di exception del task.

- TF_SWITCH. Si deve impostare questo flag se si vuole avvertire il sistema che esiste un vettore di salto coerente nel parametro tc_Switch() della struttura Task; in questo caso bisogna anche definire opportunamente il parametro tc_Switch().

- TF_LAUNCH. Si deve impostare questo flag se si vuole avvertire il sistema che esiste un vettore di salto coerente nel parametro tc_Launch() della struttura Task; in questo caso bisogna anche definire il parametro tc_Launch().

Gli stati dei task

Un task nello stato TS_RUN detiene il controllo della CPU. Un solo task per volta può detenere il controllo della CPU. Task a uguale priorità condividono uguali porzioni di tempo della CPU.

Un task in stato TS_READY è pronto per ricevere il controllo della CPU, quindi non lo detiene. Non è in attesa di un messaggio, di un segnale o più in generale di un evento, ma non detiene il controllo della CPU a causa della sua bassa priorità o a causa di alcune altre condizioni che caratterizzano il sistema (per esempio, un interrupt in elaborazione) e che richiedono l'esecuzione di qualche altro task. I task nello stato TS_READY vengono elencati nella lista di sistema TaskReady.

Un task nello stato TS_WAIT sta aspettando che si verifichi un evento, in genere l'arrivo di un segnale, e pertanto non riceve cicli della CPU. I task che si trovano in questo stato sono elencati nella lista di sistema TaskWait. Il segnale può provenire da qualsiasi sorgente (per esempio da una message port di proprietà del task). In genere i task chiamano la funzione Wait ed entrano in attesa volontaria, così da non impegnare la CPU, quando devono attendere che si verifichi un certo evento per proseguire.

Un task nello stato TS_ADDED è appena stato aggiunto al sistema tramite la funzione AddTask e non ha ancora ricevuto il controllo della CPU. Generalmente si tratta di uno stato transitorio.

Un task è nello stato TS_EXCEPT quando è in esecuzione una delle sue routine di exception. Significa che è accaduto qualcosa che ha causato la sospensione temporanea dell'esecuzione del task in favore della routine di exception. Questo stato perdura fino a quando la routine di exception non ha terminato la propria esecuzione.

Un task nello stato TS_REMOVED non si trova né nella lista TaskReady né nella lista TaskWait. Generalmente questo accade quando viene rimosso tramite la funzione RemTask. Tutte le risorse di sistema usate da questo task sono state liberate per poter essere impiegate da altri task.

La Figura 1.5 mostra il comportamento dello stack nelle varie fasi che un task attraversa nel corso della sua esistenza. Si noti che lo stack del task cresce

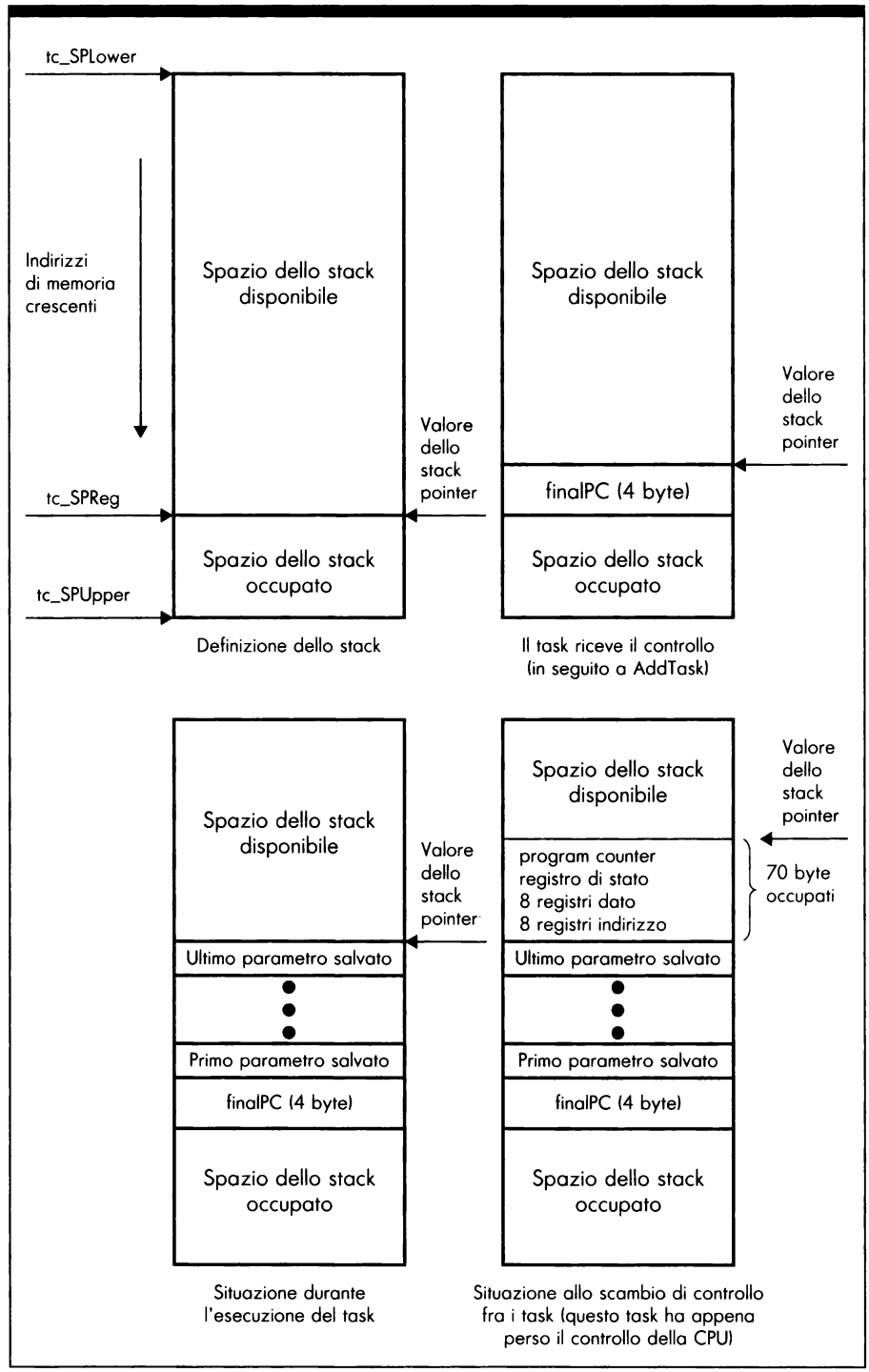


Figura 1.5: *Comportamento dello stack del task*

in memoria verso il basso, in direzione degli indirizzi inferiori (cioè in direzione di tc_SPLower). Lo stack pointer è rappresentato in ogni momento da tc_SPReg; il sistema mantiene automaticamente aggiornato lo stack pointer di ogni task. Quando non si desidera utilizzare lo stack, è opportuno che tc_SPReg venga aggiornato con lo stesso indirizzo memorizzato nel parametro tc_SPUpper.

Il primo rettangolo della figura, in alto a sinistra, mostra lo stato dello stack quando vengono impostati i parametri della struttura Task. Si noti che si può impostare tc_SPReg a un valore inferiore rispetto a tc_SPUpper per assicurare che parte dello stack rimanga "protetta" durante l'esecuzione del task. Questa sezione protetta può essere inizializzata con un insieme di valori ai quali il task, quando riceve il controllo, può poi accedere per ottenere i necessari dati iniziali (questi valori sono memorizzati nelle locazioni di memoria comprese tra tc_SPReg e tc_SPUpper). Questi dati iniziali possono essere alterati durante l'esecuzione del task.

Il secondo rettangolo della Figura 1.5, in alto a destra, rappresenta lo stato dello stack appena dopo l'esecuzione della funzione AddTask. Questo rettangolo mostra che il valore del parametro finalPC, specificato come terzo argomento nella chiamata della funzione AddTask, è stato inserito nello stack. Come conseguenza, il valore dello stack pointer è diminuito di quattro byte per riflettere l'aumento della quantità di parametri presenti nello stack. Inserendo questo indirizzo sullo stack, quando la routine principale del task termina la propria esecuzione e restituisce il controllo, questo passa alla routine il cui indirizzo è finalPC.

Il terzo rettangolo, quello in basso a sinistra, rappresenta lo stack del task dopo che ha ricevuto il controllo della CPU. Per mostrare l'esecuzione in corso, la figura ipotizza che il task abbia salvato alcuni dati sullo stack (e magari che ne abbia recuperati altri); sullo stack possono anche trovarsi indirizzi di ritorno salvati durante le chiamate alle sotto-routine del task. Il primo dato salvato sullo stack è appena sotto al valore finalPC. L'ultimo dato salvato si trova nella posizione indicata dallo stack pointer. Si noti che l'area dati, quella che contiene i dati iniziali quando il task riceve il controllo per la prima volta, può anche essere stata modificata in dimensioni e contenuto dal task nel corso della sua elaborazione.

L'ultimo rettangolo nella Figura 1.5, quello in basso a destra, rappresenta lo stack immediatamente dopo che il task ha perso il controllo della CPU in seguito a uno scambio di controllo. Questo può accadere in qualsiasi fase dell'elaborazione, senza preavviso; in effetti, lo scambio di controllo fra i task viene eseguito dal sistema senza che i task possano rilevarlo. Il task è ora inattivo ma pronto a riottenere il controllo alla prima occasione. Per conservare l'ambiente in cui il task è stato interrotto, il sistema salva automaticamente 70 byte d'informazioni sullo stack del task. I primi elementi salvati sono i valori contenuti nei registri d'indirizzamento del 68000 (otto valori da 4 byte l'uno). Gli elementi seguenti sono i valori contenuti nei registri dati del 68000 (altri otto valori da 4 byte l'uno). Segue il valore contenuto nel registro di stato del 68000 (2 byte). Infine viene salvato sullo stack il program counter del 68000 (4 byte). Compiuta questa operazione di "conservazione" il sistema è pronto per cedere il controllo a un altro task; quando sarà di nuovo il turno di questo task, il sistema provvederà a prelevare dallo stack questi 70 byte e a ripristinare lo

stato dell'intera CPU, in modo che il task riottenga il controllo senza che nulla sia cambiato. Una conseguenza di questa gestione multitasking è che i task devono disporre di stack di capienza sufficiente da avere sempre 70 byte liberi. Il valore dello stack pointer, invece, viene salvato nella struttura Task del task e non nello stack. In questo modo il sistema, quando decide di cedere nuovamente il controllo della CPU a questo task, non deve fare altro che accedere alla relativa struttura Task e prelevare gli indirizzi di gestione dello stack.

Questo è l'uso che viene fatto dello stack del task (che viene chiamato anche "user stack"). Si tenga presenta che quanto detto vale per lo stack di qualunque task del sistema, senza eccezioni.

AllocAbs

Sintassi di chiamata della funzione

```
blocco = AllocAbs (dimensioni, indirizzo)
D0               D0          A1
```

Scopo della funzione

Questa funzione tenta di allocare memoria a un indirizzo assoluto specifico. In caso di esito positivo restituisce l'indirizzo del blocco allocato (eventualmente arrotondato per difetto a un multiplo della dimensione minima dei blocchi, attualmente 8 byte), altrimenti restituisce il valore zero.

Argomenti della funzione

dimensioni	le dimensioni in byte del blocco da allocare. Questo numero viene arrotondato per eccesso a un multiplo di 8 byte.
indirizzo	l'indirizzo al quale deve trovarsi la memoria allocata.

Discussione

Una delle caratteristiche fondamentali dell'Amiga è quella di non fare quasi mai uso di indirizzi assoluti. Grazie alla particolare struttura del sistema,

dal solo indirizzo immutabile (AbsExecBase, l'indirizzo $4, contenente un puntatore alla base della libreria Exec), sono ricavabili tutte le informazioni necessarie al funzionamento della macchina. Tramite le opportune chiamate alle funzioni di libreria, nessun task ha la necessità di conoscere in anticipo la posizione in memoria di una particolare struttura dati, posizione che può variare su macchine aventi una diversa configurazione, o addirittura a ogni reset.
È però possibile che si verifichi una situazione in cui un task ha la necessità di operare con indirizzi assoluti. Consideriamo il caso, per esempio, di un RAM disk recuperabile, che conserva cioè i suoi dati anche dopo un reset. Questi dati si troveranno in memoria in posizioni ben precise, conosciute dal task che controlla il RAM disk, ma, dopo il reset, figureranno anche nella lista della memoria libera, reinizializzata dal processo di boot. Non appena riceve il controllo, quindi, il task cercherà presumibilmente di riottenere il controllo di quella stessa memoria, prima che possa essere allocata e utilizzata da altri task. Ciò può essere ottenuto tramite la funzione AllocAbs.

La funzione tenta di allocare l'area richiesta, eventualmente arrotondandola per eccesso per allinearla correttamente. Se tutto il blocco corrispondente è ancora libero, viene allocato, e AllocAbs ne restituisce l'indirizzo iniziale. In ogni altro caso l'allocazione fallisce e la funzione restituisce il valore zero.

Allocate

Sintassi di chiamata della funzione

```
memBlock = Allocate (memHeader, byteSize)
D0                  A0          D0
```

Scopo della funzione

Questa funzione serve per allocare un blocco di memoria da un pool privato di memoria libera di proprietà del task. Restituisce l'indirizzo del primo blocco di memoria di dimensione uguale o maggiore a quella richiesta nella chiamata della funzione. Viene impiegata quando un task possiede un'area di memoria di sistema allocata per sé, e la gestisce come una riserva privata di memoria libera. Per effettuare questa gestione, deve anche possedere una lista dei chunk di memoria libera dell'area privata. Tramite le funzioni Allocate e Deallocate alloca e libera memoria da quest'area privata, e queste due funzioni si preoccupano di mantenere aggiornata la lista privata di memoria libera del task. Sono molto simili ad AllocMem e a FreeMem, con la differenza che queste ultime lavorano con la lista di memoria libera del sistema.

Tutti i blocchi di memoria, sia liberi che allocati, sono sempre multipli del

blocco minimo, che è di otto byte. Ciò significa che tutte le dimensioni per l'allocazione di memoria sono sempre arrotondate al successivo multiplo di otto. Questo vale per tutti i problemi di allocazione della memoria, sia per la lista di memoria libera del sistema, sia per la lista privata di memoria libera di un task.

Allocate, quando viene usata in congiunzione con una lista privata di memoria libera, contribuisce alla gestione della memoria posseduta da un task.

Se non ci sono regioni libere abbastanza ampie da soddisfare la richiesta di allocazione, oppure l'ammontare della memoria richiesta è un valore non coerente, Allocate restituisce il valore zero.

Argomenti della funzione

memHeader Indirizzo della struttura MemHeader che intesta la serie privata di chunk di memoria libera del task; ogni chunk è un blocco continuo di memoria libera. La struttura MemHeader può anche essere legata ad altre strutture, per differenziare aree di memoria con caratteristiche diverse (chip RAM e fast RAM, per esempio).

byteSize Dimensione in byte del blocco di memoria che la funzione deve allocare.

Discussione

Nella libreria Exec esistono nove funzioni di gestione della memoria: AllocAbs, Allocate, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem.

Ci sono diversi aspetti che dovrebbero essere tenuti presenti a proposito della gestione della memoria dell'Amiga. Prima di tutto, il sistema mantiene una lista di memoria libera che lo mette in grado di sapere quali regioni di memoria sono libere per effettuare allocazioni. Questa lista di sistema si chiama MemList, e riflette lo stato della memoria a mano a mano che nel sistema viene allocata e liberata memoria. Le funzioni AllocMem e FreeMem permettono di allocare e liberare memoria nell'intero sistema, e quindi usano la lista di memoria libera gestita dal sistema, che non è controllabile dai task.

Le funzioni AllocEntry e FreeEntry possono essere usate per allocare e liberare in un colpo solo diversi blocchi di memoria spaiati, ognuno con dimensioni e attributi propri. In contrasto, le funzioni Allocate, Deallocate, AllocMem e FreeMem manovrano soltanto un blocco di memoria per volta.

Diversamente dalle funzioni AllocMem e FreeMem, le funzioni Allocate e Deallocate permettono ai task di gestire una particolare area di memoria di loro proprietà, cioè allocata tramite la funzione AllocMem, come se fosse un pool di

memoria libera. Per il sistema quest'area risulta allocata, e il task la può poi gestire come un'area privata di memoria libera. Queste funzioni vanno quindi utilizzate da parte dei task con opportune liste private di memoria libera. Un argomento richiesto da queste funzioni è infatti l'indirizzo di una struttura MemHeader.

Un task può decidere di gestire un'area privata di memoria libera quando desidera essere certo che in qualunque momento della propria esecuzione ci sia memoria libera sufficiente per soddisfare le sue esigenze, sicurezza che non potrebbe avere confidando nella memoria libera di sistema. In questo modo il task e tutti i suoi sotto-task possono condividere la stessa area privata di memoria libera, senza possibilità d'interferenza da parte del "mondo esterno". Un aspetto fondamentale di questo tipo di gestione riguarda il grado di frammentazione della memoria di sistema, che verrebbe sensibilmente ridotto se tutti i task utilizzassero pool privati di memoria libera. Analizziamo meglio il problema.

La frammentazione della memoria è un fenomeno che aumenta a mano a mano che nel sistema si susseguono e si accavallano molteplici task, ognuno con più o meno grandi necessità di memoria. Si verifica in quanto il sistema non può rilocare in un unico blocco continuo i blocchi di memoria che in un dato momento risultano liberi se tra loro vi sono blocchi di memoria allocata, per quanto piccoli siano questi ultimi. Dopo un periodo d'intensa attività della macchina, possono quindi presentarsi situazioni paradossali nelle quali la quantità totale di memoria libera, pur essendo elevata, è talmente frammentata in piccoli blocchi da non poter soddisfare nemmeno le normali esigenze di gestione di un task.

A questa situazione non c'è rimedio, in quanto i blocchi di memoria allocati non sono rilocabili, e le dimensioni dei blocchi continui più grandi sono degli ottimi indicatori del grado di disordine raggiunto dal sistema. Inoltre, la frammentazione della memoria causa anche veri e propri sprechi di memoria. Si può limitare il fenomeno creando task che si approprino subito di tutta la memoria che prevedono d'impiegare, e che gestiscano questa memoria con una lista privata di memoria libera tramite le funzioni Allocate e Deallocate. In questo modo, agli occhi del sistema risulta che il task ha allocato un unico grande blocco di memoria, mentre all'interno del task la frammentazione dovuta alle allocazioni e ai rilasci di memoria si restringe allo spazio privato di memoria libera che ha allocato dal pool di memoria libera del sistema. Quando il task termina di esistere, un grande blocco continuo di memoria torna a far parte della memoria libera di sistema al posto di tanti piccoli blocchi sparsi qua e là. Ecco perché è consigliabile che i task si servano di Allocate e Deallocate anziché delle funzioni di gestione della memoria che utilizzano la lista di sistema della memoria libera.

Nel sistema Amiga ci sono quattro attributi specificabili quando si alloca memoria: MEMF_CHIP, MEMF_FAST, MEMF_PUBLIC e MEMF_CLEAR. L'attributo MEMF_CHIP dice al sistema che si richiede di allocare memoria nella chip RAM; questa memoria è detta *chip* perché si trova nello spazio indirizzabile dai chip dedicati dell'Amiga; proprio per via di questo doppio accesso (la CPU e i chip dedicati) questa memoria è più lenta rispetto alla fast RAM. L'attributo MEMF_FAST specifica al sistema che si desidera allocare

memoria nella fast RAM; questa memoria è detta *fast* (veloce) in quanto i chip dedicati non vi accedono e quindi i cicli di accesso sui bus dati e indirizzi sono a completa disposizione della CPU. L'attributo MEMF_PUBLIC indica al sistema che la memoria in oggetto non dev'essere rilocata o comunque resa non indirizzabile. Future revisioni del sistema potrebbero infatti avvalersi dell'uso di una MMU (memory management unit, unità di gestione della memoria), che, allo scopo di preservare l'integrità dell'area di dati o di codice di un task, ne impedisca fisicamente l'accesso agli altri task. Per informare perciò il sistema che la memoria da allocare è pubblica, ossia accessibile da chiunque, è stato previsto l'uso di questo flag, che, per garantire la compatibilità con versioni future dell'Amiga, *deve* essere usato per allocare strutture in qualche modo condivise fra più task. Per esempio la struttura MsgPort o la struttura Interrupt.

L'attributo MEMF_CLEAR, infine, specifica al sistema che oltre ad allocare memoria deve anche azzerarla. Ogni volta che un task deve allocare memoria, può combinare questi attributi nel modo più opportuno per le sue necessità.

La struttura MemHeader

La funzione Allocate si aspetta come primo argomento della chiamata l'indirizzo di una struttura di tipo MemHeader. La struttura MemHeader è definita come segue:

```
struct MemHeader {
    struct Node mh_Node;
    UWORD mh_Attributes;
    struct MemChunk *mh_First;
    APTR mh_Lower;
    APTR mh_Upper;
    ULONG mh_Free;
};
```

Ecco i parametri della struttura MemHeader:

- il parametro mh_Node è la sotto-struttura Node della struttura MemHeader, tramite la quale è possibile concatenare in un'unica lista diverse strutture MemHeader. Il parametro ln_Name di questa struttura Node individua il nome dell'area di memoria rappresentata dalla struttura MemHeader. Il parametro ln_Pri nella struttura Node determina la posizione della struttura nella lista di memoria libera in cui si trova. Il parametro ln_Type dovrebbe essere impostato a NT_MEMORY.

- Il parametro mh_Attributes contiene gli attributi per quest'area di memoria. Si veda il paragrafo precedente e la spiegazione della funzione AllocMem.

- Il parametro mh_First è un puntatore alla struttura MemChunk che rappresenta il primo blocco (detto anche chunk) di memoria libera dell'area individuata dalla struttura MemHeader.
- Il parametro mh_Lower contiene l'indirizzo del primo byte dell'area di memoria associata alla struttura MemHeader.
- Il parametro mh_Upper contiene l'indirizzo dell'ultimo byte dell'area di memoria associata alla struttura MemHeader, aumentato di uno.
- Il parametro mh_Free contiene il numero totale di byte liberi presenti nell'area associata alla struttura MemHeader.

AllocEntry

Sintassi di chiamata della funzione

```
memList = AllocEntry (memList)
D0                   A0
```

Scopo della funzione

Questa funzione alloca tutti i blocchi di memoria indicati nella struttura MemList che il task ha indicato come argomento. Il task alloca una struttura MemList inizializzandone i parametri, e fa in modo che in memoria sia seguita da tante strutture MemEntry quanti sono i blocchi di memoria che vuole allocare. Ogni struttura MemEntry è in pratica un elemento del vettore ml_ME della struttura MemList, e descrive le caratteristiche che deve avere il relativo blocco di memoria (attributi e dimensione). Una volta che è stata creata la struttura MemList, si chiama la funzione AllocEntry perché tutti i blocchi di memoria descritti vengano allocati. La funzione scandisce a uno a uno gli elementi dell'array ml_ME e alloca il quantitivo di memoria indicato e con gli attributi specificati.

AllocEntry restituisce l'indirizzo di un'altra struttura MemList che ha provveduto a creare e ad aggiornare con gli indirizzi dei vari blocchi di memoria allocati. In essa, ogni elemento del vettore ml_ME indica nel parametro me_Addr (il primo di ogni struttura MemEntry) l'indirizzo del relativo blocco di memoria allocato. In questo modo, il task può accedere alla struttura MemList creata da AllocEntry per sapere quali sono gli indirizzi di memoria di tutti i blocchi che ha richiesto. Si noti che questa struttura MemList può essere indicata a FreeEntry per liberare in un colpo solo tutti i blocchi allocati da

AllocEntry.

Le strutture MemList possono anche essere legate insieme per comporre una lista da associare alla struttura Task del task, tramite il parametro tc_MemEntry, per mantenere sotto controllo la memoria utilizzata dal task. Così facendo, si delega al sistema la liberazione di quella memoria quando il task conclude la propria esecuzione.

Se la funzione non riesce a ottenere abbastanza memoria, vengono restituiti gli attributi dell'allocazione che non ha avuto successo, e il valore restituito nella variabile memList possiede il bit 31 impostato a 1.

Argomenti della funzione

memList — Indirizzo della struttura MemList che intesta una serie di strutture MemEntry, ognuna delle quali indica un blocco di memoria da allocare.

Discussione

Ci sono sette funzioni dell'Exec che riguardano la gestione della memoria: AllocAbs, Allocate, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. In aggiunta ai punti discussi nel corso della spiegazione relativa alla funzione Allocate vi sono diversi aspetti da considerare.

Per prima cosa, ciascun task presente nel sistema possiede fino a sei aree di memoria diverse. La localizzazione di ciascuna di queste aree dev'essere definita aggiornando alcuni parametri della struttura Task prima di creare il task. Vediamo quali sono i parametri della struttura Task che definiscono queste sei aree di memoria:

tc_SPUpper — Puntatore al byte superiore dell'area occupata dallo stack del task, aumentato di 1.

tc_SPLower — Puntatore al byte inferiore dell'area occupata dallo stack del task.

tc_ExceptCode — Puntatore alla routine di exception associata al task.

tc_ExceptData — Puntatore all'area dati per la routine di exception del task.

tc_TrapCode — Puntatore alla routine di trap associata al task.

tc_TrapData — Puntatore all'area dati per la routine di trap del task.

tc_MemEntry — Puntatore a una struttura List che fornisce al sistema indicazioni sull'allocazione dinamica della memoria. I blocchi di memoria presenti in questa lista vengono automaticamente liberati dal sistema quando il task conclude la propria esecuzione.

Il sistema mantiene automaticamente una lista della memoria libera. Ogni volta che un task viene aggiunto al sistema, questa lista viene aggiornata. Il sistema non mantiene una lista della memoria allocata. Una volta che un blocco di RAM è stato allocato, il sistema non è in grado di rilevare quale task lo detiene.

Spesso un task crea uno o più sotto-task durante la sua esecuzione, utilizzando la funzione AddTask. Quando questo accade, devono essere allocate dinamicamente le necessarie aree di memoria. Si può usare la struttura MemList per ottenere un aiuto nella gestione di queste allocazioni di memoria. La struttura MemList consente di costruire una lista di tutte le aree di memoria che il task ha allocato dinamicamente.

Quando un task ha concluso la propria esecuzione, una chiamata alla funzione RemTask libera automaticamente tutta la memoria associata a esso. Contemporaneamente viene liberata anche la memoria associata a tutti i sotto-task. Tutti i blocchi di memoria che diventano liberi entrano a far parte della lista di memoria libera del sistema. Se l'esecuzione del task termina prematuramente, la struttura MemList permette comunque al sistema di liberare automaticamente tutta la memoria associata a quel task e a ogni suo sotto-task.

Le funzioni AllocEntry e FreeEntry permettono di allocare e liberare diversi blocchi di memoria con una singola chiamata. AllocEntry richiede che il task abbia allocato e definito una struttura MemList nella quale siano elencati, sotto forma di strutture MemEntry, gli attributi e le dimensioni di ogni blocco di memoria da allocare.

La struttura MemList

La struttura MemList è definita come segue:

```
struct MemList {
    struct Node ml_Node;
    UWORD ml_NumEntries;
    struct MemEntry ml_ME[1];
};
```

La struttura MemList possiede tre parametri. Il parametro ml_Node è la sotto-struttura Node che permette di organizzare diverse struttura MemList in una lista. Il parametro ln_Name contiene il nome dell'area di memoria rappresentato dalla struttura MemList. Il parametro ln_Pri della struttura Node determina la posizione di quest'area nelle liste di memoria libera. Il parametro ln_Type dovrebbe essere impostato a NT_MEMORY. Il parametro

ml_NumEntries deve contenere il numero di blocchi di memoria elencati nel vettore lm_ME; ogni blocco è rappresentato da una struttura MemEntry, e quindi il vettore ml_ME elenca strutture di tipo MemEntry. Si tenga quindi presente che lo spazio di memoria occupato dalla struttura MemList varia a seconda del numero di strutture MemEntry elencate nell'array ml_ME.

AllocEntry cerca di allocare tutti i blocchi di memoria rappresentati dalla struttura MemList che le è stata passata come argomento, ma a volte non vi riesce. In questo caso, invece di restituire l'indirizzo di una nuova struttura MemList, la funzione restituisce un valore che indica il tipo di memoria della quale è fallito il tentativo d'allocazione (come, per esempio, MEMF_CHIP|MEMF_PUBLIC). Inoltre il bit 31 viene impostato a 1. Il task può poi decidere le azioni da intraprendere di conseguenza.

Un esempio

Il seguente è un esempio in linguaggio Assembly sull'uso della funzione AllocEntry, che alloca cinque aree di memoria da 8, 16, 24, 32 e 40 byte, ognuna delle quali con particolari attributi.

```
MyMemList:
DS.B LN_SIZE                    ; riserva spazio per la struttura Node
DC.W 5                          ; numero dei blocchi da allocare
DC.L MEMF_CLEAR                 ; elemento numero Ø
DC.L 8
DC.L MEMF_PUBLIC                ; elemento numero 1
DC.L 16
DC.L MEMF_CHIP|MEMF_CLEAR       ; elemento numero 2;
DC.L 24
DC.L MEMF_FAST|MEMF_CLEAR       ; elemento numero 3
DC.L 32
DC.L MEMF_PUBLIC|MEMF_CLEAR ; elemento numero 4
DC.L 4Ø
start:
LEA MyMemList, AØ
CALLLIB _LVOAllocEntry, A6
BTST.L #31, DØ
BEQ.S success
```

La prima parte di questo esempio (fino all'etichetta start) costituisce una struttura MemList in RAM. La seconda parte pone, per prima cosa, l'indirizzo della struttura MemList nel registro A0, poi chiama la funzione AllocEntry. Il registro A6 contiene l'indirizzo base della libreria Exec, cioè quello della struttura ExecBase. Questo indirizzo viene messo nella variabile SysBase quando si apre la libreria Exec con una chiamata a OpenLibrary. La variabile _LVOAllocEntry è la notazione in Assembly per l'offset del vettore di salto al punto di entrata della funzione AllocEntry nella libreria Exec. L'istruzione successiva controlla lo stato del bit 31 del registro D0. Se è impostato a 1

significa che l'allocazione di memoria non ha avuto successo. Questo schema può essere facilmente tradotto in C.

AllocMem

Sintassi di chiamata della funzione

```
memBlock = AllocMem (byteSize, attributi)
DØ                  DØ        D1: Ø-31
```

Scopo della funzione

Questa funzione costituisce lo strumento di allocazione della memoria che viene maggiormente usato dal sistema e dai programmi applicativi. A differenza di Allocate, AllocMem non richiede di specificare una lista privata di memoria libera, in quanto accede direttamente alla lista di sistema MemList, e quindi i blocchi di memoria che è in grado di allocare possono risiedere in qualunque area dello spazio indirizzabile. Quando un task ha bisogno di allocare memoria dinamicamente, chiama AllocMem indicando la dimensione del blocco da allocare e i suoi attributi. Se AllocMem, nella lista di sistema della memoria libera, rileva la presenza di un blocco libero di dimensioni sufficienti e con gli attributi richiesti, restituisce l'indirizzo di quel blocco di memoria, il quale in seguito non comparirà più in alcuna lista di sistema. Da quel momento, il blocco di memoria è di proprietà del task e viene considerato allocato.

Se non ci sono blocchi liberi di memoria sufficientemente grandi e con gli attributi richiesti, o se l'ammontare di memoria richiesta non è coerente, la funzione restituisce il valore zero. Si ricordi che la misura minima di un blocco è di otto byte, e quindi le dimensioni dei blocchi di memoria effettivamente allocati sono sempre arrotondate per eccesso a multipli di otto. Anche se può capitare che venga allocata più memoria di quanto richiesto, non occorre tenerne conto quando poi si chiama FreeMem per liberarla, dal momento che anche questa funzione esegue l'arrotondamento per eccesso a multipli di otto.

Argomenti della funzione

byteSize — Dimensione in byte del blocco di memoria che si desidera allocare: il numero viene arrotondato in eccesso a un valore multiplo di otto.

attributi Ecco l'elenco degli attributi:

MEMF_PUBLIC Questo attributo indica al sistema che la memoria richiesta dev'essere usata per memorizzarvi task, message port, codici o dati di interrupt. Questo attributo dev'essere indicato quando si desidera assicurare la compatibilità con future versioni del sistema operativo. La memoria che si richiede con questo attributo non dev'essere scambiata, indirizzata in aree diverse o comunque resa non indirizzabile.

MEMF_CHIP Questo attributo indica al sistema che la memoria richiesta dev'essere allocata nella chip RAM, lo spazio d'indirizzamento al quale i chip dedicati dell'Amiga sono in grado di accedere tramite canali DMA. Si tratta della memoria compresa nel primo megabyte dello spazio indirizzabile (512K nelle macchine meno recenti). I chip dedicati dell'Amiga sono tre e sono realizzati in tecnologia VLSI: Agnus (il chip dedicato alle animazioni), Denise (il chip dedicato alla grafica) e Paula (il chip dedicato alle periferiche e al suono). Generalmente, questi chip devono accedere a dati presenti in memoria, come bitmap o forme d'onda campionate, ma i loro canali DMA possono accedere soltanto alla chip RAM. Ecco perché la memoria per questi dati dev'essere allocata nella chip RAM. Il DMA viene impiegato per accedere alle bitmap dello schermo, agli oggetti che vengono mossi dal Blitter, ai dati audio, ai buffer per dati grezzi di accesso al disco e così via.

MEMF_FAST Questo attributo indica al sistema che la memoria richiesta dev'essere allocata al di fuori della chip RAM. Ci sono diverse ragioni per cui la fast RAM è preferibile alla chip RAM. La prima è che per le applicazioni grafiche e musicali è opportuno che nella chip RAM ci sia sempre spazio di memoria sufficiente per i dati a cui accedono i chip dedicati. In secondo luogo, quando i chip dedicati accedono alla chip RAM, devono necessariamente appropriarsi di cicli sui bus indirizzi e dati per permettere ai loro canali DMA di accedere alla chip RAM, e questo causa un rallentamento del tempo di accesso della CPU a questo spazio d'indirizzamento. Se il rallentamento non è accettabile, occorre che il task e le relative aree dati risiedano nella fast RAM, lo spazio d'indirizzamento superiore al primo megabyte al quale i chip dedicati non possono accedere. Questo spazio d'indi-

rizzamento non è infatti condiviso con i canali DMA.

MEMF_CLEAR Questo attributo indica al sistema che la memoria richiesta dev'essere azzerata. Se non viene specificato questo attributo, non si può fare nessuna ipotesi sul contenuto della memoria allocata.

Discussione

Ci sono nove funzioni dell'Exec che riguardano la gestione della memoria nel sistema Amiga: AllocAbs, Allocate, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. Si vedano anche le spiegazioni relative a queste funzioni.

La lista di sistema della memoria libera

Il sistema riconosce e mantiene una sola lista di memoria libera, nella quale vengono elencati tutti i blocchi di memoria che risultano liberi. Questa lista viene mantenuta continuamente aggiornata a mano a mano che i task allocano e liberano memoria. Per quanto riguarda invece la memoria allocata, il sistema non mantiene una lista che elenchi i blocchi di memoria allocati. Una volta che un blocco di memoria viene allocato, il sistema non ha modo di sapere quale task lo ha sotto controllo. Si tenga presente che AllocMem non può essere chiamata dalle routine di interrupt.

La programmazione in un ambiente multitasking

Si possono usare due "stili" per programmare nel sistema multitasking dell'Amiga, eventualmente anche sovrapponendoli.

Il primo metodo utilizza soltanto le funzioni AllocMem e FreeMem. Con questo sistema, vengono impiegate AllocMem e FreeMem per allocare e liberare blocchi di memoria in modo globale. I blocchi che vengono allocati vengono automaticamente estratti dalla lista della memoria libera mantenuta dal sistema.

Il secondo metodo usa la funzione AllocEntry insieme alle funzioni Allocate e Deallocate. Con questo sistema, come prima operazione il task utilizza le funzioni AllocEntry oppure AllocMem (che funzionano con la lista di sistema della memoria libera) per allocare un'area di memoria sufficiente a soddisfare tutte le sue esigenze successive. In pratica, il task stima la quantità di cui avrà bisogno, pensando anche agli eventuali sotto-task, e provvede ad allocarla appena riceve il controllo, avendo così la sicurezza che tutte le sue esigenze di memoria saranno certamente soddisfatte nel corso dell'elaborazione. A questo punto, crea una propria lista della memoria libera per gestire l'area di memoria diventata di sua proprietà: per tutte le successive necessità di allocazione e

liberazione di memoria userà le funzioni Allocate e Deallocate, che funzionano con le liste private di memoria libera.

Questo secondo metodo richiede una previsione delle necessità di memoria del task. Se non è possibile farla, oppure se la memoria è troppo ristretta, conviene ricorrere al primo metodo.

AllocSignal

Sintassi di chiamata della funzione

```
signalNum = AllocSignal (signalNum)
DØ                       DØ
```

Scopo della funzione

Questa funzione permette ai task di allocare uno dei bit di segnale che hanno a disposizione per assegnarlo poi, per esempio, a una message port o a una routine di interrupt. Il task ha la facoltà di specificare un particolare bit di segnale tra i 16 che ha a disposizione, oppure, indicando come argomento il valore -1, di lasciare al sistema l'onere di decidere quale bit di segnale allocare. In entrambi i casi, la funzione restituisce il numero del bit di segnale se l'allocazione è riuscita, altrimenti restituisce il valore -1. Questa funzione cambia il parametro tc_AllocSignal della struttura Task del task.

Argomenti della funzione

signalNum Numero del bit di segnale che si desidera allocare (tra 0 e 31) oppure -1 se non si desidera indicare una preferenza.

Discussione

Ci sono sei funzioni dell'Exec che riguardano i segnali software nel sistema Amiga: AllocSignal, FreeSignal, SetExcept, SetSignal, Signal e Wait. Anche le funzioni GetMsg, PutMsg e ReplyMsg ricadono nella categoria delle funzioni di gestione dei segnali (i messaggi, infatti, possono generare segnali). Si vedano anche le spiegazioni relative a queste funzioni.

I task comunicano tra loro passandosi segnali attraverso le message port. Lo stesso tipo di comunicazione avviene anche con i dispositivi di I/O.

Le message port

Per scambiare messaggi tra task occorre che ogni task disponga almeno di una message port e di una reply port (possono anche essere la stessa message port), in modo da poter ricevere messaggi da altri task e le risposte ai messaggi che ha inviato. Si può predisporre più di una message port per ogni task. Basta impostare una struttura MsgPort per ciascuna message port da associare al task. Se si desidera che la message port sia pubblica, occorre dotarla di nome e dichiararla tale tramite la funzione AddPort. In pratica si possono impostare diverse strutture MsgPort, tante quante sono necessarie per gestire le necessità di scambio di messaggi del task che si sta definendo. Il solo limite con cui ci si può scontrare è l'eventuale carenza di memoria. Le message port che si associano a un task possono essere utilizzate per qualsiasi tipo di comunicazione con altri task, dispositivi di sistema, chip dedicati, dispositivi esterni e così via.

Le code alle message port

Tutti i messaggi che arrivano alla message port di un task vengono messi nella relativa coda. Ciascun messaggio in arrivo costituisce un evento. Quando arriva un messaggio (se la message port è stata opportunamente predisposta), il task indicato nella struttura MsgPort riceve un segnale. Il bit di segnale della struttura Task che all'arrivo del messaggio viene impostato a 1 è quello che era stato allocato e assegnato a quella message port.

La variabile tc_SigAlloc

I bit di segnale vengono allocati dinamicamente dalla funzione AllocSignal usando la variabile tc_SigAlloc della struttura Task. Questa variabile dovrebbe essere impostata a zero prima che d'inserire il task nel sistema. È una variabile a 32 bit nella quale ogni bit identifica lo stato di allocato o libero di un segnale, per un totale di 32 segnali numerati da 0 a 31. I 16 bit inferiori sono riservati al sistema (i bit numerati da 0 a 15); i bit disponibili per il task sono i 16 superiori (i bit numerati da 16 a 31).

Come tenere nota dei segnali

Le cose possono complicarsi quando un task dispone di diverse message port, a ognuna delle quali ha assegnato un bit di segnale. In questi casi il task, tramite la funzione Wait, entra generalmente in attesa di tutti i segnali che ha allocato. Quando una message port riceve un messaggio, la funzione Wait

restituisce il controllo al task passandogli il numero del bit corrispondente al segnale assegnato alla message port. Grazie a questo numero, il task deduce in quale message port è giunto il segnale. Supponendo che il controllo inizi da una message port individuata da un puntatore di nome msgPort, l'istruzione che segue permette al task, dopo l'esecuzione della funzione Wait, di rilevare se il segnale è stato causato da un messaggio giunto in quella particolare message port.

```
if (signalNum & (1 << msgPort->mp_SigBit)) { ... }
```

Le istruzioni fra parentesi graffe vengono eseguite solo se il messaggio è stato ricevuto dalla message port individuata dal puntatore msgPort.

Attenzione: i segnali non possono essere allocati o liberati da codici di gestione delle exception. Non esiste la possibilità di scambi di messaggi tra diverse sezioni di codici delle exception e/o altri task nel sistema.

AllocTrap

Sintassi di chiamata della funzione

```
trapNum = AllocTrap (trapNum)
D0                  D0
```

Scopo della funzione

Questa funzione alloca un numero di trap fra quelli che risultano disponibili per il task, e lo restituisce. Il valore restituito è in pratica il numero del bit di trap che la funzione ha allocato. Questi numeri di trap sono associati con le istruzioni TRAP del 68000. La funzione AllocTrap alloca il particolare numero di trap indicato dal task, oppure il primo numero di trap libero se il task non indica preferenze. Se la trap richiesta è già in uso, o non sono disponibili trap libere, la funzione restituisce il valore -1.

Argomenti della funzione

trapNum	Numero di trap desiderato (compreso fra 0 e 15), oppure -1 se non si vogliono indicare preferenze.

Discussione

Ci sono due funzioni dell'Exec che hanno esplicitamente a che fare con la gestione delle trap: AllocTrap e FreeTrap. AllocTrap alloca un bit di trap e FreeTrap lo libera. Il sistema mantiene automaticamente una lista dei numeri di trap disponibili per essere allocati dai task.

Il sistema multitasking dell'Amiga permette a ciascun task di prendere pieno controllo del sistema hardware, rendendo virtualmente ciascun task un terminale della macchina. Questa capacità si estende anche alle trap del 68000 e a tutti gli altri tipi di condizioni di exception. Si noti tuttavia che le trap sono distinte dalle exception, nella terminologia impiegata per descrivere la libreria Exec, terminologia che non concorda con quella del 68000 stesso così com'è definita nei manuali della Motorola. Per esempio le trap del 68000 sono separate in trap hardware e software. Le trap hardware hanno numeri di vettore da 2 a 15, e comprendono quelle causate da errori d'indirizzamento, da divisioni per zero, da istruzioni illegali e da altre undici condizioni d'errore. I numeri delle trap riconosciute dal 68000 vanno da 0 a 255.

Tutti i vettori di trap dell'Amiga sono invece tradotti esclusivamente in trap software del 68000, e quando si verificano causano l'esecuzione di particolari routine dell'Exec. Queste rilevano qual è il task in esecuzione e inseriscono il numero di trap e lo stato della CPU nel supervisor stack prima di cedere il controllo alla routine di gestione delle trap del task. Si noti che questa procedura non prevede nessun'altra preparazione prima di cedere il controllo alla routine di trap del task. In particolare, il sistema non si preoccupa di salvare lo stato dei registri della CPU, ma soltanto quello del registro di stato. Si tenga presente che il valore del numero di trap che la routine di trap può ottenere accedendo allo stack del task può variare fra 32 e 47, corrispondente alle istruzioni che vanno da TRAP #0 a TRAP #15.

Una trap è sempre causata dai codici del task che la CPU sta eseguendo. Può avere un carattere involontario, o non premeditato, come nel caso di una divisione per zero; oppure può essere causata esplicitamente dall'esecuzione di un'istruzione TRAP #N. È anche importante notare che il sistema gestisce le elaborazioni delle trap nel supervisor mode della CPU. Quindi, quando si verifica una trap, lo scambio di controllo fra i task viene momentaneamente sospeso finché non è stata completamente elaborata.

Quando il task alloca un bit di trap, dev'essere sicuro che la sua routine di trap sia pronta, in quanto l'Exec non esegue controlli in merito quando le cede il controllo. La routine di trap di un task, dev'essere in grado di rilevare quale trap si è verificata, per svolgere le appropriate operazioni. Per farlo confronta il numero di trap memorizzato nello stack con i numeri di trap allocati dal task durante la cui esecuzione si è verificata la trap. Ciascun task ha il proprio personale insieme di numeri di trap allocati, collegati a una stessa routine di trap che li gestisce. Si può osservare la variabile tc_TrapAlloc del task, nella sua struttura Task, per vedere quali numeri di trap sono allocati per un particolare task.

Si possono utilizzare quattro variabili della struttura Task per specificare la routine di gestione delle trap associata al task.

tc_TrapCode	Deve contenere l'indirizzo della routine di trap che elaborerà la trap. Ciascun task ha una sola routine di trap. Tuttavia si può progettare questa routine perché si comporti in modo diverso a seconda del numero di trap. Si veda anche la funzione FreeTrap.
tc_TrapData	È l'indirizzo dell'area dati per i dati richiesti dalla routine di trap.
tc_TrapAlloc	È la maschera dei bit di trap allocati dal task. Tale variabile è opzionale; l'Exec la imposta a zero quando il task viene avviato.
tc_TrapAble	Questo parametro descrive l'insieme delle trap abilitate per il task.

Una volta che la routine di trap è stata progettata, devono essere impostate le variabili tc_TrapCode e tc_TrapData perché puntino rispettivamente al codice della trap e all'area dati. Ciò può essere svolto come segue:

```
VOID MyTrapRoutine();
MyTask.tc_TrapCode = &MyTrapRoutine;
/* imposta un puntatore all'indirizzo della routine di trap che gestirà
   l'elaborazione delle trap */
MyTask.tc_TrapData = &MyTrapData;
/* qui si assume che sia stato allocato e inizializzato un blocco di memoria
   denominato MyTrapData. Questo blocco di memoria sarà usato come area dati
   per la routine di trap */
```

Per allocare un numero di trap si possono impartire le seguenti istruzioni:

```
mytrap = AllocTrap (-1);
if (mytrap == -1)
{
     printf ("Questo task non ha altre trap allocabili");
     cleanup();
}
```

Oppure si può selezionare una specifica trap in questo modo:

```
mytrap = AllocTrap (3);
if (mytrap == -1)
{
     printf ("La trap 3 è già stata allocata");
     cleanup();
}
```

Le istruzioni di trap del 68000 includono CHK, TRAP e TRAPV. TRAP è l'istruzione esplicita di trap software e TRAPV è l'istruzione di trap in overflow. Si noti che tutte e tre le istruzioni possono essere eseguite dal 68000 quando funziona nel modo user. Tuttavia la routine di trap riceve il controllo solo quando la CPU è entrata nel modo supervisor. Le istruzioni di trap consentono a qualsiasi programma di causare un'exception. D'altro canto, in genere le exception sono causate da condizioni di natura hardware.

Quando il 68000 inizia l'elaborazione di un'exception, viene generato un numero di vettore in riferimento al vettore di exception indicato come argomento dell'istruzione TRAP (specificato dai quattro bit di ordine inferiore nel formato in memoria dell'istruzione TRAP). Con il 68000 sono perciò disponibili 16 vettori di trap corrispondenti a 16 istruzioni TRAP (da TRAP #0 a TRAP #15). Si possono usare questi vettori di trap per saltare a brevi routine di elaborazione delle trap. Queste routine possono essere progettate per scopi specifici legati alle necessità di elaborazione dei programmi.

Attenzione: i numeri di trap non possono essere allocati o liberati, una volta che siano all'interno del codice di gestione di un'exception. Inoltre, non esiste capacità di scambiare messaggi tra diverse sezioni di codici delle exception e/o altri task nel sistema.

AttemptSemaphore

Sintassi di chiamata della funzione

```
success = AttemptSemaphore (signalSemaphore)
DØ                         AØ
```

Scopo della funzione

Questa funzione tenta di ottenere (avere accesso a) un semaforo, cioè di bloccare un semaforo. AttemptSemaphore è molto simile a ObtainSemaphore con la differenza che non sospende l'esecuzione del task se non è possibile ottenere e bloccare il semaforo (in genere perché è detenuto da un altro task). La funzione restituisce il valore FALSE se qualche altro task detiene il semaforo.

Argomenti della funzione

signalSemaphore Indirizzo di una struttura SignalSemaphore opportunamente inizializzata.

Discussione

Ci sono sette funzioni dell'Exec che hanno a che fare con i semafori di segnalazione e due funzioni che riguardano la lista dei semafori di segnalazione. Queste funzioni sono illustrate brevemente nel corso della spiegazione di AddSemaphore.

I task interagiscono con i semafori impossessandosene (AttemptSemaphore e ObtainSemaphore) e liberandoli (ReleaseSemaphore). Un task può anche ottenere e bloccare un semaforo diverse volte prima di liberarlo. Ogni volta viene incrementato il parametro ss_NestCount della struttura SignalSemaphore; al contrario, ogni volta che il task libera il semaforo, il relativo parametro ss_NestCount viene decrementato.

Quando il parametro ss_NestCount contiene un valore diverso da zero, la struttura SignalSemaphore è bloccata, e uno specifico task ha il controllo esclusivo del semaforo. In tal caso, gli altri task non possono ottenere quel semaforo e se chiamano ObtainSemaphore entrano in attesa. Anche con AttemptSemaphore il task non riesce a ottenere il semaforo, ma in quel caso non viene messo in attesa e può proseguire la propria esecuzione.

AvailMem

Sintassi di chiamata della funzione

```
size = AvailMem (attributi)
D0               D1
```

Scopo della funzione

Questa funzione restituisce la misura della quantità totale di memoria libera conforme agli attributi indicati. Se negli attributi viene indicato MEMF_LARGEST, la funzione restituisce la misura del più grande blocco di memoria disponibile nel sistema e conforme agli attributi indicati.

Argomenti della funzione

attributi Maschera degli attributi che indicano alla funzione su quale tipo di RAM deve eseguire il controllo della memoria libera. Oltre agli attributi descritti nella spiegazione della funzione AllocMem, AvailMem permette d'indicare anche l'attributo MEMF_LARGEST, il quale le indica di restituire non la dimensione della quantità totale di memoria libera conforme agli attributi, ma la dimensione del più grande blocco (o chunk) di memoria libera continua e conforme agli attributi.

Discussione

Ci sono nove funzioni dell'Exec che riguardano la gestione della memoria: Allocate, AllocAbs, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. Si vedano anche le spiegazioni relative a queste funzioni.

Le allocazioni e i rilasci di memoria effettuati dai task comportano continue modifiche nella lista di sistema della memoria libera. A rendere le cose ancor più complicate, nell'Amiga la memoria può essere identificata per mezzo di diversi attributi (MEMF_CHIP, MEMF_FAST, MEMF_PUBLIC). Quindi, in un particolare momento il quantitativo di memoria disponibile può non essere sufficiente per soddisfare le richieste di un task. Questo è il motivo per cui è utile la funzione AvailMem.

Si può chiamare AvailMem prima di chiamare le altre funzioni di allocazione della memoria, al fine di verificare se esiste abbastanza memoria per soddisfare le allocazioni di memoria necessarie. Si noti però che la dimensione restituita da AvailMem costituisce la memoria totale libera conforme agli attributi indicati, ma questa memoria è generalmente dispersa sull'intero spazio indirizzabile. Anzi, è improbabile che occupi un'area continua dello spazio di memoria indirizzabile. Se si ha bisogno di allocare un unico blocco particolarmente esteso, per esempio un blocco di chip RAM da utilizzare come bitplane di una bitmap, e si desidera sapere se esiste nel sistema un blocco continuo di chip RAM di quelle dimensioni, occorre chiamare la funzione AvailMem indicando gli attributi MEMF_CHIP e MEMF_LARGEST.

Si tenga presente che in un sistema multitasking come l'Amiga la memoria viene continuamente allocata e liberata dai task che vanno in esecuzione da quando la macchina termina la procedura d'attivazione (dopo l'accensione o il reset). Questa continua evoluzione porta a un vero e proprio stato di disordine per quanto riguarda la memoria, cioè a una frammentazione sempre crescente con il progredire delle attività dei task. L'eccessiva frammentazione è estremamente dannosa, e porta a situazioni paradossali, nelle quali può esserci moltissima memoria libera, ma così frammentata in blocchi da pochi byte da

essere quasi inservibile. Può cioè accadere che chiamando la funzione AvailMem senza indicare l'attributo MEMF_LARGEST, il risultato sia più che tranquillizzante, ma che a un più attento esame, cioè richiamando la funzione indicando anche MEMF_LARGEST, la situazione si riveli disastrosa.

A questo malessere non c'è rimedio, in quanto i blocchi di memoria allocati non sono rilocabili. Le dimensioni dei più grandi blocchi continui per ogni tipo di RAM sono ottimi indicatori del grado di disordine raggiunto dal sistema. Si può solo cercare di limitare la frammentazione creando task che si approprino subito di tutta la memoria che prevedono d'impiegare, e che gestiscano questa memoria con una lista privata di memoria libera tramite le funzioni Allocate e Deallocate. In questo modo, al sistema risulta che il task ha allocato un unico grande blocco di memoria, e la frammentazione dovuta alle allocazioni e ai rilasci di memoria da parte del task si restringe al suo spazio privato di memoria libera. Quando il task termina di esistere, un grande blocco continuo di memoria torna a far parte della memoria libera di sistema al posto di tanti piccoli blocchi sparsi qua e là. A questo proposito si vedano le spiegazioni delle funzioni Allocate e Deallocate.

Da queste considerazioni, sorge spontanea una domanda: come viene occupata la memoria RAM dell'Amiga nei diversi momenti e nelle diverse situazioni in cui si trova la macchina? Per rispondere occorre considerare le varie strutture che intervengono durante l'esecuzione di un task.

Strutture in memoria per ciascun task

Per prima cosa esiste una struttura Task per ogni task, la quale occupa una certa quantità di memoria. A questa si aggiunge la memoria occupata dai vari elementi costitutivi di un task. Per ogni task viene allocata memoria per:

- lo stack del task, detto anche user stack per distinguerlo dal supervisor stack, lo stack di sistema.
- La routine di exception del task.
- L'area dati per la routine di exception.
- La routine di trap del task.
- L'area dati per la routine di trap.
- Le aree per mantenere le informazioni sui segnali, le informazioni di controllo della memoria allocata e così via.
- I codici (ed eventualmente i dati) dello stesso task.

Secondo, viene allocata memoria per le librerie che il task apre, soprattutto quando non sono disponibili e vengono quindi caricate in memoria dal disco. La memoria per le librerie è allocata dalla funzione OpenLibrary e viene

occupata dalle voci che seguono:

- le istruzioni di salto alle funzioni della libreria, cioè la tavola dei vettori di salto della libreria.
- La struttura Library di gestione della libreria.
- Il segmento dati della libreria.
- Le funzioni vere e proprie della libreria (per essere precisi, si noti che le funzioni non fanno formalmente parte della libreria, o quanto meno possono risiedere in qualunque area della memoria, anche in ROM).

Terzo, quando il task interagisce con i dispositivi provvede ad allocare tutte le strutture IORequest di cui può aver bisogno. Queste strutture vengono parzialmente inizializzate dalla funzione OpenDevice. Nell'interazione con i dispositivi viene inoltre allocata memoria per i buffer nei quali transitano i dati scambiati con i task. Questi buffer appartengono in parte ai dispositivi e in parte sono sotto il controllo del task.

Quarto, c'è la memoria allocata per tutte le message port di cui il task può aver bisogno nelle comunicazioni con i dispositivi e con gli altri task. Per esempio, se un task lavora con dieci dispositivi, ci possono essere dieci o più message port a esso associate (con il termine "message port" ci riferiamo anche alle eventuali reply port del task). Inoltre, viene allocata memoria per tutti i messaggi che un task scambia con il mondo esterno.

Quinto, c'è la memoria allocata alle strutture di gestione della memoria, necessarie al controllo delle aree di memoria che il task ha allocato.

Sesto, c'è la memoria allocata per tutte le svariate ed eterogenee funzioni dell'Exec e della libreria Graphics.

Infine, viene allocata memoria per le informazioni grafiche che il task decide di visualizzare e per le forme d'onda che decide di far riprodurre. La memoria per le informazioni grafiche, in particolare, può essere particolarmente ampia. Si supponga di avere un display in modo dual-playfield, con un totale di sei bitplane per definirne la presentazione video, con ogni bitplane che occupa 10240 byte di memoria (in standard PAL). Si aggiunga la memoria per gli sprite che agiranno sull'area video. Diventa evidente che la memoria per la grafica costituisce il blocco più rilevante impiegato dal task.

Cause

Sintassi di chiamata della funzione

Cause (interrupt)
A1

Scopo della funzione

Questa funzione provoca un interrupt software. Se viene chiamata quando la CPU è nel modo user, l'interrupt software sospende l'esecuzione del task in favore della relativa routine di gestione.

Attualmente sono disponibili solo cinque priorità di interrupt software: -32, -16, 0, +16 e +32. La priorità di esecuzione di una routine di interrupt è contenuta nel parametro ln_Pri della sotto-struttura Node della relativa struttura Interrupt.

Argomenti della funzione

interrupt	Indirizzo di una struttura Interrupt opportunamente inizializzata.

Discussione

Nella maggior parte dei casi, uno scambio di controllo fra i task si verifica a causa di un interrupt hardware o software (per esempio una trap software). Durante l'interrupt (ma dopo l'esecuzione della relativa routine di gestione) il rescheduler del sistema, ovvero un algoritmo che ha il compito di decidere a quale nuovo task cedere il controllo in un ambiente multitasking, esamina la priorità del task che è stato sospeso. A volte, appena una routine di interrupt ha terminato l'elaborazione, viene avviato nel sistema un nuovo task la cui priorità supera quella del task che era stato sospeso. Il task a più alta priorità può essere stato appena aggiunto con la funzione AddTask o reso attivo dopo il verificarsi un evento (un messaggio è giunto nella sua message port). Il task che ha perso il controllo entra in attesa del suo prossimo turno. Questo è il meccanismo principale mediante il quale un interrupt produce uno scambio di controllo fra i task.

Gli stack dei task, i cosiddetti user stack, non intervengono nella gestione degli interrupt. È infatti sul supervisor stack, lo stack di sistema, che grava tutta

la gestione degli interrupt (ovviamente in termini di parametri e configurazioni da salvare e ripristinare). Questa scelta è dettata dal fatto che gli interrupt possono anche nidificarsi, cioè può accadere che durante l'elaborazione di un interrupt se ne verifichi un altro a più alta priorità che sospende l'elaborazione del primo per essere elaborato a sua volta. Questo annidamento comporta ingenti salvataggi di dati nello stack, e lo stack del task potrebbe non essere sufficiente.

La sola eccezione a questa regola è la situazione che si verifica quando il task chiama la funzione SuperState, la quale forza la CPU a entrare nel modo supervisor, ma fa in modo che il task possa continuare a impiegare il proprio user stack. In questa condizione, gli eventuali interrupt nidificati sono costretti a servirsi dello stack del task. Quando si usa questa funzione è quindi necessario essere sicuri che il proprio user stack sia abbastanza grande da sostenere il carico di gestione degli interrupt nidificati.

La Tavola 1.3 fornisce una lista dei 15 possibili livelli di interrupt previsti

Livello di priorità del 68000	**Priorità dell'Amiga**	**Interrupt dell'Amiga**
6	14	Coprocessore Copper, interrupt ad alta priorità
	13	Porta d'espansione e interrupt del CIA 8250B
5	12	Disk Sync Byte restituito dal DMA
	11	Buffer di ricezione del dispositivo Serial pieno
4	10	Canale audio 1
	9	Canale audio 3
	8	Canale audio 0
	7	Canale audio 2
3	6	Blitter DMA done (il Blitter ha completato un compito)
	4	Coprocessore Copper, interrupt a bassa priorità
	5	Interrupt di vertical-blanking
2	3	Porte di I/O, tastiera, timer; interrupt hardware esterno, livello 2
1	2	Buffer di trasmissione del dispositivo Serial vuoto
	1	Blocco di dati del disco trasferito
	0	Interrupt software

Tavola 1.3: *Gli interrupt dell'Amiga trasformati nei livelli di priorità del 68000*

per l'Amiga e mostra come questi interrupt vengono tradotti in sei dei sette livelli di priorità riconosciuti e utilizzati dal 68000. Si noti che la routine Cause è in grado di causare solo interrupt software, che hanno il livello di priorità 1 della CPU, corrispondente al livello di priorità 0 dell'Amiga, il più basso in assoluto.

CheckIO

Sintassi di chiamata della funzione

```
result = CheckIO (iORequest)
DØ               A1
```

Scopo della funzione

Questa funzione rileva se una richiesta di I/O inoltrata dal task a un dispositivo è stata restituita, cioè elaborata. Essa restituisce il valore zero se la richiesta di I/O non è ancora stata restituita, permettendo un controllo asincrono (WaitIO, invece, consente un controllo sincrono). Tramite CheckIO i task possono periodicamente verificare se la richiesta di I/O è stata soddisfatta e nel frattempo svolgere altre operazioni.

Se CheckIO rileva che la richiesta di I/O ha il flag IOF_QUICK impostato (il task ha richiesto il QuickIO), restituisce subito il controllo indicando che la richiesta è stata elaborata (in questo caso, però, non si trova nella reply port del task dal momento che il QuickIO ha avuto successo). Si noti che quando la funzione CheckIO rileva che la richiesta di I/O indicata è stata elaborata, non la rimuove dalla reply port del task. A questo scopo il task deve usare la funzione Remove. La funzione CheckIO non dovrebbe comunque essere utilizzata per creare un ciclo di attesa mentre un'operazione di I/O viene portata a termine, dal momento che WaitIO è in questo caso molto più efficiente.

Si noti infine che quando la richiesta di I/O risulta elaborata, CheckIO ne restituisce l'indirizzo, lo stesso indicato dal task nella chiamata della funzione.

Argomenti della funzione

iORequest	Indirizzo della struttura IORequest che costituisce la richiesta di I/O inoltrata al dispositivo.

Discussione

Ci sono otto funzioni dell'Exec che riguardano direttamente i dispositivi di I/O: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Si vedano anche le spiegazioni relative a queste funzioni.

Quando un task emette una richiesta di I/O, si trova di fronte a due scelte: può entrare in attesa che l'I/O sia completato (I/O sincrono) oppure può dedicarsi ad altre attività mentre la richiesta di I/O rimane in coda alla message port dell'unità del dispositivo indirizzata (I/O asincrono). Poniamoci la domanda: perché questo task sta chiedendo un'operazione di I/O e cosa intende farne? Se sta inviando dati a un dispositivo può essere sufficiente porre questi dati in un buffer di memoria e proseguire con le altre attività; in questo caso è comodo inviare la richiesta in modo asincrono. Se invece il task vuole ricevere dati da un dispositivo, può accadere che questi dati gli siano assolutamente necessari per proseguire nelle sue operazioni; il task deve quindi inviare la richiesta di I/O in modo sincrono.

Per ottenere maggiori dettagli sulla gestione dei dispositivi di I/O dell'Amiga e in particolare sulla funzione CheckIO, si consulti il volume II.

CloseDevice

Sintassi di chiamata della funzione

CloseDevice (iORequest)
A1

Scopo della funzione

Questa funzione informa il sistema che il task ha concluso l'accesso all'unità di un dispositivo aperta precedentemente con la funzione OpenDevice. Il task che ha effettuato la chiamata a CloseDevice non può più accedere al dispositivo a meno che non lo riapra con OpenDevice. CloseDevice chiama anche la routine CLOSE del dispositivo, la quale effettua le necessarie procedure di chiusura dell'unità. Se il task non possiede altre unità del dispositivo aperte, può liberare la memoria occupata dalla struttura IORequest che aveva allocato per chiamare OpenDevice.

Argomenti della funzione

iORequest Indirizzo della struttura IORequest inizializzata dalla funzione OpenDevice.

Discussione

Ci sono otto funzioni dell'Exec preposte alla gestione dei dispositivi di I/O del sistema Amiga: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Si vedano anche le spiegazioni relative a queste funzioni.

Quando un task apre l'unità di un dispositivo tramite la funzione OpenDevice, vengono inizializzati due parametri della struttura di I/O indicata come argomento. Il parametro io_Device viene inizializzato con l'indirizzo base della libreria del dispositivo in memoria, cioè della struttura Device di gestione del dispositivo. Questo indirizzo viene impiegato dalle funzioni DoIO e SendIO per sapere a quale dispositivo è diretta la richiesta di I/O. Oltre a questo parametro, che è di vitale importanza per poter comunicare con il dispositivo, talvolta la funzione OpenDevice inizializza anche il parametro io_Unit (per ulteriori informazioni su questo parametro si consulti *Programmare l'Amiga Volume II*). Una volta che OpenDevice ha restituito il controllo, il task può modificare gli altri parametri per inviare richieste di I/O all'unità del dispositivo che ha aperto. Quando non ha più bisogno del dispositivo, o magari desidera chiudere un'unità del dispositivo fra quelle che ha aperto, deve chiamare CloseDevice. Questa funzione chiude l'unità aperta, e provvede a rendere inutilizzabile la struttura di I/O inserendo valori negativi nei parametri io_Device e io_Unit.

La chiusura anche totale di un dispositivo non-residente non comporta la sua rimozione dal sistema: il dispositivo rimane sempre disponibile in memoria, fino a quando un task non ne richiede l'esplicita rimozione tramite la funzione RemDevice. Se la funzione RemDevice viene chiamata quando il dispositivo risulta ancora aperto da qualche task nel sistema, non lo rimuove subito, ma predispone il sistema a chiuderlo non appena risulterà completamente chiuso. Quando un dispositivo non-residente viene rimosso, la memoria che occupa viene completamente liberata. Alla successiva chiamata di OpenDevice per aprire quel dispositivo, il sistema provvede a ricaricarlo da disco.

CloseLibrary

Sintassi di chiamata della funzione

CloseLibrary (library)
A1

Scopo della funzione

Questa funzione informa il sistema che il task non intende più impiegare una libreria aperta con OpenLibrary. Il task che effettua la chiamata a CloseLibrary non può più accedere alla libreria a meno che non la riapra con una chiamata a OpenLibrary.

Argomenti della funzione

library	Puntatore a una struttura Library.

Discussione

Ci sono otto funzioni dell'Exec che riguardano le libreria nel sistema Amiga: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Si vedano anche le spiegazioni relative a queste funzioni.

L'organizzazione a librerie del sistema operativo dell'Amiga permette di raggruppare le funzioni più utili in moduli separati e indipendenti. Le librerie di sistema sono in parte su ROM (quelle che vengono usate più di frequente, come le librerie Exec e Intuition), e in parte su disco (come le librerie Version e Icon). Oltre alle librerie di sistema, i programmatori possono aggiungerne altre per raccogliere nuove funzioni non disponibili a livello di sistema e abbastanza standardizzate per essere utilizzabili da programmi diversi. Le librerie presentano parecchi vantaggi sia per il programmatore che per il sistema. Permettono di creare programmi più brevi e spingono i programmatori a mantenere una certa uniformità nei loro programmi. Inoltre, le librerie non-residenti consentono un uso efficiente della memoria, in quanto vengono caricate da disco solo dietro esplicita richiesta, e in situazioni di carenza di memoria possono anche essere rimosse dalla memoria, esattamente come i dispositivi di I/O (che in realtà sono delle particolari librerie).

Le librerie vengono identificate attraverso i loro nomi e (opzionalmente) attraverso i loro numeri di versione. Il numero di versione serve per distinguere una libreria dalle versioni precedenti: alcuni programmi potrebbero richiedere che una particolare libreria abbia un numero di versione non inferiore a un certo valore per garantire la compatibilità software.

Il programmatore che desidera sfruttare a fondo le capacità dell'Amiga può crearsi una collezione di librerie su disco, che i suoi programmi possono richiamare in memoria in qualsiasi momento tramite la funzione OpenLibrary.

ColdReset

Sintassi di chiamata della funzione

ColdReset ()

Scopo della funzione

Questa funzione causa il reset della macchina, anche dal punto di vista hardware, cioè avvia la stessa sequenza che si verifica all'accensione del sistema. Tutte le attività del sistema vengono arrestate. L'intero software sistema viene reinizializzato e nulla viene preservato. Questa funzione causa anche un reset hardware del 68000 per riportare allo stato iniziale tutti i dispositivi hardware.

Argomenti della funzione

Questa funzione non prevede argomenti.

Discussione

ColdReset opera soltanto nel modo supervisor del 68000. Se viene chiamata quando la CPU è nel modo user, si provoca l'exception relativa alla violazione di privilegio nell'esecuzione delle istruzioni.

L'operazione di reset fa entrare il 68000 nel più alto livello di exception. Ci sono due modi per generare il segnale di reset: si può premere il tasto Ctrl contemporaneamente ai due tasti Amiga, oppure usare in un programma l'istruzione RESET del 68000. Due vettori di reset sono disposti al punto più alto

della tavola dei vettori di exception del 68000. Questa tavola contiene un totale di 255 vettori di exception. Il primo vettore di reset richiede quattro word, a differenza degli altri vettori di exception che richiedono soltanto due word, e si trova nello spazio di programma del modo supervisor.

Il segnale di reset è progettato per riportare l'hardware del sistema a uno stato di default. Molto spesso la necessità d'impartire il reset nasce da un funzionamento catastrofico della macchina dovuto a un task che presenta un malfunzionamento, ma le cause possono essere anche altre. Per esempio, se si verifica un errore sul bus durante l'elaborazione di un'exception causata da un precedente errore sul bus, o da un errore d'indirizzamento o da un errore di accesso in lettura alla memoria, il 68000 si blocca e cessa ogni elaborazione. Quando questo si verifica, il 68000 rimuove se stesso dal sistema, preservando in tal modo tutti i contenuti della memoria. In tali circostanze, soltanto un reset può riavviare il processore.

Quando il processore viene sottoposto al reset, qualsiasi processo in corso in quell'istante viene abbandonato e non può essere recuperato. Il 68000 viene forzato a funzionare nel modo supervisor, e l'esecuzione delle istruzioni viene bloccata. La maschera di interrupt del 68000 viene impostata al livello 7, il più alto possibile. Ciò significa che nessun altro interrupt può prendere il controllo della macchina dopo che è stato avviato il reset.

Il numero del vettore di reset viene poi generato internamente per individuare il vettore corrispondente all'exception di reset della locazione 0 nello spazio programma del modo supervisor. Non si può fare alcun affidamento sulla validità dei contenuti dei registri in situazioni del genere. In particolare è sconosciuto l'esatto contenuto, bit per bit, dello stack pointer del supervisor stack. Per questa ragione, né il program counter né il registro di stato vengono salvati dopo che si è verificato un reset.

In seguito, l'indirizzo contenuto nelle prime due word del vettore di exception relativo al reset viene preso come valore iniziale per lo stack pointer del supervisor stack, mentre l'indirizzo contenuto nelle ultime due word del vettore di exception del reset viene preso come valore iniziale per il program counter. L'esecuzione riprende quindi dal nuovo indirizzo assunto dal program counter, indirizzo che dovrebbe essere quello della procedura di reset dell'Amiga.

Si noti che l'istruzione RESET (eseguibile soltanto nel modo supervisor, essendo un'istruzione privilegiata), a differenza del reset hardware non causa il caricamento del vettore di reset ma pone invece al livello basso la linea di reset del 68000 per riportare nella loro condizione iniziale tutti i dispositivi hardware esterni (esterni al 68000 ovviamente). Questo consente alle istruzioni software di un task di reimpostare il sistema hardware a uno stato conosciuto e poi continuare l'elaborazione con l'istruzione seguente, senza avere in effetti avviato la procedura di reset della macchina.

CopyMem

Sintassi di chiamata della funzione

CopyMem (srcPointer, destPointer, size)
A0 A1 D0

Scopo della funzione

Questa funzione copia un blocco di memoria da una locazione RAM a un'altra, eseguendo l'operazione in modo molto veloce ed efficiente. CopyMem è in grado di spostare blocchi di qualsiasi dimensione da qualsiasi locazione a qualsiasi altra; ottimizza le operazioni di copia di grandi aree di memoria tramite copie parziali di sotto-aree. Quando deve effettuare la copia di aree di dimensioni ridotte, o quando gli indirizzi dei blocchi di memoria sono disallineati (cioè non multipli di long word), utilizza un modo di copia a byte.

Argomenti della funzione

srcPointer Indirizzo dell'area sorgente di memoria che dev'essere copiata.

destPointer Indirizzo della locazione di memoria che individua l'inizio dell'area nella quale dev'essere trasferito il contenuto dell'area sorgente.

size Dimensione in byte dell'area da copiare.

Discussione

Ci sono due funzioni dell'Exec che permettono di copiare blocchi di memoria: CopyMem e CopyMemQuick. Quando si deve copiare una sezione di memoria da una posizione all'altra dello spazio indirizzabile, con i due puntatori eventualmente non allineati alle long word (cioè non divisibili per quattro) o con una quantità di memoria da spostare che non è un multiplo di quattro, occorre impiegare la funzione CopyMem.

Se invece i puntatori sono allineati alle long word e la dimensione dell'area di memoria da spostare è un multiplo di quattro, è preferibile impiegare

CopyMemQuick, che è più efficiente.

Si tenga infine presente che CopyMem e CopyMemQuick sono funzioni di copia, non funzioni di movimento. Quando queste funzioni restituiscono il controllo, ci sono due copie dei dati in due diverse aree RAM. Le due aree, inoltre, non devono sovrapporsi.

CopyMemQuick

Sintassi di chiamata della funzione

```
CopyMemQuick (srcPointer, destPointer, size)
              A0          A1           D0
```

Scopo della funzione

Questa funzione permette di effettuare la copia di un'area di memoria in modo ancor più veloce che con CopyMem. Questo è però possibile grazie ad alcune restrizioni che riguardano gli indirizzi delle due aree e la grandezza dell'area da spostare.

Argomenti della funzione

srcPointer — Indirizzo dell'area di memoria destinata a essere copiata. Quest'area di memoria può essere situata in qualsiasi punto della RAM, ma il suo indirizzo dev'essere allineato alle long word, cioè dev'essere un multiplo di quattro.

destPointer — Indirizzo dell'area di memoria destinata a ricevere la copia dell'area origine. Quest'area di memoria può essere situata in qualsiasi punto della RAM, ma il suo indirizzo dev'essere allineato alle long word, cioè dev'essere un multiplo di quattro.

size — Dimensione in byte dell'area di memoria da copiare. Questo argomento può rappresentare un blocco di memoria di qualsiasi estensione, ma deve sempre essere un multiplo di quattro.

Discussione

Ci sono due funzioni nell'Exec che permettono di copiare blocchi di memoria: CopyMem e CopyMemQuick. Per esempio, se si desidera copiare una bitmap da una locazione RAM a un'altra, si può usare una di queste due funzioni. Si tenga presente che CopyMem e CopyMemQuick sono funzioni di copia, non funzioni di movimento. Quando queste funzioni restituiscono il controllo, si avranno due copie dei dati in due diverse aree RAM. Si dovrebbe usare CopyMemQuick quando i blocchi sorgente e destinazione sono allineati alle long word e il blocco sorgente termina ancora allineato alle long word; altrimenti si usa la funzione CopyMem. Le due aree di memoria, inoltre, non devono sovrapporsi.

Deallocate

Sintassi di chiamata della funzione

```
Deallocate (memHeader, memBlock, byteSize)
           A0          A1        D0
```

Scopo della funzione

Questa funzione libera un blocco di memoria appartenente a un pool privato di memoria del task. Il blocco liberato torna a far parte della lista privata di memoria libera indicata dal task come argomento della funzione. Il blocco di memoria da liberare generalmente è stato allocato con la funzione Allocate, la quale alloca memoria dal pool privato di memoria libera del task.

Se l'indirizzo memBlock non è un multiplo di otto (la dimensione minima di un blocco di memoria), sarà arrotondato verso il basso. Questo permette di servirsi di tutte le routine di allocazione della memoria. Tuttavia possono verificarsi alcuni problemi se si sta liberando solo una parte di un blocco di memoria. Se byteSize vale zero non succede nulla; altrimenti la misura del blocco viene arrotondata al più vicino multiplo di otto. In questo modo, il blocco liberato si estende sempre su un numero intero di blocchi di memoria da otto byte.

Argomenti della funzione

memHeader Indirizzo della struttura MemHeader che intesta la lista privata della memoria libera mantenuta dal task.

memoryBlock Indirizzo del blocco di memoria da liberare; di solito è l'indirizzo restituito dalla funzione Allocate.

byteSize Dimensione in byte del blocco da liberare.

Discussione

Ci sono nove funzioni dell'Exec che riguardano la gestione della memoria: AllocAbs, Allocate, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. Si vedano anche le spiegazioni relative a queste funzioni.

Oltre alla lista di sistema della memoria libera, i task possono mantenere liste private, che usano per amministrare l'allocazione della memoria all'interno di aree private di memoria. Se una lista di memoria libera è privata, può essere usata e consultata solo da pochi task cooperanti.

La lista di memoria libera mentenuta dal sistema è composta da diverse sotto-liste, ognuna delle quali elenca la memoria libera all'interno di un particolare tipo e regione di memoria. Lo stesso vale per qualsiasi task che desideri gestirsi privatamente una parte della memoria libera presente nel sistema. Può per esempio riservarsi 10K di chip RAM e 10K di fast RAM. Li alloca chiamando due volte la funzione AllocMem, o chiamando una sola volta la funzione AllocEntry. Compiuta questa operazione, le due aree di memoria diventano di proprietà del task, che deve inizializzare due strutture MemHeader, una per area. Successivamente, ogni volta che desidera allocare e liberare memoria all'interno di una delle due regioni di memoria privata, deve chiamare Allocate e Deallocate, indicando come primo argomento l'indirizzo della relativa struttura MemHeader di gestione.

La memoria dovrebbe essere restituita al sistema quando il task ha concluso la sua esecuzione. Si ricordi che il sistema tiene conto soltanto della memoria libera; la ripartizione fra i task della memoria allocata non è conservata in nessuna lista di sistema. Perciò, se il task non restituisce esplicitamente la memoria che ha allocato, il sistema non può provvedere a liberarla automaticamente. Questa memoria non sarà perciò disponibile per gli altri task. Per prevenire questa evenienza è consigliabile delegare al sistema la liberazione della memoria impiegata dal task, indicandola nella lista di nome tc_MemEntry della struttura Task.

La struttura MemHeader il cui indirizzo dev'essere indicato come primo argomento della funzione Deallocate, coordina una lista privata di memoria libera definita da una serie di strutture MemChunk concatenate in una lista semplice (cioè percorribile solo in un senso). L'organizzazione di queste strutture è illustrata nella spiegazione della funzione Allocate.

Deallocate e Allocate permettono anche di limitare gli effetti negativi del fenomeno di frammentazione della memoria libera. Questo argomento è ampiamente illustrato nelle spiegazioni delle funzioni Allocate e AvailMem.

DoIO

Sintassi di chiamata della funzione

```
error = DoIO (iORequest)
D0            A1
```

Scopo della funzione

Questa funzione serve per inviare in modo sincrono una richiesta di I/O a un dispositivo precedentemente aperto tramite la funzione OpenDevice. Il task alloca una struttura IORequest, la inizializza parzialmente aprendo il dispositivo con la funzione OpenDevice, inizializza i parametri che permettono di formulare una richiesta di I/O, e la invia al dispositivo tramite la funzione DoIO. La richiesta di I/O è sempre un comando, scelto fra quelli previsti dal dispositivo indirizzato. Il comportamento sincrono della funzione fa sì che il task non riottenga il controllo fino a quando il dispositivo non ha elaborato la richiesta di I/O. La funzione DoIO richiede sempre il QuickIO, cioè l'elaborazione immediata della richiesta di I/O, come viene ampiamente spiegato in *Programmare l'Amiga Volume II*. Se la richiesta di I/O è stata elaborata con successo, il parametro io_Error della struttura IORequest contiene il valore zero, altrimenti un codice d'errore.

Argomenti della funzione

iORequest	Indirizzo di una struttura IORequest opportunamente inizializzata per formulare la richiesta di I/O.

Discussione

Ci sono otto funzioni dell'Exec che riguardano direttamente i dispositivi di I/O: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Si vedano anche le spiegazioni relative a queste funzioni.

Le richieste di I/O che intervengono nelle comunicazioni con i dispositivi non sono altro che messaggi standard. Mentre nelle comunicazioni inter-task i messaggi vengono inviati con la funzione PutMsg, nelle comunicazioni con i dispositivi occorre servirsi delle funzioni SendIO, DoIO, BeginIO. Queste funzioni non hanno bisogno di ricevere fra gli argomenti della chiamata l'indirizzo del destinatario del messaggio, dal momento che questa informazione è già presente nel messaggio stesso, nei parametri io_Device (indirizzo della struttura Device di gestione del dispositivo) e io_Unit (individua l'unità del dispositivo indirizzata). In particolare, DoIO invia una richiesta I/O a un task e attende il suo completamento. Il task deve usare questa funzione quando non può proseguire la sua esecuzione finché il dispositivo non ha eseguito la richiesta. Se il dispositivo ha diverse richieste in coda nella sua message port, il task deve attendere il suo turno. Se il dispositivo non ha richieste in coda, il task sarà immediatamente servito dal dispositivo.

Durante l'attesa richiesta dalla funzione DoIO, il task perde il controllo della CPU e quindi un altro task (o sequenza di task) può diventare attivo. Quando il dispositivo restituisce la richiesta di I/O in risposta, la funzione DoIO riottiene il controllo e quindi lo riottiene il task. Ora il task sa che il dispositivo ha elaborato la richiesta, e può accedere alla struttura di I/O per verificare l'esito.

Enqueue

Sintassi di chiamata della funzione

Enqueue (list, node)
AØ A1

Scopo della funzione

Questa funzione inserisce o aggiunge un nodo in una lista a doppia concatenazione ordinata. Il punto della lista nel quale viene inserito il nodo indicato come argomento dipende dalla priorità del nodo, cioè dal valore presente nel parametro ln_Pri della struttura Node che costituisce il nodo. Quindi questa funzione dev'essere usata con le liste i cui nodi sono ordinati per livello di priorità. I nodi sono inseriti sempre davanti al primo nodo a priorità appena inferiore. Ne consegue che Enqueue costruisce le liste in modo che i nodi di eguale priorità formino sotto-liste di tipo FIFO (First-In, First-Out, il primo entrato è il primo a uscire).

Argomenti della funzione

list — Indirizzo della struttura List che intesta la lista nella quale la funzione deve inserire il nodo.

node — Indirizzo della struttura Node che dev'essere inserita nella lista.

Discussione

L'Exec possiede diverse funzioni progettate per operare con i nodi e le liste a doppia concatenazione: Insert, Remove, AddHead, RemHead, AddTail, RemTail, Enqueue e FindName. Si noti che le funzioni Insert, Remove, AddHead e AddTail, contrariamente alla funzione Enqueue, non usano il campo priorità della struttura Node. Per questa ragione, Enqueue è l'equivalente di Insert da usarsi nel caso di liste ordinate. Essa fa in modo che le liste crescano mantenendo l'ordine nelle priorità dei nodi.

A causa di questa impostazione, tutti i nodi passati alla funzione Enqueue devono avere una priorità. RemHead rimuove il nodo con la priorità più alta della lista, mentre RemTail rimuove il nodo con la priorità più bassa. Se s'inserisce un nodo che possiede priorità uguale a un altro già presente nella lista, Enqueue inserisce il nuovo nodo in modo che la sotto-lista dei nodi di uguale priorità costituisca una lista di tipo FIFO.

FindName

Sintassi di chiamata della funzione

```
node = FindName (start, name)
D0              A0   A1
```

Scopo della funzione

Questa funzione permette ai task d'individuare un particolare nodo all'interno di una qualunque lista. La funzione cerca il nodo il cui nome corrisponde a quello indicato dal task come secondo argomento. Data una lista, la funzione è anche in grado d'iniziare la ricerca del nodo a partire da un particolare nodo, che peraltro viene escluso dalla ricerca. Il risultato di

FindName è l'indirizzo della struttura Node dotata del nome indicato come secondo argomento della funzione; se questo indirizzo vale zero significa che non è stata rintracciata nessuna struttura Node con quel nome.

Argomenti della funzione

start Indirizzo della struttura List che intesta la lista, o della struttura Node da cui la funzione deve iniziare la ricerca. Nel secondo caso, la struttura Node indicata viene saltata e la ricerca inizia con la struttura successiva.

name Indirizzo del nome della struttura Node ricercata; dev'essere una stringa di testo a terminazione nulla; la funzione confronta questa stringa in sequenza con quelle individuate dal parametro ln_Name delle strutture Node presenti nella lista.

Discussione

L'Exec ha diverse funzioni progettate per operare con i nodi e le liste a doppia concatenazione: Insert, Remove, AddHead, RemHead, AddTail, RemTail, Enqueue e FindName.

Ciascuna struttura Node in una lista può essere dotata di nome. Il nome è costituito da una stringa individuata dal parametro ln_Name nella struttura Node. FindName permette d'individuare il primo nodo che risponde a un certo nome all'interno di una data lista. Per proseguire la ricerca dopo aver individuato un nodo di nome corrispondente, è sufficiente chiamare nuovamente la funzione indicando l'indirizzo dell'ultimo nodo trovato. Se invece si desidera iniziare la ricerca partendo dalla testa della lista, occorre che nell'argomento start sia indicato l'indirizzo della struttura List che intesta la lista. Una volta ottenuto l'indirizzo della struttura Node con il nome desiderato, si potranno usare le altre funzioni di gestione delle liste per operare sul nodo.

FindPort

Sintassi di chiamata della funzione

```
msgPort = FindPort (name)
DØ                  A1
```

Scopo della funzione

Questa funzione esamina la lista di sistema PortList, quella che elenca le message port pubbliche, alla ricerca di una message port pubblica che risponda a un particolare nome. Se la trova restituisce l'indirizzo della relativa struttura MsgPort, altrimenti restituisce un indirizzo nullo.

Argomenti della funzione

name	Indirizzo della stringa a terminazione nulla che costituisce il nome della message port da individuare.

Discussione

Ci sono quattro funzioni dell'Exec che riguardano esplicitamente la gestione delle message port: AddPort, FindPort, RemPort e WaitPort. AddPort. FindPort e RemPort riguardano esclusivamente le message port pubbliche. Queste sono message port che vengono aggiunte alla lista di sistema delle message port pubbliche tramite la funzione AddPort, e sono dotate di un nome che ne permette l'individuazione. WaitPort, invece, riguarda sia le message port pubbliche sia quelle private. Strettamente correlate alle routine di gestione delle message port sono le funzioni di gestione dei messaggi: PutMsg, GetMsg e ReplyMsg.

Le message port possono essere dotate di nomi oppure no. Se la message port è pubblica deve avere un nome, che dev'essere individuato dal parametro ln_Name della sotto-struttura Node contenuta nella struttura MsgPort. Se la message port è privata, il che significa che verrà usata solo da poche routine strettamente correlate, può rimanere senza nome. Le comunicazioni possono ancora aver luogo tra queste routine strettamente unite anche se le loro message port non possiedono nomi; l'importante in questo caso è che l'indirizzo della message port sia noto a tutte le routine che devono usarla, in quanto non

sarebbe possibile ottenerlo tramite la funzione FindPort.

Se invece una message port possiede un nome ed è stata dichiarata pubblica, chiunque nel sistema può individuarla e inviarle messaggi conoscendone il nome. In questo modo la message port diventa una porta di comunicazione che può condurre anche a interazioni molto complesse fra i task.

L'indirizzo che un task ottiene chiamando la funzione FindPort viene generalmente impiegato per inviare messaggi a quella message port tramite la funzione PutMsg. Ma può anche servire per altri scopi. Per esempio, potrebbe essere l'indirizzo di una message port installata nel sistema per ricevere messaggi diretti a chiunque li desideri, cioè non diretti a un particolare task. In una simile situazione, certi task cercherebbero questa message port, che potrebbe chiamarsi per esempio "Ufficio smistamento", per lasciare messaggi a chiunque sia interessato, e altri task la cercherebbero per verificare se c'è qualche messaggio di loro interesse.

FindResident

Sintassi di chiamata della funzione

```
resident = FindResident (name)
D0                      A1
```

Scopo della funzione

Questa funzione esamina l'array di sistema dei moduli residenti per individuare la struttura Resident che risponde al nome indicato dal task come argomento. Questo array è individuato in memoria dall'indirizzo contenuto nel parametro ResModules della struttura ExecBase; ogni suo elemento è l'indirizzo di una struttura Resident, e il primo elemento di valore zero rappresenta la fine dell'array. Se FindResident viene elaborata con successo, restituisce l'indirizzo della struttura Resident desiderata, altrimenti restituisce il valore zero. Una volta che il task ha ottenuto l'indirizzo della struttura Resident cercata, può rilevare tutte le caratteristiche del modulo residente (versione, priorità d'esecuzione, tipo, indirizzo del codice d'inizializzazione...).

Argomenti della funzione

name	Indirizzo della stringa contenente il nome del modulo residente da cercare; questa stringa viene confrontata

con le stringhe puntate dai parametri rt_Name delle strutture Resident elencate nell'array di sistema.

Discussione

Ci sono tre funzioni che riguardano la gestione dei moduli residenti: FindResident, InitCode e InitResident. Si vedano anche le spiegazioni relative a queste funzioni.

La struttura Resident

Ciascun modulo residente installato nell'Amiga viene gestito attraverso una struttura di tipo Resident. Questa struttura è definita come segue nel file INCLUDE resident.h:

```
struct Resident {
    UWORD rt_MatchWord;
    struct Resident *rt_MatchTag;
    APTR rt_EndSkip;
    UBYTE rt_Flags;
    UBYTE rt_Version;
    UBYTE rt_Type;
    BYTE rt_Pri;
    char *rt_Name;
    char *rt_IdString;
    APTR rt_Init;
};
```

Ecco il significato dei parametri della struttura Resident:

- il parametro rt_MatchWord costituisce la word che dev'essere impiegata come chiave nella ricerca di un particolare modulo residente associato a una specifica struttura Resident. Tutti i moduli residenti di sistema contengono in questo parametro il valore 0x4AFC, che nel file INCLUDE exec/resident.h è definito dalla costante RTC_MATCHWORD.

- Il parametro rt_MatchTag è un puntatore e contiene sempre l'indirizzo della struttura Resident a cui appartiene, cioè l'indirizzo del parametro rt_MatchWord.

- Il parametro rt_EndSkip è un indirizzo a una locazione RAM che serve per continuare la scansione. Molto spesso individua la fine dei codici del modulo residente.

- Il parametro rt_Flags contiene i flag che indicano al sistema come e quando dev'essere inizializzato il modulo residente. Se il flag RTF_AUTOINIT è impostato, la funzione InitResident, che serve per inizializzare un particolare modulo, ne deduce che il parametro rt_Init della struttura Resident non contiene l'indirizzo di una routine d'inizializzazione, ma l'indirizzo di una serie di quattro long word che deve impiegare come argomenti della funzione MakeLibrary; in questo caso, se il modulo è di tipo NT_DEVICE, NT_LIBRARY o NT_RESOURCE, InitResident chiama rispettivamente le funzioni AddDevice, AddLibrary, o AddResource. In altre parole, se il flag RTF_AUTOINIT è impostato significa che il modulo residente dev'essere inizializzato tramite MakeLibrary. Se invece il flag RTF_AUTOINIT vale zero, la funzione InitResident provvede semplicemente a cedere il controllo alla routine d'inizializzazione puntata dal parametro rt_Init della struttura Resident. In questo caso il modulo residente dispone di una sua routine d'inizializzazione, e non viene chiamata la funzione MakeLibrary. I moduli di sistema che possiedono il flag RTF_AUTOINIT impostato sono Expansion, Ramlib, Audio, Intuition, e Mathffp. Si veda anche la spiegazione della funzione InitResident. Oltre al flag RTF_AUTOINIT, esiste anche il flag RTF_COLDSTART, il quale differenzia i moduli residenti che devono essere inizializzati durante il boot (flag RTF_COLDSTART impostato) da quelli che invece non devono essere inizializzati durante il boot (flag RTF_COLDSTART azzerato). La presenza di questo flag permette al sistema, durante il boot, di chiamare la funzione InitCode per inizializzare i moduli residenti che possiedono questo flag impostato. Si veda anche la spiegazione della funzione InitCode.

- Il parametro rt_Version è il numero di versione del modulo residente. Viene usato dalla funzione InitCode per inizializzare tutti i moduli residenti dotati di un numero di versione superiore o uguale a quello indicato dal task come argomento.

- Il parametro rt_Type indica il tipo assegnato al modulo residente. Si tratta degli stessi tipi previsti per i nodi: per esempio, NT_LIBRARY, NT_DEVICE, NT_UNKNOWN. Queste costanti sono definite nel file INCLUDE exec/nodes.h. Il parametro rt_Type assume particolare importanza quando viene chiamata la funzione InitResident per inizializzare un modulo residente nella cui struttura Resident il parametro rt_Flags ha il flag RTF_AUTOINIT impostato. In questo caso, infatti, la funzione accede al parametro rt_Type per decidere se chiamare AddDevice, AddLibrary o AddResource durante l'inizializzazione del modulo residente.

- Il parametro rt_Pri è la priorità d'inizializzazione del modulo residente. Dal momento che i moduli residenti sono elencati nell'array di sistema secondo il loro ordine di priorità, la priorità determina anche l'ordine in cui la funzione InitCode inizializza i moduli residenti.

- Il parametro rt_Name è un puntatore a una stringa a terminazione nulla che contiene il nome del modulo residente. Questo parametro è usato dalla funzione FindResident per localizzare una struttura Resident con un nome specifico.

- Il parametro rt_IdString è un puntatore a una stringa a terminazione nulla che identifica il modulo residente.

- Il parametro rt_Init è un puntatore alla locazione di memoria dove iniziano i codici d'inizializzazione del modulo residente (flag RTF_AUTOINIT azzerato), o inizia la serie di quattro long word che devono essere passate a MakeLibrary come argomenti (flag RTF_AUTOINIT impostato).

Ciascun modulo residente possiede un nome. Il nome viene usato insieme al numero di versione e alla stringa d'identificazione per individuare univocamente il modulo residente. Una volta che un modulo residente è stato caricato in RAM, si può usare la funzione FindResident per localizzare la struttura Resident che lo definisce.

FindSemaphore

Sintassi di chiamata della funzione

```
signalSemaphore = FindSemaphore (name)
D0                               A1
```

Scopo della funzione

Questa funzione tenta d'individuare una struttura SignalSemaphore dotata di un particolare nome. FindSemaphore svolge la ricerca all'interno della lista di sistema dei semafori di segnalazione. Se FindSemaphore ha successo, restituisce l'indirizzo della prima struttura SignalSemaphore con quel nome; se invece non trova nessuna struttura SignalSemaphore con quel nome, restituisce un indirizzo di valore zero.

Argomenti della funzione

name	Indirizzo della stringa che costituisce il nome del

semaforo che si desidera individuare. Questa stringa viene confrontata con quelle individuate dai parametri ln_Name delle sotto-strutture Node presenti nelle strutture SignalSemaphore.

Discussione

Ci sono sette funzioni che riguardano i semafori di segnalazione e due funzioni che riguardano la lista dei semafori di segnalazione. Queste funzioni sono esposte nel corso della spiegazione di AddSemaphore.

Talvolta i task possono trovarsi nella necessità d'individuare una specifica struttura SignalSemaphore. FindSemaphore fornisce la possibilità di farlo. Una volta che si conosce il nome del semaforo si può usare FindSemaphore per ottenerne l'indirizzo in memoria.

FindTask

Sintassi di chiamata della funzione

```
taskCS = FindTask (name)
DØ                 A1
```

Scopo della funzione

Questa funzione esamina le liste per la gestione dei task alla ricerca della struttura Task che risponde al nome indicato come argomento: se la trova, ne restituisce l'indirizzo. Questa funzione può anche servire a un task per individuare in memoria la propria struttura Task; basta infatti indicare come argomento di FindTask un indirizzo nullo.

Argomenti della funzione

name — Indirizzo della stringa di testo che contiene il nome del task da individuare; questa stringa viene confrontata con quella individuata dai parametri ln_Name contenuti nelle strutture Task. Si noti che se il task indica come parametro un indirizzo nullo, la funzione

restituisce l'indirizzo della struttura Task relativa a quel task.

Discussione

Ci sono quattro funzioni dell'Exec che riguardano specificamente la gestione dei task: AddTask, FindTask, RemTask e SetTaskPri. Si vedano anche le spiegazioni relative a queste funzioni. La struttura Task è definita nei file INCLUDE tasks.h (per il linguaggio C) e tasks.i (per il linguaggio Assembly).

A volte è necessario conoscere la posizione in RAM della struttura Task di gestione di un task, per esempio quando se ne vogliono modificare alcuni parametri. Questo è lo scopo della funzione FindTask, la quale esamina le liste di sistema di gestione dei task alla ricerca della struttura Task che risponde al nome indicato come argomento; quando individua una struttura Task con quel nome, ne restituisce l'indirizzo.

FreeEntry

Sintassi di chiamata della funzione

FreeEntry (memList)
A0

Scopo della funzione

Questa funzione libera tutti i blocchi di memoria definiti nella struttura MemList che riceve come argomento. L'indirizzo della struttura MemList è in genere quello restituito dalla funzione AllocEntry, la quale serve per allocare con un'unica chiamata diversi blocchi di memoria. Quando FreeEntry restituisce il controllo, tutti i blocchi di memoria indicati nella struttura MemList sono stati restituiti al pool di memoria libera del sistema.

Argomenti della funzione

memList Indirizzo della struttura MemList che elenca nell'array ml_ME tutte le strutture MemEntry che definiscono i vari blocchi di memoria da liberare.

Discussione

Ci sono nove funzioni dell'Exec che riguardano la gestione della memoria: AllocAbs, Allocate, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. Si vedano anche le spiegazioni relative a queste funzioni.

Tre metodi per liberare la memoria

Così come esistono diverse soluzioni per allocare memoria (AllocMem, AllocEntry, Allocate), esistono altrettanti metodi per liberarla. Qualsiasi metodo si scelga, la memoria viene sempre restituita a un pool di memoria libera, che può essere del task, oppure del sistema. Per liberare memoria possono essere usate le funzioni Deallocate, FreeMem o FreeEntry. Ciascuna funzione opera in modo diverso:

1. La funzione Deallocate libera un determinato blocco di memoria precedentemente allocato con la funzione Allocate. Essa restituisce tale blocco alla lista di memoria libera indicata dal task come argomento, e non alla lista di sistema. Pertanto presuppone che il task possieda e mantenga un personale pool di memoria libera tramite un'apposita lista. I vantaggi derivanti dall'uso di un pool privato di memoria libera e delle funzioni Allocate e Deallocate sono ampiamente trattati nelle descrizioni di queste due funzioni.

2. La funzione FreeMem libera un blocco di memoria allocato con la funzione AllocMem, restituendolo alla lista di sistema della memoria libera. Queste due funzioni, come AllocEntry e FreeEntry, allocano e liberano memoria di sistema.

3. La funzione FreeEntry restituisce alla memoria libera di sistema tutti i blocchi di memoria allocati tramite la funzione AllocEntry. Le caratteristiche di questi blocchi devono essere elencate nelle strutture MemEntry di una struttura MemList opportunamente inizializzata. Generalmente si tratta della struttura MemList creata dalla funzione AllocEntry. I blocchi di memoria liberati vengono restituiti alla lista di sistema della memoria libera.

Strutture e puntatori

Le funzioni per la gestione della memoria hanno a che fare con diverse strutture e puntatori:

1. La struttura MemList serve per elencare una serie di strutture MemEntry, ognuna delle quali può rappresentare un blocco di memoria

da allocare o un blocco di memoria già allocato, a seconda che la struttura MemList debba ancora essere passata ad AllocEntry o debba essere passata a FreeEntry. Le strutture MemEntry che appartengono a una struttura MemList sono elencate all'interno dell'array ml_ME della struttura MemList, la quale prevede anche un parametro che ne indica il numero. La struttura MemList è usata dalle funzioni AllocEntry e FreeEntry.

2. La struttura MemEntry è una sotto-struttura della struttura MemList, e serve per descrivere un blocco di memoria da allocare o già allocato. Essa è composta dai seguenti parametri: l'unione me_Un, che può essere utilizzata dal task come parametro me_Reqs per indicare ad AllocEntry gli attributi del blocco di memoria da allocare, oppure può essere utilizzata dalla funzione AllocEntry come parametro me_Addr per indicare al task e a FreeEntry l'indirizzo del blocco allocato (ovviamente, me_Reqs e me_Addr occupano le stesse locazioni di memoria, e pertanto si escludono a vicenda); segue il parametro me_Length, che il task deve inizializzare con la quantità di memoria che desidera venga allocata per questo blocco di memoria.

3. Le funzioni Allocate e AllocMem restituiscono come risultato l'indirizzo del blocco di memoria allocato.

4. La struttura MemHeader serve ai task per creare e gestire liste private di memoria libera, dalle quali possono allocare e liberare memoria tramite le funzioni Allocate e Deallocate, che a differenza di AllocMem, FreeMem, AllocEntry e FreeEntry non agiscono sulla lista di sistema della memoria libera. Le strutture MemHeader possono anche essere concatenate fra loro per creare una lista vera e propria; in essa ogni struttura MemHeader individua una particolare area della memoria di sistema interamente allocata con una chiamata ad AllocMem o AllocEntry, e che dev'essere gestita come memoria privata, a uso esclusivo del task. Ogni struttura MemHeader intesta una sotto-lista di chunk di memoria, ovvero di strutture MemChunk, ognuna delle quali definisce un blocco di memoria libera. Ognuna di queste sotto-liste costituisce la lista della memoria libera nella gestione di una particolare area privata di memoria.

FreeMem

Sintassi di chiamata della funzione

FreeMem (memBlock, byteSize)
A1 DØ

Scopo della funzione

Questa funzione libera un blocco di memoria restituendolo alla lista di sistema della memoria libera. Generalmente si tratta di un blocco allocato con la funzione AllocMem.

Argomenti della funzione

memBlock Indirizzo del particolare blocco di memoria da liberare; questo indirizzo è generalmente quello restituito dalla funzione AllocMem.

byteSize Dimensione in byte del blocco di memoria da liberare.

Discussione

Ci sono nove funzioni dell'Exec che riguardano la gestione della memoria: AllocAbs, Allocate, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. Per maggiori particolari sulla gestione della memoria si consultino le spiegazioni di queste funzioni.

La funzione FreeMem richiede che siano specificati l'indirizzo del blocco di memoria da liberare e la sua dimensione. La funzione restituisce il blocco indicato al pool di sistema della memoria libera, e quindi non richiede d'indicare una lista privata di memoria libera, dal momento che impiega la lista di sistema. Questa funzione svolge lo stesso compito che Deallocate esegue su una particolare lista privata di memoria libera.

Un esempio d'istruzione che libera un blocco di memoria utilizzando FreeMem è:

```
FreeMem (memptr, 100)
```

Questa istruzione libera 104 byte di memoria a partire dall'indirizzo memptr, quello originariamente restituito dalla funzione AllocMem. Si noti che gli originali 100 byte vengono arrotondati fino al successivo valore multiplo di otto (le funzioni di gestione della memoria lavorano infatti su un blocco minimo di otto byte, la minima misura di blocco di memoria con cui il sistema può operare). Questo comportamento è perfettamente coerente con quello della funzione AllocMem, che ha sicuramente allocato 104 byte se il task ne ha richiesti 100.

La funzione FreeMem non restituisce alcun risultato per indicare se il blocco di memoria indicato come argomento è veramente da liberare o meno. Si tenga quindi presente che se si cerca di liberare un blocco di memoria contenuto in una regione di memoria che il sistema ritiene già libera, si causa il crash del sistema. Se questo accade, l'unica soluzione consiste nel riavviare il sistema (procedere cioè a una nuova operazione di boot).

FreeSignal

Sintassi di chiamata della funzione

FreeSignal (signalNum)
DØ

Scopo della funzione

Questa funzione libera il bit di segnale indicato dal task come argomento; questo bit è quello che il task aveva allocato con una chiamata alla funzione AllocSignal. I bit di segnale di un task che dopo essere stati allocati devono essere liberati tramite la funzione FreeSignal, possono essere liberati solo dal task a cui appartengono. Dopo la chiamata a FreeSignal, il bit di segnale liberato torna a essere disponibile per nuove allocazioni e assegnazioni alle message port del task. Questa funzione agisce sul parametro tc_AllocSignal della struttura Task che definisce il task.

Argomenti della funzione

signalNum — Il numero del bit di segnale che dev'essere liberato (numero che può variare fra 0 e 31). Generalmente è il valore restituito dalla funzione AllocSignal.

Discussione

Ci sono sei funzioni dell'Exec che riguardano i segnali software nel sistema Amiga: AllocSignal, FreeSignal, SetExcept, SetSignal, Signal e Wait. Inoltre possono essere incluse nella categoria delle funzioni di gestione dei segnali GetMsg, PutMsg e ReplyMsg; i segnali, infatti, sono spesso provocati dall'arrivo di un messaggio in una message port predisposta per inviare segnali ai task a cui appartengono. Si vedano anche le spiegazioni relative a queste funzioni.

I task comunicano tra loro scambiandosi messaggi attraverso le message port. I task inviano e ricevono anche messaggi dai dispositivi, trasmettendo le informazioni attraverso le strutture IORequest.

I segnali di cui un task può servirsi devono essere allocati usando la funzione AllocSignal. In seguito, il task ha la facoltà di assegnare quel segnale, o bit di segnale, a una delle sue message port, a una routine di interrupt o a un altro task. Nel primo caso la message port, opportunamente definita, invia il segnale al task ogni volta che riceve un messaggio; questo comportamento è controllato automaticamente dal sistema. Nel secondo e nel terzo caso, sono invece la routine di interrupt o il secondo task a inviare quel segnale tramite la funzione Signal quando si verificano determinate condizioni. A prescindere dalla causa di un segnale, comunque, il task si accorge del suo invio soltanto se ne è entrato in attesa attraverso la funzione Wait, la quale lo "risveglia" quando si verifica uno dei segnali indicati come argomento. Altrimenti non c'è alcun modo perché un task possa accorgersi dell'arrivo di un segnale.

A ciascuna message port che crea, il task può assegnare uno dei suoi bit di segnale. Per farlo deve prima allocare il bit di segnale tramite la funzione AllocSignal, e poi memorizzare il valore restituito da questa funzione nel parametro mp_SigBit nella struttura MsgPort che definisce quella message port. Infine, deve impostare a 1 il flag PA_SIGNAL del parametro mp_Flags della stessa struttura MsgPort. D'ora in poi, se il task entra in attesa di quel segnale tramite la funzione Wait, la message port lo risveglia generando un segnale quando arriva il messaggio. Si noti che le message port devono sempre appartenere a uno e un solo task (quello indicato nei loro parametri mp_SigTask), e che a ogni message port può essere assegnato uno e un solo segnale.

Se si desidera cambiare l'assegnazione di alcuni di questi bit di segnale occorre prima liberarli tramite la funzione FreeSignal. L'allocazione di un bit di segnale si svolge basandosi sul task interessato. Se si libera un bit di segnale di un task, non si coinvolge alcun bit di segnale usato da altri task nel sistema. Se ci sono dieci task nel sistema, ciascuno dei quali ha allocato tutti i bit di segnale a sua disposizione (16), si avranno in tutto 160 bit di segnale allocati.

Oltre al parametro tc_AllocSignal, la struttura Task contiene il parametro tc_SigRecvd, che viene usato dalla funzione Wait per verificare se il segnale è fra quelli di cui il task è in attesa; se infatti si verifica un segnale fra quelli allocati, ma il task non l'ha indicato come argomento della funzione Wait, la funzione non restituisce il controllo e continua a lasciare in attesa il task. Generalmente i task non hanno necessità di accedere direttamente a questo

parametro, che Wait provvede sempre ad azzerare prima di restituire il controllo.

Se al task viene associata una routine di exception, è necessario indicare al sistema quali segnali causano l'exception e quindi l'esecuzione della relativa routine di gestione. Questo si ottiene usando il parametro tc_SigExcept della struttura Task del task a cui viene associata la routine di exception, nel quale i bit a 1 indicano quali segnali provocano l'esecuzione della routine di gestione delle exception. Questi bit di segnale devono ovviamente essere fra quelli allocati.

Se non si desidera l'elaborazione di alcuna exception, il parametro tc_SigExcept dev'essere impostato a zero. Dato che vi sono 32 bit di segnale disponibili, sono potenzialmente impiegabili 32 bit di exception.

Attenzione: i segnali non possono essere allocati o liberati dai codici di gestione delle exception, cioè dai codici di interrupt. Non esiste la possibilità di scambio di messaggi tra le routine di exception e i task. I codici di interrupt, invece, hanno la facoltà di chiamare la funzione Signal per inviare segnali ai task.

FreeTrap

Sintassi di chiamata della funzione

FreeTrap (trapNum)
DØ

Scopo della funzione

Questa funzione libera un numero di trap precedentemente allocato, al fine di renderlo riutilizzabile. La chiamata dev'essere effettuata durante l'esecuzione del task nel quale la trap era stata allocata con la funzione AllocTrap.

Argomenti della funzione

trapNum Numero di trap da liberare (un valore tra 0 e 15).

Discussione

Ci sono due funzioni dell'Exec che riguardano esplicitamente le trap nel sistema Amiga: AllocTrap e FreeTrap. La funzione AllocTrap permette a un task di allocare una trap per uso privato. La funzione FreeTrap restituisce un particolare numero di trap alla lista delle trap non allocate, cioè disponibili. Questa lista viene mantenuta automaticamente dal sistema.

Nell'aggiungere o togliere trap dai propri task, si dovrebbe tener presente che quando il 68000 rileva una trap, il task perde il controllo in favore di alcune routine dell'Exec. Queste routine rilevano quale task è in esecuzione e se la trap deve causare l'esecuzione della routine di trap di quel task; in caso affermativo, il controllo viene ceduto alla routine di trap. Si ricordi che l'indirizzo della routine di trap dev'essere indicato nel parametro tc_TrapCode della struttura Task del task considerato.

Quando si verifica una trap, il supervisor stack del 68000 contiene le seguenti informazioni:

- un insieme di dati che viene salvato per preservare il contenuto di alcuni registri della CPU. I dati vengono salvati secondo l'organizzazione dettata dallo stack frame della particolare CPU presente nella macchina (lo stack frame è il formato con cui viene salvato lo stato della CPU quando si verifica una exception, e varia da CPU a CPU). Nell'ambiente di programmazione dell'Amiga ci riferiamo allo stack frame con il termine trap frame quando ci riferiamo alle trap. I manuali della Motorola illustrano gli stack frame per ogni CPU della famiglia 680XX.

- Il numero di trap. Un numero da 0 a 255 come descritto nel manuale del 68000.

Quando il sistema rileva una trap, esegue l'istruzione Assembly:

TRAP #n

dove *n* è un numero da 0 a 15. Questa istruzione del 68000 permette di causare una delle 16 trap software disponibili; quando il 68000 la esegue, aggiunge a *n* il numero 32 per individuare il corretto vettore di trap, al quale cede poi il controllo. Pertanto, le trap causabili con l'istruzione TRAP #n possono provocare l'esecuzione delle routine i cui indirizzi sono contenuti nei vettori dal 32 al 47 della tavola di vettori della CPU. Il numero che viene quindi memorizzato nello stack è *n* + 32. Questo significa che, per convenzione, le istruzioni di trap software generano numeri di trap compresi fra 32 e 47.

Il task potrebbe non essere in grado di elaborare tutte le trap software che si verificano durante la sua esecuzione, e quindi occorre indicare al sistema, tramite la funzione AllocTrap, le trap di cui il task può prendersi cura. Un'apposita routine deve prelevare dallo stack il codice di trap e stabilire se corrisponde a una trap che il task è in grado di elaborare. Il controllo avviene

creando una maschera di bit con il numero di trap e confrontandola con la maschera dei numeri di trap allocati contenuta nel parametro tc_TrapAlloc della struttura Task (inizializzato dalla funzione AllocTrap). Tramite una routine in linguaggio Assembly si effettuano le seguenti operazioni:

1. Si salvano i registri che saranno usati nella stessa routine.

2. Si preleva dallo stack il numero di trap.

3. Si confronta il numero di trap con i numeri di trap allocati dal task per stabilire se corrisponde a una delle trap che il task richiede di elaborare personalmente. Se il numero di trap è fra quelli allocati la trap può essere elaborata, altrimenti il controllo viene ceduto al sistema, il quale procede con l'elaborazione standard della trap. Naturalmente questo evento fa sì che il sistema sospenda il task che era in esecuzione quando si è verificata la trap.

4. Si effettuano le necessarie operazioni di "pulizia" e si esce dalla routine.

Ci si ricordi sempre che l'elaborazione delle trap avviene nel modo supervisor della CPU. In tale condizione, tutte le caratteristiche multitasking del sistema sono disabilitate, e quindi lo scambio di controllo fra i task è sospeso. Non è consigliabile che questa situazione duri troppo a lungo. Per questa ragione, si dovrebbero creare routine di trap quanto più brevi possibile.

GetCC

Sintassi di chiamata della funzione

conditions = GetCC()
DØ: Ø-4

Scopo della funzione

Questa funzione rileva e restituisce il valore contenuto nel registro di stato della CPU, qualsiasi sia il tipo di CPU Motorola 680XX installato sulla macchina. Utilizzando questa funzione per rilevare lo stato della CPU, si rendono i propri task compatibili, almeno sotto questo aspetto, con le CPU 68010/20/30 della Motorola.

Il registro di stato della CPU è costituito da una word, nella quale gli otto bit più significativi (8-15) sono accessibili solo quando è attivo il modo

supervisor della CPU (questo byte del registro di stato è detto system byte), mentre gli otto bit più bassi (0-7) sono accessibili anche quando è attivo il modo user (questo byte del registro di stato è detto user byte). Di questi 16 bit solo 10 vengono impiegati dal 68000 (il 68030 ne usa 12, due in più nel system byte). Quando è attivo il modo user, i bit che vengono impiegati sono cinque, dal bit 0 al bit 4, e vengono chiamati codici di condizione; lo user byte è uguale per tutti i microprocessori della famiglia 680XX. La funzione GetCC restituisce nei primi cinque bit del registro D0 i valori assunti dai cinque codici di condizione della CPU.

Argomenti della funzione

Questa funzione non prevede argomenti.

Discussione

Ciascuna CPU della famiglia Motorola 680XX genera fino a cinque codici di condizione durante l'esecuzione delle sue istruzioni. Questi codici di condizione sono contenuti nei primi cinque bit del registro di stato della CPU. Ecco i loro significati:

N (negative)	Questo bit (il bit 3) viene impostato dalla CPU quando il risultato dell'operazione ha il bit più significativo impostato; altrimenti viene azzerato.
Z (zero)	Questo bit (il bit 2) viene impostato dalla CPU quando il risultato dell'operazione vale zero; altrimenti viene azzerato.
V (overflow)	Questo bit (il bit 1) viene impostato dalla CPU quando un'istruzione eseguita dalla CPU genera una condizione di overflow. Questo si verifica quando il risultato di un'operazione aritmetica non può essere rappresentato nel numero di bit del registro o della locazione di memoria destinazione.
C (carry)	Questo bit (il bit 0) viene impostato dalla CPU quando si verifica un riporto in un'operazione aritmetica, che può essere anche una sottrazione. In caso contrario viene azzerato.
X (extend)	Questo bit (il bit 4) non viene alterato dai trasferimenti di dati. Quando lo si impiega, viene impostato allo stesso stato del bit di carry.

Attenzione: vi sono anche istruzioni della CPU che lasciano inalterati alcuni di questi flag, cioè non si preoccupano né d'impostarli né di azzerarli.

I primi quattro bit rappresentano condizioni "vere" quando sono a 1, e riflettono il risultato di un'operazione svolta dalla CPU. Il bit X è disponibile per le operazioni in multi-precisione (o estese).

Se si vuole che i programmi siano compatibili verso l'alto con le CPU 68010, 68020, e 68030, si devono evitare certe istruzioni in linguaggio Assembly che con il 68000 sono eseguibili in modo user, mentre con i più recenti microprocessori sono diventate privilegiate, cioè eseguibili soltanto in modo supervisor. In particolare, l'istruzione MOVE SR, <ea> è un'istruzione normale con il 68000, mentre è un'istruzione privilegiata con le CPU 68010, 68020, 68030. Quindi, un task che la contiene funziona correttamente se viene elaborato dal 68000, mentre causa una trap da violazione di privilegio se viene elaborato dalle CPU 68010/20/30. Quando è attivo il modo user, queste CPU richiedono che venga impiegata l'istruzione MOVE CCR, <ea>. Per evitare questo problema di compatibilità, è bene che i task impieghino sempre la funzione GetCC per rilevare i codici di condizione.

GetMsg

Sintassi di chiamata della funzione

```
message = GetMsg (msgPort)
DØ                AØ
```

Scopo della funzione

Questa funzione permette a un task di ottenere l'indirizzo del messaggio che si trova alla sommità della coda alla message port indicata come argomento. La funzione, oltre a restituire l'indirizzo di quel messaggio, provvede a rimuoverlo dalla coda.

GetMsg non mette il task in attesa: se non è presente alcun messaggio nella coda alla message port, restituisce il valore zero. Se un task si trova nella necessità di attendere che giunga un messaggio a una delle sue message port, deve chiamare la funzione Wait o WaitPort, le quali lo fanno entrare in attesa. La prima restituisce il controllo quando giunge uno dei segnali indicati dal task come argomenti, fra i quali dev'esserci il segnale assegnato alla specifica message port. WaitPort, invece, attende che giunga un messaggio nella coda alla message port indicata come argomento. Dal momento che ogni message port deve appartenere a uno e un solo task, in ogni momento solo un task può attendere che in quella message port giunga un messaggio. Quando il task

riottiene il controllo da Wait o WaitPort (nel caso di Wait deve verificare in quale message port è giunto il messaggio), può chiamare la funzione GetMsg per svuotare la coda alla message port. È necessario chiamare GetMsg diverse volte in quanto mentre il task riottiene il controllo da Wait o WaitPort, altri messaggi possono giungere nella coda alla message port, ma il task non potrebbe accorgersi dei segnali pervenuti.

GetMsg restituisce l'indirizzo del primo messaggio disponibile, che è necessariamente il messaggio più vecchio tra quelli pervenuti dopo l'ultimo svuotamento della coda alla message port. Quando la coda non contiene più messaggi, la funzione restituisce un indirizzo nullo.

Quando un task riceve un messaggio, può trattarsi della risposta a un messaggio che aveva precedentemente inviato, o del messaggio di un altro task. Generalmente si tende a differenziare la message port adibita a ricevere normali messaggi da quella adibita a ricevere le risposte ai propri messaggi (che viene chiamata "reply port"). Comunque, se il messaggio è una risposta, il task può riutilizzarne la struttura come preferisce (può anche disallocarla), mentre se non è una risposta può soltanto leggerlo, alterandone eventualmente alcuni parametri, e deve poi restituirlo al mittente tramite la funzione ReplyMsg.

Argomenti della funzione

msgPort — Indirizzo della struttura MsgPort che definisce la message port dalla quale si desidera estrarre il primo e più vecchio messaggio in coda.

Discussione

Ci sono tre funzioni dell'Exec che riguardano esplicitamente l'elaborazione dei messaggi: PutMsg (inoltra il messaggio), GetMsg (riceve il messaggio) e ReplyMsg (restituisce al mittente il messaggio). Siccome i messaggi vengono inviati alle message port, sono importanti anche le funzioni di gestione delle message port: AddPort, FindPort, RemPort e WaitPort. Si vedano anche le spiegazioni relative a queste funzioni.

A proposito dei messaggi nel sistema Amiga esistono alcune nozioni di base: i messaggi possono essere scambiati fra due task, anche di sistema, o fra task e dispositivi di I/O. In questo secondo caso i task devono inviare messaggi ai dispositivi per comunicare le loro richieste di I/O, e i dispositivi li restituiscono (in genere) per indicare l'esito della richiesta e gli eventuali risultati. I dispositivi sono controllati da opportuni task residenti in attesa di messaggi dai task definiti dai programmatori. Essi gestiscono i dispositivi di I/O installati nel sistema: la tastiera, il drive del disco, il disco rigido, il mouse, la porta giochi, e ogni altro elemento hardware collegato al sistema. Questi task vengono classificati come task di gestione dei dispositivi e hanno il compito di

attendere gli eventi di input.

Nello scambio di messaggi con i dispositivi vengono usate le funzioni GetMsg, Wait e WaitPort, mentre non viene usata la funzione PutMsg; al suo posto infatti i task devono impiegare le funzioni DoIO, SendIO, e BeginIO. Si tratta di un caso particolare nel meccanismo di scambio dei messaggi. Nelle comunicazioni inter-task, invece, queste funzioni non vengono impiegate, e i task devono usare la funzione PutMsg.

I task comunicano fra loro inoltrando messaggi alle rispettive message port. Quando un task esegue la funzione PutMsg, il messaggio indicato come argomento viene inserito nella coda alla message port, anch'essa indicata come argomento della funzione. In maniera analoga, la funzione ReplyMsg inserisce il messaggio indicato nella coda a una reply port del sistema, ma ricava l'indirizzo di questa particolare message port dalla struttura stessa del messaggio (si ricordi infatti che nella struttura Message di ogni messaggio si può indicare il mittente memorizzandone l'indirizzo nel parametro mn_ReplyPort; se non viene memorizzato alcun indirizzo significa che il mittente non si aspetta la restituzione del messaggio). Le code alle message port sono liste a doppia concatenazione gestite in modo FIFO, che servono ad accumulare i messaggi a mano a mano che giungono; il task proprietario di una message port deve periodicamente sondarla ed estrarre i messaggi in essa accodati, altrimenti la coda potrebbe diventare troppo lunga. Per ottenere i messaggi può utilizzare la funzione GetMsg, che li estrae dalla coda della message port indirizzata e restituisce i messaggi secondo l'ordine in cui sono pervenuti.

Quando una message port riceve un messaggio nella sua coda, il sistema può scegliere tra vari comportamenti.

1. Può impostare il bit di segnale del task associato con quella message port. Quest'azione corrisponde a inviare un segnale al task, il quale riottiene il controllo se ne era in attesa.

2. Può riattivare il task e contemporaneamente causare un interrupt software. Questo forza l'esecuzione della routine di exception del task. Dopo che è stata eseguita la routine di exception, il task riprende il controllo della CPU dalla stessa posizione in cui è stato interrotto. Naturalmente, tutto questo accade se il bit di segnale assegnato dal task alla message port è in grado di produrre una exception.

Lo scambio dei messaggi che avviene all'interno dell'Amiga permette ai task di comunicare fra loro e di non entrare in dannosi quanto poco efficienti loop di attesa dei messaggi. Quando un task entra in attesa di un segnale (in genere un messaggio da una delle sue message port), causa uno scambio di controllo della CPU, e non riottiene più cicli della CPU finché non giunge un messaggio in una delle sue message port.

InitCode

Sintassi di chiamata della funzione

InitCode (startClass, version)
DØ D1

Scopo della funzione

Questa funzione inizializza tutti i moduli residenti che hanno i flag indicati dal task come primo argomento impostati a 1, e che possiedono un numero di versione uguale o maggiore di quello indicato dal task come secondo argomento della funzione. I moduli residenti vengono inizializzati nella sequenza con cui appaiono nell'array di sistema dei moduli residenti individuato dal parametro ResModules della struttura ExecBase. Questo ordine è determinato dalle loro priorità, cioè dai valori contenuti nei parametri rt_Pri delle relative strutture Resident.

Argomenti della funzione

startClass Il task deve indicare quali flag devono risultare impostati a 1 nel parametro rt_Flags delle strutture Resident dei moduli. InitCode chiama InitResident e inizializza soltanto quei moduli nei quali i flag indicati dal task nell'argomento startClass risultano impostati a 1. Questo parametro, in pratica, indica a InitCode la classe alla quale devono appartenere i moduli residenti da inizializzare. Il sistema prevede i flag RTF_AUTOINIT e RTF_COLDSTART, ma moduli residenti non di sistema potrebbero essere caratterizzati anche da altri flag a 1 nel parametro rt_Flags delle loro strutture Resident.

version Numero di versione; questo valore viene confrontato con quelli memorizzati nei parametri rt_Version delle strutture Resident. È il primo controllo svolto da InitCode per distinguere i moduli da inizializzare (sempre che i flag a 1 siano quelli indicati dal task) da quelli da non devono essere inizializzati perché il loro numero di versione è troppo basso. I moduli

inizializzabili sono soltanto quelli che possiedono un numero di versione maggiore o uguale a quello indicato dal task come secondo argomento della funzione.

Discussione

Ci sono tre funzioni dell'Exec che riguardano i moduli residenti: FindResident, InitCode e InitResident. Per comprendere come vengono usati e gestiti i moduli residenti si consultino le spiegazioni di queste funzioni.

La funzione InitCode permette d'inizializzare tutti i moduli residenti che rispondono alle caratteristiche indicate dal task negli argomenti della funzione. Il primo criterio di selezione dei moduli da inizializzare rispetto a quelli da non inizializzare è il numero di versione: i moduli che possiedono nel parametro rt_Version della struttura Resident un numero inferiore a quello indicato dal task come secondo argomento non vengono sicuramente inizializzati. Questo consente di riconoscere e scartare i moduli residenti obsoleti.

Il secondo criterio di selezione, subordinato al primo, controlla se i flag indicati dal task come primo argomento della funzione sono impostati o meno nei parametri rt_Flags delle strutture Resident che hanno superato con successo il controllo della versione: le strutture Resident nelle quali i flag indicati dal task sono a 1 vengono inizializzate chiamando la funzione InitResident, mentre le altre strutture Resident vengono scartate. Il task può indicare qualsiasi flag nel primo argomento della funzione. Se per esempio desidera che vengano inizializzati tutti i moduli residenti che abbiano il flag RTF_AUTOINIT impostato e che abbiano un numero di versione superiore o uguale a 34, dev'essere impartita la seguente istruzione in linguaggio C:

```
InitCode (RTF_AUTOINIT, 34);
```

Oppure, può decidere d'inizializzare tutti i moduli residenti che hanno anche il flag RTF_COLDSTART impostato con la seguente istruzione in linguaggio C:

```
InitCode (RTF_AUTOINIT|RTF_COLDSTART, 34);
```

Naturalmente, il criterio di selezione dei flag permette anche d'inizializzare particolari moduli residenti non di sistema.

L'array dei moduli residenti

Nella struttura ExecBase il parametro ResModules individua in memoria un array d'indirizzi. Ogni indirizzo individua in memoria una struttura Resident che definisce un modulo residente. Il primo indirizzo nullo di questo array ne definisce la fine. Quando le funzioni InitCode e FindResident esaminano questo

array, interrompono la lettura al primo indirizzo nullo che incontrano. Una particolarità di queste due funzioni, e quindi anche dell'array, è che se incontrano un elemento dell'array nel quale il bit 31, il più significativo, è impostato, trattano questo elemento come l'indirizzo di un nuovo array di moduli residenti su cui continuare la lettura (ovviamente, provvedendo prima ad azzerarne il bit 31). Questo sistema consente di legare insieme diversi array all'array di sistema dei moduli residenti, come se fossero delle prolunghe; le funzioni InitCode e FindResident sono predisposte per riconoscere le concatenazioni e trattare la serie di array come un array unico. Questo sistema è stato ideato per consentire di sfruttare l'elemento nullo dell'array come elemento di legame dell'array di sistema con un nuovo array di moduli residenti non standard creato dal programmatore. In questo modo, non è necessario estendere l'array di sistema, operazione che sarebbe anche molto rischiosa; basta allocare un nuovo array e memorizzarne l'indirizzo con il bit 31 impostato nell'elemento nullo dell'array di sistema.

InitResident

Sintassi di chiamata della funzione

InitResident (resident, segList)
A1 D1

Scopo della funzione

Questa funzione permette d'inizializzare un particolare modulo residente conoscendo l'indirizzo della struttura Resident che lo definisce. Questa struttura non deve necessariamente appartenere all'array di sistema dei moduli residenti. Il task, oltre all'indirizzo della struttura Resident che definisce il modulo da inizializzare, deve indicare come secondo argomento la lista dei segmenti.

Argomenti della funzione

resident	Indirizzo della struttura Resident che definisce il modulo da inizializzare.
segList	Indirizzo di una lista di segmenti per il particolare modulo residente.

Discussione

Ci sono tre funzioni dell'Exec che riguardano i moduli residenti: FindResident, InitCode e InitResident.

Il codice di ciascun modulo residente è composto da segmenti di codice. Ciascun segmento contiene parti di codice eseguibile e punta al segmento seguente. Una lista di segmenti è una lista che collega tutti i segmenti di codice raggruppati sotto una specifica unità di codice eseguibile; in questo caso, il codice di un modulo residente.

La funzione InitResident usa per inizializzare un modulo residente la struttura Resident e la lista di segmenti a esso associate. Una volta inizializzato, il modulo residente rimane pronto in RAM per essere eseguito in qualsiasi momento senza dover essere caricato dal disco.

Un esame più accurato delle attività svolte dalla funzione InitResident permette di comprendere più a fondo il meccanismo dei moduli residenti. Quando InitResident ottiene il controllo, la prima operazione che esegue è verificare se nel parametro rt_Flags della struttura Resident il flag RTF_AUTOINIT è impostato o azzerato.

Se il flag RTF_AUTOINIT risulta azzerato, la funzione non fa altro che cedere il controllo all'indirizzo di memoria individuato dal parametro rt_Init della struttura Resident. Questo parametro deve quindi contenere l'indirizzo di una routine d'inizializzazione, alla quale InitResident delega completamente l'inizializzazione del modulo residente. Quando questa routine d'inizializzazione ha completato il suo lavoro, InitResident restituisce il controllo.

Se invece il flag RTF_AUTOINIT risulta impostato, InitResident suppone che il parametro rt_Init contenga l'indirizzo di una serie di quattro long word, e che i dati contenuti in queste locazioni siano i parametri della funzione MakeLibrary che deve chiamare. Adottando la terminologia indicata nella spiegazione della funzione MakeLibrary, InitResident ipotizza che all'indirizzo indicato da rt_Init sia presente l'argomento dataSize, seguito da funcInit, structInit, libInit; per quanto invece riguarda l'argomento segList di MakeLibrary, InitResident lo identifica con il valore che ha ricevuto come secondo argomento della chiamata. Chiamando MakeLibrary, InitResident trasforma il modulo residente in una libreria software.

Successivamente, se la funzione MakeLibrary restituisce un valore diverso da zero, cioè l'indirizzo base della libreria, InitResident chiama rispettivamente AddDevice, AddLibrary oppure AddResource, a seconda che il parametro rt_Type della struttura Resident contenga il valore NT_DEVICE, NT_LIBRARY oppure NT_RESOURCE.

InitResident può anche servire per installare in memoria una libreria o un dispositivo. Il seguente programmino dimostrativo in C mostra l'impiego della funzione InitResident per installare in memoria la libreria Version.

```
#include <exec/types.h>
#include <exec/resident.h>
```

```
main()
{
     ULONG segList, codeloc;
     struct Resident *resident;

     segList = (ULONG)LoadSeg("LIBS:version.library");
     if (segList == NULL)
     {
          puts ("Non trovo la libreria");
          exit (0);
     }
     codeloc = segList << 2;
     /* trasforma il BPTR restituito da LoadSeg in un indirizzo reale */
     resident = (struct Resident *)(codeloc + 8);
     /* vedere anche il Manuale dell'AmigaDOS, pagina 326, paragrafo 3.5 */
     InitResident (resident, segList);
}
```

Ovviamente perché la funzione InitResident possa essere impiegata in questo modo occorre che la libreria o il dispositivo siano strutturati secondo un particolare formato. L'hunk di codice iniziale della libreria o del dispositivo deve iniziare con le istruzioni Assembly "MOVEQ #0,D0; RTS;" (in esadecimale 0x70004E75), in modo che se si cerca di mandarlo in esecuzione da CLI come un qualunque comando, restituisca solo un codice d'errore. Queste istruzioni Assembly devono essere seguite da una struttura Resident opportunamente inizializzata, nella quale il parametro rt_Flags ha il flag RTF_AUTOINIT impostato a 1, e il parametro rt_Init contiene l'indirizzo delle quattro long word già descritte.

InitSemaphore

Sintassi di chiamata della funzione

```
InitSemaphore (signalSemaphore)
               A0
```

Scopo della funzione

Questa funzione inizializza opportunamente la struttura SignalSemaphore che il task ha provveduto ad allocare per creare un semaforo. La funzione non alloca memoria; inizializza soltanto i puntatori di lista e i contatori del semaforo.

Argomenti della funzione

signalSemaphore Indirizzo della struttura SignalSemaphore da inizializzare; tutti i suoi parametri devono essere a zero prima di chiamare InitSemaphore.

Discussione

Nella libreria Exec ci sono sette funzioni che riguardano i semafori di segnalazione e due funzioni che riguardano la lista dei semafori di segnalazione. Queste funzioni sono elencate nel corso della spiegazione di AddSemaphore, dove viene anche illustrata la struttura SignalSemaphore. La struttura SignalSemaphore possiede due parametri contatori e due parametri di collegamento. I parametri contatori sono ss_NestCount e ss_QueueCount. I parametri di collegamento sono ss_Link e ss_WaitQueue. Questi sono i quattro parametri inizializzati dalla funzione InitSemaphore.

Ci sono due modi per impostare la struttura SignalSemaphore. Si può provvedere "manualmente", cioè impiegando un insieme di istruzioni di assegnazione dei parametri di struttura, oppure si può usare la funzione InitSemaphore. Se si decide di seguire quest'ultima via, si dovrebbe chiamare la funzione prima di usare la struttura SignalSemaphore con qualsiasi altra funzione di gestione dei semafori di segnalazione. Si noti che InitSemaphore non inizializza il parametro ln_Pri della sotto-struttura Node della struttura SignalSemaphore. Questo compito è affidato al task, che deve stabilire la posizione del semaforo nella lista dei semafori.

I semafori nel sistema Amiga

Una delle più importanti migliorie apportate alla libreria Exec è la possibilità di definire e utilizzare i semafori. I semafori costituiscono un modo per impedire o permettere l'accesso da parte di un gruppo di task a un insieme di strutture dati che possono essere condivise da più task. La necessità di regolare gli accessi a particolari aree di memoria nasce dal fatto che il sistema operativo dell'Amiga è multitasking, e quindi consente a diversi task di accedere "simultaneamente" alle stesse risorse senza restrizioni. I semafori di segnalazione sono gli strumenti che permettono a un task di assicurarsi l'accesso esclusivo a un'area di memoria senza dover bloccare l'attività multitasking del sistema. I semafori di segnalazione vengono usati internamente dalle librerie Graphics, Layers e Intuition per permettere o vietare a diversi task in mutua competizione di disegnare simultaneamente nelle bitmap condivise.

In un sistema multitasking come l'Amiga, un task che accede a una particolare area di memoria condivisa con il sistema e con altri task non può ritenere che i dati in essa presenti siano sotto il suo completo controllo per tutto

il tempo che dura l'accesso. Infatti, mentre il task sta leggendo quei dati può verificarsi uno scambio di controllo che manda in esecuzione un altro task, il quale potrebbe essere predisposto per accedere in scrittura alla stessa area di memoria condivisa. Quando un nuovo scambio di controllo della CPU riattiva il primo task, questo continua ad accedere all'area dati condivisa come se niente fosse accaduto, ma i dati in essa presenti non sono più quelli che erano presenti quando il task aveva iniziato l'accesso.

Un altro esempio. A ogni lista di sistema dei chunk di memoria liberi presenti in una particolare area RAM (come la chip RAM), dovrebbe sempre accedere uno e un solo task per volta. Se un task altera una lista di memoria dopo che un altro task ha iniziato a esaminarla (e prima che abbia concluso), il task "invasore" produce confusione nel task che sta consultando la lista, causando così una situazione anomala nella gestione della memoria.

I task hanno a disposizione diversi modi per proteggere un'area RAM alla quale intendono accedere. Il primo consiste nell'usare le routine di sistema Forbid e Permit, le quali rispettivamente sospendono la gestione multitasking dei task (ma non l'attività degli interrupt) e la riattivano. Racchiudendo i codici del proprio task fra un Forbid e un Permit si ha la certezza che durante l'esecuzione di quei codici nessun altro task può ottenere il controllo della CPU. Questo schema di azione assume il seguente aspetto:

```
Forbid()
/* L'insieme delle istruzioni che seguono possono accedere a un'area condivisa
   di memoria senza temere che vi accedano anche altri task */
Permit()
```

Lo svantaggio di questo metodo consiste nel fatto che nel periodo di tempo che trascorre dall'esecuzione della funzione Forbid all'esecuzione della funzione Permit viene completamente sospesa la gestione multitasking. Fra tutti i task che non ricevono più il controllo, probabilmente sono pochissimi quelli pericolosi per l'integrità dell'area di memoria alla quale si desidera accedere, mentre la maggior parte rimangono sospesi inutilmente. In pratica, l'uso della coppia di funzioni Forbid e Permit in una macchina multitasking come l'Amiga è una pratica molto poco efficiente, che vanifica lo sforzo di creare un sistema operativo multitasking.

L'alternativa è l'uso dei semafori. Questo secondo metodo di protezione delle aree di memoria è più efficiente perché non sospende la gestione multitasking della macchina, ma richiede che il task si adegui a un insieme di convenzioni. In pratica, prima di accedere a un'area di memoria condivisa il task deve richiederne l'accesso; solo quando lo ottiene può accedere all'area con la sicurezza che i dati in essa presenti non saranno modificati da altri task. Tutti gli altri task che richiedono l'accesso vengono sospesi oppure ricevono una condizione d'errore (a seconda di come hanno formulato la richiesta). In questo modo, i task che non tentano di accedere a quell'area di memoria non subiscono alcun genere d'interdizione nella loro esecuzione e quindi la gestione multitasking non viene sospesa. La mutua esclusione avviene soltanto fra i task che tentano di accedere alla stessa area di memoria condivisa e protetta da un semaforo.

Due sono i tipi di semafori disponibili nel sistema Exec: i semafori di segnalazione e i semafori basati sui messaggi. Nove funzioni dell'Exec riguardano i semafori di segnalazione, mentre due funzioni (Procure e Vacate) riguardano i semafori basati su messaggi.

I semafori di segnalazione usano le strutture SemaphoreRequest e SignalSemaphore, e non fanno uso della possibilità di scambiare messaggi offerta dal sistema Exec; ciò significa che non hanno alcun rapporto con le strutture MsgPort e Message. I semafori di segnalazione consentono di proteggere un'area di memoria da accessi intrecciati e non coordinati di diversi task, in modo che per un certo periodo solo un task vi possa accedere (l'accesso diventa esclusivo).

I semafori basati su messaggi, invece, usano la struttura Semaphore e la possibilità di scambiare messaggi offerta dal sistema Exec. La struttura Semaphore contiene una sotto-struttura MsgPort nella quale si accumulano i messaggi relativi ai task che tentano di ottenere il semaforo quando non è disponibile, cioè quando un task lo detiene. Per ottenere il semaforo, i task devono chiamare la funzione Procure, e nella chiamata devono indicare come secondo argomento l'indirizzo di un messaggio; se Procure rileva che il semaforo non è libero, accoda quel messaggio alla message port del semaforo. Non appena il semaforo viene liberato dal task che lo detiene (attraverso la funzione Vacate), il sistema provvede a estrarre il primo messaggio in coda alla relativa message port e a rispedirlo al mittente, il quale ricevendolo saprà che possiede il semaforo e può quindi accedere all'area dati protetta. Da questa semplice descrizione dei semafori basati su messaggi si possono fare alcune considerazioni. Innanzitutto, consentono a un task di richiedere contemporaneamente diversi semafori, e non uno solo come accade con i semafori di segnalazione; il task può infatti chiamare la funzione Procure più volte, e se nessuna delle tre chiamate sortisce immediato successo, può entrare in attesa che alla sua reply port giunga uno dei tre messaggi indicati nelle chiamate alla funzione Procure. Quando il primo messaggio arriva, significa che il corrispondente semaforo si è liberato ed è diventato di proprietà del task. Rispetto ai semafori di segnalazione, i semafori basati su messaggi offrono un sistema per le richieste di accesso che potremmo definire asincrono, in quanto il task non entra mai in attesa.

L'uso dei semafori di segnalazione all'interno dei task è molto simile all'uso delle funzioni di sistema Forbid() e Permit(); ci si limita a isolare l'insieme delle istruzioni protette chiamando ObtainSemaphore e poi ReleaseSemaphore, come nell'esempio seguente:

```
ObtainSemaphore()
/* L'insieme delle istruzioni che seguono possono accedere a un'area di memoria
   condivisa senza temere che vi accedano anche altri task */
ReleaseSemaphore()
```

Se invece si impiegano i semafori basati sui messaggi, occorre racchiudere i codici tra le funzioni Procure e Vacate, con la differenza però che se Procure non ha successo il task non entra in attesa del messaggio indicato nella chiamata, ma continua l'elaborazione.

InitStruct

Sintassi di chiamata della funzione

InitStruct (initTable, memBlock, size)
A1 A2 DØ: Ø-15

Scopo della funzione

Questa funzione serve per inizializzare un'area di memoria seguendo le indicazioni contenute in un'opportuna tavola d'inizializzazione. L'area di memoria da inizializzare dev'essere preventivamente allocata dal task, il quale ne indica l'indirizzo come secondo argomento della chiamata. Una tavola d'inizializzazione creata dal task è lo strumento per indicare a InitStruct come inizializzare l'area di memoria. InitStruct consente per esempio d'inizializzare una struttura di dati partendo da un insieme di dati non strutturato che si trova altrove nella memoria. Quando la struttura è stata inizializzata secondo le indicazioni contenute nella tavola d'inizializzazione, il task può impiegarla come se l'avesse inizializzata parametro per parametro. Generalmente questa funzione viene impiegata dai programmatori che creano i task direttamente in Assembly.

Argomenti della funzione

initTable Indirizzo della tavola d'inizializzazione. Questa tavola deve contenere le indicazioni che permettono a InitStruct d'inizializzare la struttura puntata dall'argomento memBlock, e deve iniziare in memoria a un indirizzo pari.

memBlock Indirizzo del blocco di memoria che una volta elaborato dalla funzione costituirà la struttura dati inizializzata. Se nell'argomento size viene indicato un valore diverso da zero, l'indirizzo di questo blocco di memoria dev'essere pari.

size Dimensione in byte del blocco di memoria da inizializzare. Se questo argomento è diverso da zero, InitStruct azzera il blocco di memoria prima di inizializzarlo secondo quanto indicato nella tavola

individuata da initTable. Prima di venire usato, il valore indicato nell'argomento size viene arrotondato per difetto a un numero pari; per evitare il troncamento, è opportuno che size indichi sempre un numero pari.

Discussione

InitStruct è l'unica funzione che consente ai task d'inizializzare un'area di memoria mantenendo l'esatto e diretto controllo dell'inizializzazione. Infatti, funzioni come MakeLibrary, OpenLibrary, OpenDevice e OpenResource, inizializzano strutture di dati secondo le esigenze del sistema, in modo che queste strutture s'integrino in modo appropriato con le altre strutture e funzioni del sistema (molto spesso al loro interno impiegano la funzione InitStruct). Quando il task chiama una di queste funzioni è perché si affida al sistema per i valori iniziali dei parametri della struttura, e quindi non detiene un completo controllo sull'inizializzazione. Talvolta però c'è l'esigenza d'inizializzare strutture di dati per le quali il sistema non offre alcuna funzione dedicata. Generalmente si tratta di strutture definite dal task o strutture di sistema che non sono soggette a particolari vincoli d'inizializzazione. Per queste strutture i task possono quindi impiegare la funzione InitStruct.

La complessità e la flessibilità della funzione InitStruct non devono intimidire. In pratica, InitStruct legge dalla tavola d'inizializzazione una serie di comandi che indicano dove (all'interno dell'area di memoria da inizializzare) devono copiare i dati.

Prima di chiamare InitStruct, il task deve decidere se far azzerare dalla funzione l'area di memoria prima dell'inizializzazione, e deve allocare memoria sufficiente per contenere la tavola d'inizializzazione. In seguito deve inserire tutti i comandi e i dati di cui ha bisogno nella tavola d'inizializzazione. A questo punto può chiamare la funzione InitStruct per fare in modo che l'area di memoria venga inizializzata secondo quanto ha indicato nella tavola. Dopo che la funzione InitStruct ha restituito il controllo, la tavola d'inizializzazione non serve più e il task può anche liberare la memoria da essa occupata.

Se non si conosce a priori il numero di byte occupato dall'area di memoria da inizializzare, oppure non si desidera che venga azzerata, si deve indicare il valore zero nell'argomento size della chiamata alla funzione InitStruct.

La tavola d'inizializzazione che il task deve indicare nella chiamata è composta da una serie di comandi. Ogni comando inizia con un byte-comando seguito eventualmente da altri byte: l'insieme di byte-comando più i byte che lo seguono viene chiamato *definizione completa del comando*. Il primo byte-comando a zero indica alla funzione InitStruct la fine della tavola d'inizializzazione. I byte-comando devono sempre essere disposti a indirizzi pari di memoria, eventualmente aggiungendo un byte alla definizione completa del comando precedente (la funzione lo ignora automaticamente). Gli otto bit del byte comando sono divisi in tre campi che indicano: (1) come la funzione deve calcolare l'indirizzo destinazione all'interno dell'area di memoria

a partire dal quale devono essere copiati i dati associati al comando, e come deve copiarli; (2) il tipo di dati che deve copiare (byte, word o long word); (3) il numero di locazioni destinazione che il comando deve inizializzare nell'area di memoria memBlock. Quest'ultima informazione viene impiegata in due modi a seconda del tipo di comando. Se il comando prevede che venga copiato lo stesso dato per *n* volte, questa informazione indica il numero *n* − *1* di volte che il dato sorgente dev'essere copiato a partire dall'indirizzo destinazione, alrtimenti indica il numero *n* − *1* di dati da copiare di seguito nella destinazione.

A parte questi due modi per copiare i dati associati a ogni comando, i comandi disponibili consentono di effettuare le seguenti operazioni:

- possono copiare i dati in specifiche locazioni di memoria ottenute sommando un offset all'indirizzo base dell'area di memoria da inizializzare. L'offset è parte integrante della definizione completa del comando.

- Possono copiare i dati all'interno dell'area da inizializzare a partire dalla locazione di memoria successiva a quella inizializzata per ultima dall'ultimo comando eseguito. In questo secondo caso, nella definizione completa del comando non compaiono offset.

Ogni comando della tavola d'inizializzazione occupa un byte, il byte-comando. Gli otto bit di questo byte vengono suddivisi in tre gruppi nel modo seguente:

ddssnnnn

e forniscono alla funzione le seguenti informazioni:

(dd) sono i bit di destinazione. Indicano alla funzione come dev'essere calcolato l'indirizzo destinazione a partire dal quale devono essere trasferiti i dati, e come devono essere copiati i dati (un dato copiato un certo numero di volte o diversi dati da copiare una volta sola). I bit di destinazione (dd) possono assumere i seguenti valori e relativi significati:
00 Indica di memorizzare i dati a partire dalla locazione destinazione successiva dell'area da inizializzare. Con questa combinazione di bit, la funzione prevede che (nnnn) contenga un numero, *n*, che indica di copiare in blocco gli *n* + *1* dati del comando (in questo modo viene copiato un insieme di dati una sola volta). Nella definizione completa del comando il byte comando è seguito dai dati (per i quali valgono le regole di allineamento alle word specificate nella spiegazione dei bit sorgente, ss).
01 Indica di memorizzare i dati a partire dalla locazione destinazione successiva dell'area da inizializzare. Con questa combinazione di bit, la funzione prevede che (nnnn) contenga un numero, *n*, che indica di copiare *n* + *1* volte il dato del comando (in questo modo viene copiato

un solo dato diverse volte). Nella definizione completa del comando, il byte-comando è seguito dai dati (per i quali valgono le regole di allineamento alle word specificate nella spiegazione dei bit sorgente, ss).

10 Indica che il byte che segue il byte-comando contiene l'offset che dev'essere sommato all'indirizzo base dell'area da inizializzare per ottenere l'indirizzo destinazione; i dati devono essere memorizzati a partire da questo indirizzo di memoria. L'offset è quindi un numero espresso su 8 bit. Con questa combinazione di bit, la funzione prevede che (nnnn) contenga un numero, *n*, che indica di copiare in blocco gli *n + 1* dati del comando (in questo modo viene copiato un insieme di dati una sola volta). Nella definizione completa del comando, il byte comando è seguito nella tavola dal byte contenente l'offset, che a sua volta è seguito dai dati (per i quali valgono le regole di allineamento alle word specificate nella spiegazione dei bit sorgente, ss).

11 Indica che i tre byte che nella tavola seguono il byte comando contengono l'offset che dev'essere sommato all'indirizzo base dell'area da inizializzare per ottenere l'indirizzo destinazione; i dati devono essere memorizzati a partire da questo indirizzo di memoria. L'offset è quindi un numero espresso su 24 bit. Con questa combinazione di bit, la funzione prevede che (nnnn) contenga un numero, *n*, che indica di copiare in blocco gli *n + 1* dati del comando (in questo modo viene copiato un insieme di dati una sola volta). Nella definizione completa del comando, il byte comando è seguito nella tavola dai tre byte che contengono l'offset, che a loro volta sono seguiti dai dati (per i quali valgono le regole di allineamento alle word specificate nella spiegazione dei bit sorgente, ss).

Si noti che nei casi (dd) = 10 e (dd) = 11, l'insieme di byte composto dal byte-comando e dai byte di offset (un byte di offset nel primo caso e tre byte di offset nel secondo caso) è un multiplo esatto di due. Questo garantisce che i dati che seguono saranno sempre e comunque allineati a indirizzi pari, dal momento che i byte-comando, i primi byte di ogni comando, devono sempre trovarsi a indirizzi di memoria pari. Lo stesso non si può dire nei casi (dd) = 00 e (dd) = 01, dal momento che lo spazio utile per i dati inizia subito dopo il byte comando.

(ss) sono i bit sorgente. Indicano alla funzione il tipo di dati contenuti nella definizione completa del comando, cioè il tipo di dati da copiare nell'area di memoria da inizializzare a partire dall'indirizzo destinazione. I dati possono essere byte, word o long word. Se si tratta di byte non ci sono restrizioni riguardo alla locazione iniziale nella tavola d'inizializzazione; questo significa che il primo dato può anche trovarsi subito dopo il byte-comando (cioè a un indirizzo di memoria dispari). Per le word e le long word, invece, occorre che i dati siano sempre a indirizzi di memoria pari, cioè allineati alle word. Nel casi (dd) = 00 e (dd) = 01 questo significa che fra il byte-comando e i dati dev'essere presente un byte di allineamento, a cui ovviamente la funzione non bada. I bit sorgente (ss) possono assumere i seguenti valori e relativi significati.

00 I dati che devono essere copiati nell'area di memoria da inizializzare sono long word. Pertanto, devono trovarsi nella tavola d'inizializzazione a indirizzi pari di memoria.
01 I dati che devono essere copiati nell'area di memoria da inizializzare sono word. Pertanto, devono trovarsi nella tavola d'inizializzazione a indirizzi pari di memoria.
10 I dati che devono essere copiati nell'area di memoria da inizializzare sono byte. Pertanto, possono trovarsi nella tavola sia a indirizzi pari che a indirizzi dispari di memoria.
11 Questa combinazione di bit sorgente (ss) è da evitare, in quanto causa l'esecuzione della funzione Alert per la visualizzazione di un alert di sistema con il codice di alert AN_InitAPtr.

(nnnn) sono i bit di conteggio delle copie da effettuare. Il task deve sempre memorizzare in questo nibble del byte-comando un numero inferiore di una unità rispetto al numero di copie che la funzione deve eseguire quando incontra il comando. Nel caso (dd) = 01, questo contatore viene utilizzato come numero di ripetizioni dello stesso dato a partire dall'indirizzo destinazione dell'area da inizializzare. In tutti gli altri casi viene considerato il numero di dati (e non di byte) che devono essere copiati nell'area da inizializzare a partire dall'indirizzo destinazione.

Riassumendo, i byte-comando della tavola d'inizializzazione devono sempre trovarsi a indirizzi pari di memoria. I due tipi di offset previsti dalla funzione (da 8 e da 24 bit) sono sempre relativi all'indirizzo base dell'area di memoria da inizializzare, quello che il task indica come secondo argomento della funzione InitStruct. Il byte comando %00000000 indica alla funzione la fine della tavola d'inizializzazione. Si noti che un byte interamente a zero, secondo le convenzioni ora esposte, indicherebbe di copiare una longword all'indirizzo corrente. Per copiare effettivamente una long word, bisogna quindi ricorrere al comando %00010001 che copia due word.

La tavola d'inizializzazione che segue ha l'unico scopo di mostrare una buona varietà d'impieghi dei byte-comando nell'inizializzazione dei parametri di una struttura Task. Si è scelta la struttura Task in quanto ha un numero di parametri sufficiente a consentire la verifica di numerose combinazioni di bit del byte-comando. La tavola inizializza i parametri della struttura Task con valori del tutto privi di significato e con criteri volti solo a mostrare la codifica dei comandi in relazione al compito che svolgono. La tavola è redatta in codici Assembly sorgente. In corrispondenza di ogni dato, è indicato il parametro nel quale il dato verrà copiato (quando più parametri ricevono lo stesso dato, sono separati da un trattino).

```
_tavola:
        dc.b        %10100001   ;(dd)10 (ss)10 (nnnn)0001
        dc.b        $08         ;Offset (8 bit) a tc_Node.ln_Type
        dc.b        $01         ;tc_Node.ln_Type (NT_TASK)
        dc.b        $64         ;tc_Node.ln_Pri
        dc.b        %01000000   ;(dd)01 (ss)00 (nnnn)0000
```

```
        dc.b    Ø               ;Byte di allineamento
        dc.l    nome            ;tc_Node.ln_Name
        dc.b    %ØØ1ØØØØ1       ;(dd)ØØ (ss)1Ø (nnnn)ØØØ1
        dc.b    $11             ;tc_Flags (TF_PROCTIME | TF_STACKCHK)
        dc.b    $1              ;tc_State (TS_ADDED)
        dc.b    Ø               ;Byte di allineamento
        dc.b    %Ø11ØØØØ1       ;(dd)Ø1 (ss)1Ø (nnnn)ØØØ1
        dc.b    $35             ;tc_IDNestCnt - tc_TDNestCnt
        dc.b    %Ø1ØØØØ11       ;(dd)Ø1 (ss)ØØ (nnnn)ØØ11
        dc.b    Ø               ;Byte di allineamento
        dc.l    $55554444       ;tc_SigAlloc - tc_SigExcept
        dc.b    %Ø1Ø1ØØØ1       ;(dd)Ø1 (ss)Ø1 (nnnn)ØØØ1
        dc.b    Ø               ;byte di allineamento
        dc.w    $AAAA           ;tc_TrapAlloc - tc_TrapAble
        dc.b    %ØØØØØ11Ø       ;(dd)ØØ (ss)ØØ (nnnn)Ø11Ø
        dc.b    Ø               ;byte di allineamento
        dc.l    $Ø1             ;tc_ExceptData
        dc.l    $Ø2             ;tc_ExceptCode
        dc.l    $Ø3             :tc_TrapData
        dc.l    $Ø4             ;tc_TrapCode
        dc.l    $Ø5             ;tc_SPReg
        dc.l    $Ø6             ;tc_SPLower
        dc.l    $Ø7             ;tc_SPUpper
        dc.b    %11ØØØØ1Ø       ;(dd)11 (ss)ØØ (nnnn)ØØ1Ø
        dc.b    $ØØ             ;offset (24 bit) a tc_MemEntry.lh_Head
        dc.b    $ØØ
        dc.b    $4A
        dc.l    $11111111       ;tc_MemEntry.lh_Head
        dc.l    $22222222       ;tc_MemEntry.lh_Tail
        dc.l    $33333333       ;tc_MemEntry.lh_TailPred
        dc.b    %ØØ1ØØØØØ       ;(dd)ØØ (ss)1Ø (nnnn)ØØØØ
        dc.b    $EE             ;tc_MemEntry.lh_Type
        dc.b    %ØØØØØØØØ       ;Fine della tavola
nome:
        dc.b    'Struttura Task inutilizzabile',Ø
```

Con questa tavola d'inizializzazione, gli unici parametri della struttura Task che vengono ignorati sono tc_Node.ln_Succ, tc_Node.ln_Pred, tc_Switch, tc_Launch, tc_MemEntry.l_pad, e tc_UserData.

Quando InitResident viene impiegato dall'interno di una routine in Assembly, si tenga presente che la funzione distrugge il contenuto dei registri A0, A1, D0, D1.

Insert

Sintassi di chiamata della funzione

Insert (list, node, listNode)
AØ A1 A2

Scopo della funzione

Questa funzione inserisce una struttura Node in una lista a doppia concatenazione *dopo* il nodo indicato come terzo argomento. Si può inserire la struttura Node in testa alla lista indicando il valore zero come terzo argomento.

Argomenti della funzione

list — Indirizzo della struttura List alla quale dev'essere aggiunto il nodo.

node — Indirizzo della struttura Node che si desidera inserire nella lista.

listNode — Indirizzo della struttura Node della lista dopo la quale si vuole inserire il nodo individuato dall'argomento node. Se vale zero, il nodo viene inserito in testa alla lista.

Discussione

La libreria Exec ha diverse funzioni destinate a operare sui nodi e sulle liste doppie. Si veda la spiegazione della funzione AddHead per una trattazione completa di queste funzioni.

Le liste doppie, dette anche liste a doppia concatenazione, sono costituite da nodi che possiedono un puntatore al nodo successivo e un puntatore al nodo precedente. Questi due puntatori instaurano per ogni nodo un doppio legame, e consentono di leggere la lista in entrambe le direzioni. La doppia concatenazione consente anche di aggiungere e togliere nodi da qualsiasi punto della lista in maniera molto efficiente. Infatti, volendo aggiungere o togliere un nodo che si trova dopo un altro di indirizzo noto, non è necessario

leggere la lista dall'inizio.

Nell'Amiga, le liste doppie vengono usate spessissimo, sia dal sistema che dai task. Ogni lista viene gestita attraverso una struttura List che mantiene l'organizzazione dei suoi nodi.

Per inserire una nuova struttura Node nella lista, in qualsiasi posizione, occorre la funzione Insert. Questa funzione inserisce la struttura Node subito dopo la struttura Node indicata dal task come terzo argomento. Si noti che se l'argomento listNode punta alla struttura Node corrispondente al parametro lh_Tail della struttura List, il nuovo nodo viene inserito in fondo alla lista. La funzione Insert non bada alla priorità del nodo per inserirlo nella lista, e quindi non si deve utilizzarla per creare liste ordinate secondo la priorità dei nodi. A questo scopo esiste la funzione Enqueue.

MakeFunctions

Sintassi di chiamata della funzione

```
tableSize = MakeFunctions (target, funcArray, funcDispBase)
D0                         A0      A1         A2
```

Scopo della funzione

Questa funzione serve a costruire una tavola di salto del tipo di quella usata dalle risorse, dalle librerie e dai dispositivi.

La tavola può essere situata in una qualunque posizione della memoria e può essere usata sia per inizializzare una nuova tavola sia per rimpiazzarne del tutto o in parte una vecchia.

Argomenti della funzione

target — L'indirizzo di memoria a partire dal quale, a offset negativi, verrà costruita la tavola specificata. In genere si tratta dell'indirizzo base della libreria in questione.

funcArray — Puntatore a un array di puntatori a funzioni o a un array di word che rappresentano offset relativi all'indirizzo base contenuto in funcDispBase (se è diverso da zero). La fine della tavola va indicata

tramite il valore -1 (una longword nel primo caso, una word nel secondo).

funcDispBase Puntatore che rappresenta la base per gli offset contenuti nell'array puntato da funcArray. Se vale zero, l'array deve invece essere costituito da puntatori alle funzioni da inserire nella tavola.

Discussione

Ci sono otto funzioni che riguardano le librerie nel sistema Amiga: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Si vedano anche le spiegazioni relative a queste funzioni.

Se si desidera inizializzare o modificare una tavola di salto relativa a una libreria, a un dispositivo o a una risorsa, invece di ricorrere a una serie di chiamate alla funzione SetFunction, è possibile ricorrere a questa funzione. A partire dall'indirizzo base specificato viene collocata, a offset negativi, una serie di istruzioni di salto del tipo JMP $XXXXXXXX alle funzioni descritte.

Dette funzioni possono essere specificate in due modi diversi:

- preparando in memoria un array contenente i loro indirizzi e passando l'indirizzo del primo elemento dell'array nel parametro funcArray; in questo caso l'argomento funcDispBase dev'essere posto a zero. L'array va concluso con il valore -1 ($FFFFFFFF).

- Preparando in memoria un array di word il cui valore indica l'offset di ogni funzione dall'argomento funcDispBase, che dev'essere specificato. Anche in questo caso l'array dev'essere concluso dal valore -1 ($FFFF).

MakeLibrary

Sintassi di chiamata della funzione

```
library = MakeLibrary (funcInit, structInit, libInit, dataSize, segList)
D0                     A0      A1       A2      D0       D1
```

Scopo della funzione

Questa funzione alloca memoria sufficiente per creare il modulo di controllo di una libreria, composto nell'ordine da una tavola di vettori di salto alle funzioni della libreria, da una struttura Library e da un'area dati. Se si richiede l'inizializzazione automatica della struttura Library (si veda la spiegazione dell'argomento structInit), la libreria, grazie al modulo di controllo creato da MakeLibrary, può essere aggiunta al sistema tramite la funzione AddDevice, AddLibrary o AddResource, a seconda che si tratti della libreria di un dispositivo, di una libreria condivisa o di una risorsa (per semplicità, in questa spiegazione citiamo sempre e soltanto le funzioni AddLibrary, OpenLibrary, CloseLibrary e RemLibrary, ma si ricordi che le stesse considerazioni valgono anche per le librerie dei dispositivi e delle risorse). Si tenga conto che prima di chiamare questa funzione, i codici della libreria devono già trovarsi in memoria, insieme a una serie di dati che servono per trasformare questi codici in una libreria coerente con il formato previsto per esse dall'Amiga.

Oltre che creare il modulo di controllo, MakeLibrary manda in esecuzione la routine d'inizializzazione eventualmente indicata dal task nella chiamata della funzione.

Se non si verificano problemi, la funzione MakeLibrary restituisce l'indirizzo della struttura Library che ha creato, cioè l'indirizzo base della libreria. Questo indirizzo dev'essere indicato ogni volta che si chiama una funzione di gestione delle librerie; per esempio quando si chiama la funzione AddLibrary per aggiungere la libreria al sistema, oppure la funzione CloseLibrary per chiuderla. Si tenga conto che la funzione per default inizializza solo i parametri Lib_NegSize e Lib_PosSize della struttura Library, mentre l'inizializzazione degli altri e dell'area dati è a carico del task. Il task ha tre sistemi a disposizione: (1) inizializzare gli altri parametri della struttura Library e l'area dati quando riottiene il controllo da MakeLibrary; (2) indicare a MakeLibrary una tavola d'inizializzazione e quindi richiedere l'esecuzione automatica della funzione InitStruct dall'interno della stessa MakeLibrary; (3) delegare l'inizializzazione della struttura Library e dell'area dati all'eventuale routine d'inizializzazione della libreria.

Argomenti della funzione

funcInit Questo argomento dev'essere l'indirizzo della tavola che elenca i riferimenti a tutte le funzioni della libreria. MakeLibrary utilizza il contenuto di questa tavola per creare in memoria i vettori di salto alle funzioni della libreria, cioè l'elenco d'istruzioni jmp $XXXXXX che precedono in sequenza l'indirizzo base della libreria (ogni vettore occupa sei byte). La tavola indicata da funcInit può essere strutturata in due

modi. Se sono noti gli indirizzi assoluti delle funzioni della libreria, può essere costituita da una serie di long word, ognuna delle quali contiene l'indirizzo assoluto di una funzione della libreria. La prima long word contenente il valore -1 indica la fine della tavola. Se invece le funzioni della libreria sono entro 64K dall'indirizzo della tavola funcInit, possiamo riferirci a esse con offset da una word, offset che devono essere necessariamente relativi all'indirizzo indicato dal task nell'argomento funcInit. In questo caso, la prima word della tavola puntata da funcInit deve contenere il valore -1 (indica a MakeLibrary che seguono offset relativi all'indirizzo funcInit), seguita dagli offset di tutte le funzioni della libreria, con ogni offset che occupa una word. Anche in questo caso, la prima word contenente -1 indica la fine della tavola funcInit. Comunque, che venga usata una tavola d'indirizzi assoluti o una tavola di offset relativi a funcInit, la tavola dei vettori risultante dopo l'esecuzione di MakeLibrary è sempre composta da istruzioni jmp $XXXXXX, cioè da vettori di salto a indirizzi di memoria assoluti. Si tenga presente che i primi quattro riferimenti a funzioni della tavola che il task indica a MakeLibrary con l'argomento funcInit devono essere i riferimenti alle funzioni standard Open, Close, Expunge e ExtFunc che ogni libreria deve possedere.

structInit Indirizzo della tavola d'inizializzazione che MakeLibrary sottopone alla funzione InitStruct per inizializzare la struttura Library ed eventualmente l'area dati che la segue in memoria. Se questo argomento è nullo, la funzione InitStruct non viene chiamata. Questo argomento e la funzione InitStruct forniscono un modo per impostare alcuni parametri di gestione della libreria prelevando i valori da una tavola di dati, detta d'inizializzazione, conservata in un punto qualsiasi della memoria. La tavola d'inizializzazione può servire per esempio per inizializzare i parametri lib_Node.ln_Type, lib_Node.ln_Name, lib_Flags, lib_Version, lib_Revision e lib_IdString della struttura Library allocata da MakeLibrary, insieme ad altri parametri dell'area dati che segue la struttura Library. Si noti che l'uso della tavola d'inizializzazione è possibile solo perché i parametri che essa deve inizializzare sono localizzati per mezzo di offset relativi all'indirizzo base dell'area di memoria da preparare; nel caso della funzione MakeLibrary,

l'indirizzo base è ovviamente quello della struttura Library che gestirà la libreria. Si veda anche la spiegazione della funzione InitStruct e si tenga presente che gli unici due parametri della struttura Library che MakeLibrary inizializza, prima di chiamare eventualmente InitStruct, sono Lib_NegSize e Lib_PosSize.

libInit

Indirizzo della routine d'inizializzazione che si desidera venga chiamata da MakeLibrary dopo che ha completato tutti i suoi compiti, cioè dopo che ha allocato l'area di memoria contenente la tavola dei vettori di salto alle funzioni, seguita dalla struttura Library e dall'area dati, e dopo che ha creato i vettori di salto alle funzioni. Se questo argomento è nullo, non viene chiamata alcuna routine. Quando la routine d'inizializzazione riceve il controllo, nel registro D0 si trova l'indirizzo base della libreria, cioè della struttura Library, e nel registro A0 si trova l'indirizzo della lista dei segmenti. La routine può svolgere qualsiasi operazione d'inizializzazione della libreria. Generalmente svolge quei compiti che né la tavola d'inizializzazione puntata dall'argomento structInit, né la funzione MakeLibrary possono svolgere. Per esempio, potrebbe memorizzare l'indirizzo base della libreria (D0) e l'indirizzo della lista dei segmenti (A0) in due parametri dell'area dati, aprire la libreria DOS e memorizzarne l'indirizzo base in un altro parametro dell'area dati, e restituire il controllo. In seguito, le funzioni della libreria possono fare affidamento sul contenuto di questi parametri per svolgere i loro compiti. È di fondamentale importanza tener presente che se l'argomento libInit è diverso da zero, l'esito della funzione MakeLibrary, cioè il contenuto del registro D0 quando la funzione restituisce il controllo, è quello memorizzato dalla routine d'inizializzazione nel registro D0 prima di completarsi con l'istruzione RTS. Se tutto va per il verso giusto, il risultato della funzione MakeLibrary è l'indirizzo base della libreria; in caso contrario è l'indirizzo nullo. È imperativo che la routine d'inizializzazione indicata dal task si adegui a questa convenzione, perché MakeLibrary non interviene più su questo registro una volta che gli ha ceduto il controllo. Infine, nella creazione di questa routine si tenga conto che MakeLibrary, quando riceve il controllo, salva sullo stack i registri da D2 a D7 e da A2 ad A3, e provvede a ripristinarli quando la routine d'inizializzazione restituisce il controllo.

Pertanto, questi registri, insieme ai registri A0, A1, D0, D1, sono a disposizione della routine.

dataSize Dimensione in byte dell'area dati della libreria. Il task deve indicare in questo argomento la quantità di memoria necessaria per contenere oltre all'area dati anche la struttura Library che la precede in memoria. Generalmente, quando i programmatori scrivono i codici di una libreria creano una sovrastruttura apposita per definire e individuare simbolicamente tutti i parametri in essa previsti, e questa struttura contiene sempre come primo elemento la struttura Library. La dimensione da indicare nell'argomento dataSize è quella di questa sovrastruttura, comprensiva dell'iniziale struttura Library. Si veda la discussione della funzione AddLibrary per la definizione della struttura Library.

segList Indirizzo di una lista di segmenti di memoria dell'AmigaDOS. Questo indirizzo viene passato alla routine d'inizializzazione nel registro A0 per essere impiegato quando occorre liberare la memoria occupata dalla libreria. Dal momento che la funzione MakeLibrary non ne fa uso, se neanche la routine d'inizializzazione lo usa diventa del tutto inutile; in questi casi i task devono indicare nell'argomento segList l'indirizzo nullo.

Discussione

Ci sono otto funzioni dell'Exec che riguardano le librerie nel sistema Amiga: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Si vedano anche le spiegazioni relative a queste funzioni.

Nel sistema Amiga il termine "libreria" si riferisce sempre a una collezione di funzioni raggruppate in memoria sotto uno stesso nome. Il sistema prevede numerose librerie di funzioni in grado di adempiere a una grande quantità di compiti. Alcune sono contenute nella ROM, le librerie *residenti*, mentre altre, quelle di uso meno frequente, risiedono sul disco sistema per non affollare troppo la memoria di sistema, e vengono richiamate in memoria quando un task cerca di accedervi; queste vengono definite librerie *non-residenti*. L'organizzazione a librerie del sistema operativo dell'Amiga è molto flessibile e aperta; consente infatti ai programmatori e alle software house di creare le proprie librerie di funzioni e di aggiungerle al sistema, ampliando le prestazioni della macchina. Generalmente, i programmatori tendono a raggruppare in librerie le funzioni d'uso più frequente nei loro task.

Le librerie vengono poi suddivise in librerie dei dispositivi, librerie generiche e librerie delle risorse. Sostanzialmente, la struttura fisica di una libreria è sempre uguale, ma le varie categorie di librerie vengono mantenute in liste di sistema differenti; abbiamo infatti la DeviceList, la LibList e la ResourceList. Le librerie sono sempre identificate da un nome e da un numero di versione. Ma per selezionare la categoria alla quale appartiene la libreria che si desidera aprire non è sufficiente il nome; occorre anche l'appropriata funzione d'apertura della libreria. Abbiamo quindi la funzione OpenDevice per le librerie dei dispositivi, la funzione OpenLibrary per le librerie generiche, e la funzione OpenResource per le librerie delle risorse. Questi sono concetti di primaria importanza per comprendere il software sistema dell'Amiga.

La funzione MakeLibrary è indifferente alla categoria di una libreria, in quanto il suo compito è solo quello di dare forma alla libreria in memoria. In pratica, prima di chiamare MakeLibrary, in memoria risiedono una serie di funzioni e una tavola di vettori che le individua (quella il cui indirizzo dev'essere indicato dal task nell'argomento funcInit). Quando MakeLibrary restituisce il controllo, in memoria è presente anche un modulo di controllo della libreria, ovvero un'area di memoria occupata da tre sotto-moduli disposti in sequenza. Il sotto-modulo più importante è quello centrale, che contiene i parametri di gestione della libreria. Per riferirsi a questi parametri si usa la struttura Library, e pertanto diremo che il sotto-modulo centrale è una struttura Library. L'indirizzo del primo byte di questa struttura viene definito indirizzo base della libreria, ed è di fondamentale importanza per qualsiasi operazione si voglia effettuare con la libreria: per esempio, è l'indirizzo assoluto tramite il quale, per mezzo di offset, possiamo individuare tutti gli altri byte del modulo di controllo, a partire dai parametri della struttura Library.

Il sotto-modulo centrale è preceduto in memoria dalla tavola dei vettori di salto. Questi vettori sono stati creati da MakeLibrary sulla base di quanto indicato nella tavola funcInit, e sono composti da istruzioni di tipo jmp $XXXXXX; pertanto ogni vettore occupa sei byte. Per chiamare una funzione di una libreria, i task individuano il corrispondente vettore di salto tramite un offset relativo all'indirizzo base della libreria. La tipica istruzione Assembly per chiamare una funzione è jsr -XX(A6), nella quale -XX è l'offset al vettore della funzione, mentre nel registro A6 deve trovarsi l'indirizzo base della libreria.

Il sotto-modulo centrale è poi seguito dall'area dati, che le funzioni della libreria possono utilizzare per i propri scopi. Normalmente, quest'area contiene parametri di gestione della libreria non standard (quelli standard sono contenuti nella struttura Library) e condivisi da tutte le funzioni della libreria.

Grazie a MakeLibrary esiste ora in memoria una serie di funzioni organizzate secondo lo schema che l'Amiga prevede per le librerie condivise; la libreria può ora essere aggiunta al sistema tramite AddLibrary, AddResource o AddDevice, ma per automatizzare il processo d'installazione di una libreria nel sistema si può anche impiegare la funzione InitResident, la quale, sulla base di certi dati, chiama la funzione MakeLibrary e poi una delle tre funzioni d'installazione appena citate. Per maggiori dettagli si veda la descrizione di questa funzione.

La variabile lib_Flags

Il parametro lib_Flags della struttura Library è un insieme di flag che il programmatore deve conoscere per poter creare una libreria.

Se risulta impostato il flag LIBF_SUMMING, significa che un task sta effettuando il checksum, o controllo di somma, su questa libreria. Il checksum di una libreria viene avviato chiamando la funzione SumLibrary, la quale esegue il checksum e confronta il risultato con quello atteso; se sono diversi e il flag LIBF_CHANGED non risulta impostato viene causato un alert di sistema. Il task può svolgere questa operazione di tanto in tanto per assicurarsi che le librerie a cui accede siano integre.

Se il flag LIBF_CHANGED risulta impostato, significa che uno o più vettori di salto della tavola della libreria sono stati alterati. Questo flag viene usato dalla funzione SumLibrary. Ogni volta che una libreria viene cambiata, dovrebbe essere chiamata la funzione SumLibrary per ricalcolare il checksum e memorizzarne il valore nella libreria per futuri controlli di coerenza.

Se il flag LIBF_SUMUSED è impostato, significa che il progettista della libreria richiede al sistema di causare il crash della macchina qualora si verifichi una discrepanza fra il valore di checksum atteso e quello calcolato dalla funzione SumLibrary. Se si imposta questo flag e si modifica la libreria senza ricalcolare e memorizzare il nuovo checksum, le routine dell'Exec rileveranno l'incoerenza e produrranno il crash del sistema. Questo può essere evitato chiamando la funzione SumLibrary ogni volta dopo ogni aggiornamento di ogni libreria.

ObtainSemaphore

Sintassi di chiamata della funzione

ObtainSemaphore (signalSemaphore)
AØ

Scopo della funzione

Questa funzione consente al task l'accesso esclusivo a una struttura SignalSemaphore. Se la struttura SignalSemaphore indicata come argomento non è disponibile (cioè vi sta accedendo un altro task), ObtainSemaphore fa entrare il task in attesa e accoda la richiesta di accesso. Il task riotterrà il controllo quando tutte le richieste accodate prima della sua saranno state soddisfatte e il semaforo risulterà quindi libero. ObtainSemaphore non restituisce alcun valore.

Argomenti della funzione

signalSemaphore Indirizzo della struttura SignalSemaphore che costituisce il semaforo richiesto.

Discussione

Ci sono sette funzioni dell'Exec che riguardano i semafori di segnalazione e due funzioni che riguardano la lista di sistema dei semafori di segnalazione. Queste funzioni sono illustrate nella spiegazione di AddSemaphore.

I semafori di segnalazione possono essere ottenuti e rilasciati attraverso chiamate alle funzioni ObtainSemaphore, AttemptSemaphore e ReleaseSemaphore. Il task che detiene un semaforo può chiamare la funzione ObtainSemaphore più volte; l'effetto di ogni chiamata è l'incremento del valore contenuto nel parametro ss_NestCount della struttura SignalSemaphore. Perché il semaforo venga considerato libero, occorre che il contatore delle chiamate a ObtainSemaphore venga riportato a zero tramite altrettante chiamate a ReleaseSemaphore. Questo sistema consente di nidificare le richieste di accesso a un semaforo di cui già si dispone. Quando il contatore ritorna a zero, la successiva richiesta di accesso viene soddisfatta, e il task che l'ha inoltrata ottiene il controllo.

Due o più task possono trovarsi in competizione per accedere alla stessa struttura SignalSemaphore. Solo uno di essi la ottiene, mentre tutti quelli che hanno chiamato ObtainSemaphore entrano in attesa. Il sistema mantiene una lista di questi task usando il parametro ss_WaitQueue della struttura SignalSemaphore. Si noti che ss_WaitQueue è il nome di una sotto-struttura MinList contenuta nella struttura SignalSemaphore. Ogni volta che il parametro ss_NestCount giunge a zero, il sistema consulta questa lista di task in attesa per rilevare quale deve risvegliare per concedergli l'accesso alla struttura SignalSemaphore.

Si noti che i semafori di segnalazione sono molto diversi dai semafori basati su messaggi. I semafori di segnalazione richiedono meno tempo-CPU. specialmente se la struttura SignalSemaphore è libera, e inoltre richiedono meno operazioni di preparazione del semaforo (è sufficiente una chiamata a InitSemaphore).

I semafori basati su messaggi, invece, presentano il vantaggio di offrire un sistema asincrono per le richieste di accesso, che non forza quindi il task a entrare in attesa. Inoltre, permettono al task di trattare la liberazione del semaforo come un evento, e quindi di entrare in attesa anche di questo evento tramite la funzione Wait. Tuttavia non sono efficienti quanto i semafori di segnalazione e richiedono al task un maggior numero di operazioni per accedervi (la preparazione di un messaggio). Si veda la spiegazione della funzione InitSemaphore per una descrizione completa dei due tipi di semafori.

La struttura SemaphoreRequest

Quando la funzione ObtainSemaphore rileva che il semaforo indicato non è disponibile, alloca automaticamente memoria per una struttura SemaphoreRequest per accodare la richiesta di accesso a quel semaforo. La definizione della struttura SemaphoreRequest è la seguente:

```
struct SemaphoreRequest {
    struct MinNode sr_Link;
    struct Task *sr_Waiter;
};
```

Il parametro sr_Link è una sotto-struttura MinNode che serve per mantenere la struttura SemaphoreRequest all'interno della lista ss_WaitQueue che fa capo al semaforo di segnalazione, la lista che elenca le richieste di accesso a quel semaforo (la struttura MinNode è una forma abbreviata della struttura Node; essa è definita nel file INCLUDE nodes.h e viene usata con le liste semplici gestite tramite la struttura MinList). ObtainSemaphore inserisce la struttura SemaphoreRequest nella coda del semaforo al quale si desidera accedere; a mano a mano che il semaforo viene bloccato e rilasciato, le strutture SemaphoreRequest che si trovano in coda risalgono la lista ss_WaitQueue; quando una particolare struttura SemaphoreRequest giunge alla sommità e viene estratta, il corrispondente task riottiene il controllo. Prima d'inserire la struttura SemaphoreRequest in coda, la funzione ObtainSemaphore memorizza nel parametro sr_Waiter l'indirizzo della struttura Task che definisce il task, in modo che quando SemaphoreRequest giunge alla sommità e viene estratta, il sistema sappia qual è il task che deve riottenere il controllo.

ObtainSemaphoreList

Sintassi di chiamata della funzione

```
ObtainSemaphoreList (list)
                    A0
```

Scopo della funzione

Questa funzione tenta di ottenere simultaneamente accesso a tutti i semafori di segnalazione elencati nella lista indicata come argomento. L'uso della funzione ObtainSemaphoreList è preferibile al ripetuto uso della funzione ObtainSemaphore per i vari semafori della lista, in quanto evita che si creino

condizioni di attesa perpetue, nelle quali un task entra in attesa di ottenere uno dei semafori elencati, e il task che detiene quel semaforo entra in attesa di un altro semaforo detenuto dal primo.

Argomenti della funzione

list Indirizzo di una struttura List che elenca una serie di semafori di segnalazione. I semafori di segnalazione in questa lista sono collegati insieme attraverso la sotto-struttura Node ss_Link della struttura SignalSemaphore.

Discussione

Ci sono sette funzioni nell'Exec che riguardano i semafori di segnalazione e due funzioni che riguardano le liste dei semafori di segnalazione. Queste funzioni sono illustrate nella spiegazione di AddSemaphore.

ObtainSemaphoreList presuppone che in un dato istante non ci sia più di un task che richiede l'intera lista dei semafori. Siccome questa condizione non può essere sempre garantita, si dovrebbe specificare un più alto livello di blocco (per esempio un altro semaforo di segnalazione) per bloccare la struttura List dei semafori di segnalazione prima di effettuare la chiamata a ObtainSemaphoreList. Questo semaforo di livello superiore serve a prevenire la possibilità che un altro task alteri la lista dei semafori di segnalazione prima che la funzione ObtainSemaphoreList termini l'esecuzione.

Si noti che situazioni senza uscita (due task che si inibiscono reciprocamente l'esecuzione) possono verificarsi quando si chiama ObtainSemaphoreList e un altro task tenta di usare ObtainSemaphore per bloccare uno dei semafori della lista. Se si vuole bloccare più di un semaforo (ma non tutti), si dovrebbe per prima cosa ottenere il più alto livello di blocco per vincolare la struttura List che definisce i semafori di segnalazione della lista.

OpenDevice

Sintassi di chiamata della funzione

```
error = OpenDevice (devName, unitNumber, iORequest, flags)
D0                  A0       D0          A1          D1
```

Scopo della funzione

Questa funzione apre il dispositivo di I/O che possiede il nome indicato dal task nell'argomento devNome, e inizializza i parametri io_Device e io_Unit della struttura IORequest indicata. OpenDevice restituisce il valore zero se ha avuto successo, altrimenti restituisce un codice d'errore.

Argomenti della funzione

devName Indirizzo della stringa a terminazione nulla contenente il nome del dispositivo da aprire. Si noti che nel caso dei dispositivi residenti su disco il nome indicato dal task deve corrispondere al nome del file del dispositivo residente nella directory logica DEVS:. Per esempio, "narrator.device". Per convenzione, i nomi dei dispositivi di I/O sono sempre seguiti dall'estensione ".device".

unitNumber Numero dell'unità del dispositivo che dev'essere aperta. Per esempio, il controller di accesso ai dischi gestisce fino a quattro disk drive, e quindi il dispositivo TrackDisk prevede quattro unità, numerate da 0 a 3. Il numero di unità disponibili varia da dispositivo a dispositivo, e talvolta, come nel dispositivo Audio, l'argomento unitRequest assume un significato ancora diverso.

iORequest Indirizzo della struttura IORequest allocata dal task allo scopo d'inviare richieste di I/O al dispositivo. La funzione OpenDevice accede a questa struttura per inizializzarne due parametri: io_Device e io_Unit. Nel primo memorizza l'indirizzo base della struttura Library di gestione del dispositivo, che viene poi utilizzato dalle funzioni DoIO e SendIO per sapere a quale dispositivo è diretta la richiesta di I/O (cioè il messaggio predefinito rappresentato dalla struttura IORequest). Nel secondo, OpenDevice memorizza l'indirizzo della struttura Unit di gestione dell'unità che è stata aperta. Si noti che l'esatto significato attribuito dal dispositivo al valore memorizzato nel parametro io_Unit varia da dispositivo a dispositivo. La struttura di IORequest viene usata per le comunicazioni tra i task e i dispositivi, per inviare comandi a un dispositivo e ricevere dal dispositivo risposte, dati e altre informazioni. Si ricordi che la struttura

IORequest non è altro che un messaggio standardizzato per le comunicazioni con i dispositivi, e come tale dev'essere gestito come un messaggio generico. L'unica importante differenza è che quando s'invia un messaggio generico, si chiama la funzione PutMsg e si indica negli argomenti il destinatario, mentre per inviare una richiesta di I/O a un dispositivo occorre chiamare DoIO o SendIO indicando come argomento soltanto l'indirizzo della struttura IORequest: il destinatario è indicato nella struttura stessa del messaggio, all'interno del parametro io_Device.

flags Argomento flag da 32 bit tramite il quale si possono indicare alla funzione le modalità di apertura specifiche del dispositivo. Questo argomento è talvolta usato per richiedere l'apertura di un dispositivo con accesso esclusivo.

Discussione

Ci sono otto funzioni dell'Exec che hanno direttamente a che fare con i dispositivi di I/O: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Ciascuna di queste funzioni svolge un servizio per il dispositivo. Si vedano anche le spiegazioni relative a queste funzioni.

I dispositivi di I/O sono particolari librerie in grado di ricevere richieste di I/O dal sistema. Quando il task deve eseguire un'operazione di I/O con un dispositivo hardware, apre il corrispondente dispositivo software, formula una richiesta di I/O e gliela inoltra. Le richieste di I/O sono sempre contraddistinte da un comando, e quindi inoltrare una richiesta di I/O a un dispositivo significa impartire a quel dispositivo un comando. Quando il dispositivo riceve il comando, se può soddisfarlo non fa altro che cedere il controllo alla relativa routine contenuta nella sua libreria. In ultima analisi, quindi, inoltrare un comando corrisponde all'esecuzione di una funzione del dispositivo. Ma i task non hanno accesso diretto a queste funzioni, ad eccezione di BeginIO e AbortIO. comuni a tutti i dispositivi. In pratica, quando un task invia un comando con le funzioni DoIO o SendIO, è sempre la funzione BeginIO del dispositivo a riceverlo e smistarlo di conseguenza. I task hanno anche l'opportunità di chiamare BeginIO o AbortIO direttamente, ma non di chiamare le altre routine del dispositivo. Esiste un insieme standard di comandi che tutti i dispositivi del sistema riconoscono. Esso comprende i seguenti comandi:

- CMD_CLEAR. Azzera tutti i buffer interni del dispositivo relativi a una particolare unità. Si ricordi che alcuni dispositivi possiedono una serie di buffer interni che utilizzano per gestire i dati in transito verso l'hardware esterno e i task. Ovviamente, CMD_CLEAR non ha alcun effetto sui buffer definiti dal task.

- CMD_FLUSH. Ordina al dispositivo di abortire tutte le richieste di I/O che sono ancora in attesa nella coda alla request port di una particolare unità. Una volta che le richieste sono state rimosse, i task devono reinoltrarle se desiderano che vengano elaborate. Le richieste ritornano ai relativi mittenti con il codice d'errore IOERR_ABORTED memorizzato nel parametro io_Error.

- CMD_INVALID. Quando i dispositivi ricevono questo comando, in genere restituiscono il codice d'errore standard IOERR_NOCMD per indicare che il comando richiesto non è previsto dal dispositivo.

- CMD_READ. Ordina al dispositivo di leggere un certo numero di byte dai suoi buffer interni e di memorizzarli nel buffer definito dal task. Il numero dei byte da leggere viene specificato dal task nel parametro io_Length della struttura IOStdReq; il numero di byte che invece vengono effettivamente letti viene restituito dal dispositivo nel parametro io_Actual della stessa struttura. Vi sono comunque alcune eccezioni, e i dettagli che definiscono l'impiego di questo comando variano da dispositivo a dispositivo.

- CMD_RESET. Ordina al dispositivo di operare il reset di una particolare unità. Le routine interne del dispositivo vengono completamente reinizializzate, ripristinando le condizioni di default. CMD_RESET, inoltre, chiama CMD_FLUSH per abortire tutte le richieste di I/O accodate alla request port dell'unità, e chiama AbortIO per eliminare l'eventuale richiesta in corso di elaborazione. Inoltre, cancella tutte le strutture di dati utilizzate dalle routine interne del dispositivo per l'unità, e ogni registro hardware coinvolto.

- CMD_START. Ordina che riprenda l'elaborazione dei comandi da parte dell'unità precedentemente bloccata con il comando CMD_STOP. Se quando è stato inviato il comando CMD_STOP l'unità stava elaborando una richiesta, inoltrando CMD_START il comando riprende la propria esecuzione nel punto esatto in cui era stato bloccato. Quando non può farlo, è il sistema a scegliere il punto da cui deve far ripartire l'esecuzione.

- CMD_STOP. Ordina al dispositivo di sospendere l'elaborazione dei dati in una sua particolare unità. L'interruzione avviene non appena è possibile. Le richieste di I/O continuano ad accodarsi, ma l'unità non può elaborarle. La coda alla request port può in questo caso crescere rapidamente, producendo un enorme consumo di memoria da parte delle strutture di I/O che vengono accodate ma non elaborate. Il comando risulta utile per quei dispositivi che richiedono l'intervento dell'utente (come per esempio stampanti, plotter e reti di comunicazione).

- CMD_UPDATE. Ordina al dispositivo di riversare nell'hardware il

contenuto di tutti i buffer interni dell'unità indirizzata. Le informazioni mantenute in questi buffer in genere hanno origine nei buffer definiti dai task; i buffer interni dei dispositivi rappresentano punti di momentaneo parcheggio per le informazioni. Di solito, quindi, il dispositivo effettua quest'operazione automaticamente, come parte dei suoi compiti di routine; tuttavia a volte può essere utile provocare esplicitamente un aggiornamento dei dati sotto il controllo del task creato dal programmatore. Questo controllo diretto può essere necessario con i dispositivi che mantengono buffer interni di dati (cache), come i disk drive.

- CMD_WRITE. Ordina al dispositivo di trasferire un certo numero di byte da un buffer definito dal task al buffer interno dell'unità, ed eventualmente a un dispositivo hardware esterno (per esempio un disk drive). Il numero di byte viene specificato dal task nel parametro io_Length della struttura IOStdReq; il sistema indica poi il numero di byte effettivamente trasferiti nel parametro io_Actual della stessa struttura che viene restituita come risposta. Ancora una volta, i dettagli che definiscono l'impiego di questo comando variano da dispositivo a dispositivo.

Ogni dispositivo prevede poi un insieme di comandi specifici, non comuni agli altri dispositivi.

Oltre alla funzione BeginIO, i task possono anche chiamare la funzione AbortIO, prevista da ogni dispositivo. In questo caso, però, il task chiama la funzione AbortIO della libreria Exec, la quale a sua volta cede il controllo alla funzione AbortIO del dispositivo. Questa funzione serve per annullare una specifica richiesta di I/O inviata all'unità di un dispositivo, ed è in grado di sopprimere sia le richieste attive (cioè in fase di elaborazione da parte dell'unità), sia quelle ancora accodate (cioè in attesa di essere elaborate). Al contrario del comando CMD_FLUSH, la funzione AbortIO fornisce un meccanismo per annullare una singola richiesta di I/O inviata da un task a un'unità di un dispositivo.

I nomi dei dispositivi

Quando si cerca di aprire un dispositivo, occorre conoscerne il nome e indicarlo all'interno di una stringa di testo a terminazione nulla. Durante l'esecuzione della funzione OpenDevice, il sistema utilizza questo nome per rilevare se la struttura Device di gestione del dispositivo richiesto appare nella lista di sistema dei dispositivi (DeviceList). Se la trova, significa che il dispositivo è disponibile, e provvede ad aprirlo. Se invece non la trova, significa che il dispositivo è di tipo non-residente, e non è disponibile; in questo secondo caso, in maniera del tutto trasparente al task, accede alla directory logica DEVS: per rilevare se il nome indicato dal task corrisponde al nome di uno dei file presenti in quella directory. Se un file risponde a quel nome, il sistema lo carica in memoria, lo trasforma in una libreria tramite la funzione MakeLibrary, e lo

rende disponibile chiamando la funzione AddDevice. Infine lo apre. Se invece non trova nessun file con quel nome, restituisce il codice d'errore IOERR_OPENFAIL.

Se si vogliono cambiare alcune caratteristiche di un dispositivo e poi utilizzare la funzione MakeLibrary per riorganizzare il modulo di controllo della relativa libreria, è opportuno conservare il nome originale del dispositivo, altrimenti i task non riusciranno più ad aprirlo.

Ecco i nomi dei dispositivi standard dell'Amiga:

- audio.device
- clipboard.device
- console.device
- gameport.device
- input.device
- keyboard.device
- narrator.device
- parallel.device
- printer.device
- serial.device
- timer.device
- trackdisk.device.

La struttura IORequest

La struttura IORequest standard è il messaggio base che dev'essere impiegato per accedere a qualunque dispositivo. Molto spesso è il primo parametro di strutture di I/O più elaborate, come la struttura di I/O estesa standard IOStdReq, o come la struttura di I/O estesa non standard IOExtTD prevista dal dispositivo TrackDisk.

Il primo parametro della struttura IORequest è una struttura Message, che costituisce l'intestazione del messaggio. I parametri di questa sotto-struttura vengono utilizzati per gestire il messaggio. In particolare, il parametro mn_ReplyPort deve contenere l'indirizzo della reply port del task se si desidera che il messaggio, una volta giunto al dispositivo ed elaborato, venga restituito al mittente. Quando un task invia una richiesta di I/O, si aspetta che essa venga restituita, sotto forma di risposta, alla sua reply port. A questo proposito, si

tenga presente che se il task invia la richiesta tramite DoIO, è la funzione stessa a sondare la reply port e ad attendere che la richiesta ritorni al mittente; quando la risposta giunge, DoIO si preoccupa anche di estrarla dalla coda alla reply port. A causa di questo comportamento, la funzione DoIO viene definita sincrona: il task non riottiene il controllo finché la richiesta inviata non è stata restituita. Se invece la richiesta viene inoltrata tramite la funzione SendIO, è il task che deve provvedere a sondare la reply port indicata nel parametro mn_ReplyPort e a estrarre la risposta quando giunge.

Quando il dispositivo restituisce al mittente una richiesta di I/O che ha elaborato, il task che l'ha inviata può accedere ai suoi parametri per conoscere l'esito della richiesta ed eventualmente accedere agli altri dati coinvolti.

Sempre nella sotto-struttura io_Message, è presente anche il parametro mn_Length, che in genere indica la lunghezza in byte del contenuto del messaggio. Nell'interazione con i dispositivi questo parametro non viene impiegato, in quanto le strutture dei messaggi sono predefinite e quindi conosciute dai rispettivi dispositivi. Per ragioni di compatibilità è opportuno lasciare questo parametro a 0 e non attribuire alcun significato ai valori che nel corso dell'elaborazione può assumere. Per esempio, le funzioni di supporto alla libreria Exec CreateExtIO e DeleteExtIO (illustrate diffusamente in *Programmare l'Amiga Volume II*) utilizzano questo parametro per scopi interni di gestione.

Intestazione a parte, il messaggio vero e proprio della struttura IORequest è formato dai parametri che seguono la sotto-struttura io_Message, che sono nell'ordine io_Device, io_Unit, io_Command, io_Flags e io_Error. A questi, che sono essenziali, la struttura estesa IOStdReq aggiunge io_Actual, io_Length, io_Data, e io_Offset. Vediamone alcuni.

- io_Command dev'essere inizializzato con il codice del comando che si desidera inoltrare al dispositivo.

- io_Flags permette di associare alla richiesta di I/O un insieme di flag, il cui stato determina comportamenti diversi da parte del dispositivo nei confronti della richiesta. Quali flag siano riconosciuti e quali significati abbiano, è una cosa che varia da dispositivo a dispositivo; l'unico flag riconosciuto da tutti i dispositivi è IOF_QUICK, che serve per richiedere la modalità di accesso veloce. Si tenga presente che la funzione SendIO azzera sempre il parametro io_Flags prima di chiamare la funzione BeginIO del dispositivo indirizzato, mentre la funzione DoIO azzera sempre il parametro io_Flags, ma prima di chiamare la funzione BeginIO ne imposta il flag IOF_QUICK. Quindi, se un task desidera utilizzare il parametro io_Flags della struttura IORequest non deve utilizzare né SendIO, né DoIO: deve servirsi direttamente della funzione BeginIO, che non modifica il contenuto del parametro.

- io_Error è il parametro nel quale il task, quando riottiene in risposta la struttura di I/O, trova il codice che rappresenta l'esito del comando inoltrato. Il valore zero indica che non si sono verificati errori. Nel caso invece si verifichi una condizione d'errore, il dispositivo può restituire

uno dei codici d'errore standard (indicati nel file INCLUDE exec/errors.h), o uno dei suoi codici d'errore interni, evidentemente non standard.

- io_Actual indica, con quei comandi che implicano il trasferimento di dati tra task e dispositivo, il numero di byte effettivamente trasferiti, che non sempre coincidono con quelli richiesti dal comando. Le ragioni per una discrepanza fra i due valori possono essere molteplici, e variano da dispositivo a dispositivo. Non si deve comunque credere che una discrepanza corrisponda necessariamente a un errore o a una condizione anomala.

- io_Length dev'essere inizializzato dal task con il numero di byte da trasferire. Generalmente si tratta della dimensione del buffer predisposto dal task per inviare o ricevere dati. Per esempio, se un task vuole leggere un settore di una traccia su un disco, io_Length deve contenere il valore 512 dal momento che ogni settore comprende 512 byte. Se invece il task vuole leggere un'intera traccia del disco, deve indicare il valore 5632 (11 settori per 512 byte l'uno).

- io_Data, deve sempre contenere l'indirizzo del buffer che il task ha predisposto per inviare o ricevere dati.

- io_Offset serve per indicare un offset a quei dispositivi che trattano i dati suddividendoli in blocchi. Di solito si richiede che l'offset sia espresso in byte e costituisca un multiplo del blocco minimo previsto dal dispositivo. La dimensione del blocco minimo previsto da ciascun dispositivo è specificata nel relativo file INCLUDE. Per esempio, il blocco minimo previsto dal dispositivo disk drive è 512 byte, pari alla dimensione di un settore; l'offset dev'essere quindi un multiplo intero di 512 quando si accede ai disk drive.

Dalle spiegazioni di questi parametri si nota che la struttura IORequest è sufficiente per inviare tutti i comandi che non implicano trasferimenti di dati fra il task e il dispositivo, come per esempio CMD_CLEAR, mentre quando il comando comporta il trasferimento di dati è necessario ricorrere alla struttura IOStdReq.

L'uso di OpenDevice

I dispositivi di I/O sono organizzati sotto forma di librerie, al pari delle librerie condivise. Per accedere a una normale libreria occorre chiamare OpenLibrary, la quale restituisce l'indirizzo base della libreria aperta, indispensabile per chiamare le funzioni in essa contenute. Con i dispositivi, invece, le cose vanno diversamente, in quanto la funzione OpenDevice non restituisce l'indirizzo base del dispositivo, ma lo memorizza nel parametro io_Device della struttura di I/O indicata dal task.

OpenLibrary

Sintassi di chiamata della funzione

```
library = OpenLibrary (libName, version)
DØ                     A1       DØ
```

Scopo della funzione

Questa funzione consente di aprire la libreria condivisa che risponde al nome indicato dal task nell'argomento libName, e che possiede un numero di versione uguale o superiore a quello indicato nell'argomento version. Se la funzione riesce ad aprire la libreria, ne restituisce l'indirizzo base, cioè l'indirizzo della struttura Library che la definisce. Se invece l'apertura della libreria non ha successo, la funzione OpenLibrary restituisce il valore zero. Il task non è tenuto a sapere se la libreria è di tipo residente o non-residente, se è disponibile oppure no: la funzione lo solleva da questi dettagli, preoccupandosi di caricare la libreria da disco qualora non risulti disponibile in memoria. Per verificare se è disponibile, la funzione esamina la lista di sistema LibList e confronta il nome indicato dal task con quelli indicati nelle strutture Library della lista.

Argomenti della funzione

libName — Indirizzo della stringa di testo contenente il nome della libreria da aprire. Questa stringa viene confrontata con quelle individuate dai parametri lib_IdString delle strutture Library presenti nella lista di sistema LibList. Se non corrisponde a nessuna di esse, accede alla directory logica LIBS: alla ricerca di un file con quel nome. Se lo trova lo carica in memoria, chiama la funzione MakeLibrary per trasformarlo in una libreria e chiama la funzione AddLibrary per aggiungere la libreria al sistema, operazioni che svolge in maniera completamente automatica.

version — La versione minima che deve possedere la libreria perché la funzione la apra. La funzione confronta questo valore con il parametro lib_Version della struttura Library che risponde al nome indicato nell'argomento libName.

Discussione

Ci sono otto funzioni dell'Exec che riguardano le librerie condivise: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Si vedano anche le spiegazioni relative a queste funzioni.

Il sistema a librerie dell'Amiga è molto flessibile e soprattutto aperto. Chiunque può creare e aggiungere al sistema librerie di funzioni per snellire le proprie applicazioni. Le librerie non standard che i programmatori aggiungono al sistema non sono altro che nuove librerie non-residenti, al pari di quelle standard, come la libreria Version. I programmatori possono anche arrivare a crearsi vere e proprie collezioni di librerie.

Le librerie sono in genere composte da funzioni che in un certo ambito, come per esempio la grafica, sono utilizzate con una certa frequenza. Quando si creano nuove versioni di una libreria, è sempre importante cercare di mantenere la compatibilità verso il basso, cioè con le versioni precedenti. Se la nuova versione di una libreria non è compatibile verso il basso, le si dovrebbe assegnare un nuovo nome al fine di evitare problemi di coerenza che potrebbero confondere il sistema e causarne il crash.

Quando si desidera costruire una libreria personale di funzioni si devono seguire le seguenti indicazioni. Prima di tutto, il file della libreria su disco deve iniziare con le istruzioni Assembly:

```
moveq #XX, dØ
rts
```

(dove XX può essere qualunque codice d'errore ci sembri adatto) in modo che se si tenta di mandare in esecuzione la libreria come un programma qualunque, utilizzando cioè il comando RUN, l'AmigaDOS restituisce il valore XX. Queste istruzioni occupano quattro byte, e devono essere seguite da una serie di parametri organizzati secondo la struttura Resident. In particolare, se si desidera che il sistema quando carica la libreria esegua una particolare routine, occorre che il flag RTF_AUTOINIT del parametro rt_Flags sia azzerato. In questo caso, l'indirizzo della routine dev'essere inserito nel parametro rt_Init.

Se invece si vogliono predisporre alcuni dati perché il sistema chiami automaticamente la funzione MakeLibrary per trasformare il file caricato in memoria in una libreria, occorre che sia impostato il flag RTF_AUTOINIT. In questo secondo caso il parametro rt_Init deve contenere l'indirizzo di una tavola composta da quattro long word. Il sistema si aspetta che la prima long word contenga la grandezza in byte dell'area dati di gestione della libreria, cioè l'area composta dalla struttura Library e dai dati aggiuntivi previsti dalla particolare libreria. La grandezza di quest'area, sommata alla dimensione della tavola dei vettori di salto che MakeLibrary crea, indica la quantità di memoria necessaria per creare il modulo di controllo della libreria.

La seconda long word deve contenere l'indirizzo della tavola contenente i vettori (assoluti o relativi) che individuano le funzioni della libreria. La terza

long word può contenere l'indirizzo di una tavola d'inizializzazione, nel caso che il programmatore desideri inizializzare in maniera automatica alcune parti del modulo di controllo della libreria. L'indirizzo viene ceduto dalla funzione MakeLibrary alla funzione InitStruct.

Infine, l'ultima long word può contenere l'indirizzo di un'ulteriore routine d'inizializzazione della libreria, alla quale MakeLibrary cede il controllo prima di completarsi (sempre che l'indirizzo sia diverso da zero).

Si vedano anche le spiegazioni relative alle funzioni MakeLibrary e InitResident. Infatti, gli indirizzi contenuti nella tavola puntata da rt_Init vengono utilizzati dal sistema come argomenti della funzione MakeLibrary, che viene chiamata dopo che è stato caricato il file per creare il modulo di controllo della libreria, e prima chiamare la funzione AddLibrary (ovviamente, il sistema chiama automaticamente questa funzione soltanto se il flag RTF_AUTOINIT risulta impostato).

Riassumendo, il file che si crea deve iniziare con le già citate istruzioni Assembly seguite dalla struttura Resident, i cui parametri indicano al sistema come dev'essere installata la libreria. Il file della libreria deve poi contenere, in qualunque ordine, la tavola dei vettori che individuano le funzioni della libreria, le funzioni della libreria vere e proprie, la tavola d'inizializzazione da sottoporre eventualmente a InitStruct, l'eventuale routine d'inizializzazione della libreria che verrebbe automaticamente eseguita da MakeLibrary. Un file strutturato in questo modo può essere trattato dal sistema come una libreria, libreria che può anche essere quella di un dispositivo o di una risorsa. L'organizzazione descritta vale infatti per qualsiasi tipo di libreria. L'ultimo ragguaglio riguarda il nome del file, che dev'essere lo stesso da indicare poi nella struttura Library della libreria.

È anche interessante esaminare brevemente le operazioni da svolgere quando si desidera rendere disponibile una libreria dopo averne caricato in memoria il file ed essere risaliti all'indirizzo della struttura Resident. Si tratta di operazioni che il sistema svolge automaticamente, ma che un programmatore potrebbe voler conoscere, non tanto per creare un "caricatore di librerie personalizzato", quanto per modificare le librerie già residenti per poi reinstallarle.

Il file della libreria può essere caricato in memoria tramite la funzione LoadSeg dell'AmigaDOS, la quale restituisce l'indirizzo del primo byte del segmento di memoria in cui il file è stato caricato, sotto forma d'indirizzo BCPL. Per risalire all'indirizzo del primo parametro della struttura Resident, occorre moltiplicarlo per quattro e sommargli il valore otto: il numero di byte occupati dalle istruzioni Assembly che precedono la struttura Resident più la dimensione di un puntatore che collega questo ad altri eventuali segmenti.

Entrati in possesso dell'indirizzo della struttura Resident che in pratica intesta il file della libreria, si possono seguire due strade: chiamare la funzione InitResident oppure procedere manualmente alla creazione e installazione della libreria. Per il primo metodo, si consulti la spiegazione della funzione InitResident, mentre per il secondo occorre spendere qualche parola in più. Come abbiamo già puntualizzato, lo stato del flag RTF_AUTOINIT indica il modo in cui il programmatore desidera che la libreria venga installata. Se questo flag non è impostato, il controllo viene ceduto alla routine puntata dal

paramentro rt_Init. Se invece è impostato bisogna chiamare la funzione MakeLibrary indicando come argomenti i valori contenuti nella tavola puntata da rt_Init, e successivamente chiamare la funzione AddLibrary per rendere la libreria disponibile (attenzione, non per aprirla). MakeLibrary crea il modulo di controllo della libreria, composto dalla tavola dei vettori di salto alle funzioni della libreria, dalla struttura Library e dall'area dati prevista dalla libreria. Quando MakeLibrary restituisce il controllo, la libreria è pronta per essere resa disponibile tramite la funzione AddLibrary, la quale provvede a inserire la relativa struttura Library nella lista LibList.

Uno dei vantaggi delle librerie dell'Amiga è la libertà con cui le loro parti (modulo di controllo e funzioni) possono essere disposte in memoria. Con l'organizzazione descritta, si può chiamare qualsiasi funzione di qualsiasi libreria senza sapere quale posizione occupa effettivamente in memoria. Esaminiamo brevemente la procedura che permette di risalire alla posizione di una funzione in una libreria.

L'unico indirizzo assoluto e inalterabile nella memoria dell'Amiga è 0x04. In questa long word è contenuto l'indirizzo base della libreria Exec, il quale ci permette di chiamare tutte le funzioni dell'Exec. In Assembly occorre essere sicuri che l'indirizzo base della libreria Exec sia sempre memorizzato nel registro A6 ogni volta che dobbiamo accedere a una funzione della libreria Exec, come per esempio OpenLibrary. In C questa condizione è assicurata dal compilatore. Tramite un opportuno offset e l'indirizzo base della libreria Exec, possiamo chiamare OpenLibrary indicando a quale libreria vogliamo accedere. La funzione restituisce l'indirizzo base di quella libreria, che insieme agli offset ci permette di chiamare ogni funzione in essa contenuta, proprio come si accede alle funzioni della libreria Exec. Con questa organizzazione, tramite un unico indirizzo assoluto si può risalire a qualsiasi funzione presente nel sistema, e le librerie non sono costrette a risiedere in una posizione dello spazio indirizzabile piuttosto che in un'altra. Anche la struttura modulare della libreria è di grande utilità; per esempio, nel caso delle librerie residenti il modulo di controllo è sempre in RAM, ma le funzioni sono in ROM.

L'elenco che segue mostra i nomi delle librerie previste dal sistema Amiga, seguiti dai nomi dei puntatori che i programmatori in C devono impiegare per memorizzare i relativi indirizzi base (si noti che è di fondamentale importanza durante l'apertura di una particolare libreria che il programmatore associ il risultato di OpenLibrary alla variabile puntatore di quella libreria; se infatti venisse adottata una variabile di nome qualunque, il linker non potrebbe generare il programma eseguibile).

Nome della libreria	**Variabile per l'indirizzo base della libreria**
exec.library	SysBase
graphics.library	GfxBase
intuition.library	IntuitionBase
dos.library	DOSBase
layers.library	LayersBase
clist.library	CListBase
math.library	MathBase

potgo.library	PotgoBase
diskfont.library	DiskfontBase
mathtrans.library	MathTransBase
mathieeedoubbas.library	MathIeeeDoubBasBase
mathieeedoubtrans.library	MathIeeeDoubTransBase
translator.library	TranslatorBase

Per utilizzare una libreria di sistema bisogna dichiarare una variabile puntatore con il nome previsto per quella libreria dal compilatore C. Ecco alcuni esempi di istruzioni di programma che svolgono questa operazione:

```
struct ExecBase *SysBase;
struct GfxBase *GfxBase;
struct IntuitionBase *IntuitionBase;
struct Library *DOSBase;
struct Library *LayersBase;
struct Library *CListBase;
struct Library *MathBase;
struct Library *PotgoBase;
```

Per accedere alle funzioni di una libreria, occorre aprirla e ottenerne l'indirizzo base. Vediamo per esempio come si può aprire la libreria Graphics:

```
GfxBase = OpenLibrary ("graphics.library",ØL);
if (GfxBase == Ø)
exit (LIBRERIA_GRAPHICS_NON_TROVATA);
```

L'indirizzo base restituito da OpenLibrary viene poi usato insieme con gli offset per chiamare le funzioni della libreria. Quando il programma è scritto in C, le chiamate alle funzioni delle librerie vengono opportunamente filtrate da routine Assembly, le quali garantiscono che le chiamate siano comprensibili al sistema, svolgendo quindi la mansione d'interfaccia tra le chiamate delle funzioni in C e il sistema. Infatti, il linguaggio C prevede che il passaggio dei parametri alle funzioni avvenga sullo stack, mentre le funzioni delle librerie si aspettano di ricevere i parametri nei registri della CPU. Questa differenza rende necessario che il linker inserisca nel programma oggetto opportune routine d'interfacciamento in Assembly. Se invece il programma viene scritto direttamente in Assembly, è il programmatore stesso a sfruttare i registri della CPU quando deve indicare a una funzione i relativi argomenti, e quindi non sono necessarie le routine Assembly d'interfacciamento.

Nell'interazione con le librerie ci sono alcune cose che si possono dare per scontate e altre no. Per esempio, per tutto il tempo che una libreria è aperta è lecito assumere che il suo indirizzo base non può cambiare, mentre non è lecito fare affidamento sulla posizione assoluta di una sua funzione in memoria, in quanto un altro task potrebbe aver modificato la tavola dei vettori di salto tramite la funzione SetFunction. In genere, è opportuno affidarsi sempre e solo al valore dell'indirizzo base restituito dalla funzione OpenLibrary per qualsiasi operazione coinvolga la libreria.

OpenResource

Sintassi di chiamata della funzione

```
resource = OpenResource (resName)
DØ                      A1
```

Scopo della funzione

Questa funzione apre una risorsa e restituisce l'indirizzo della relativa struttura Library di gestione, oppure lo zero se l'apertura non ha avuto successo. Perché la risorsa indicata venga aperta occorre che sia disponibile, cioè che sia elencata nella lista di sistema ResourceList delle risorse disponibili.

Argomenti della funzione

resName — Indirizzo della stringa contenente il nome della risorsa da aprire; i file INCLUDE relativi ad alcune risorse definiscono i nomi delle risorse con particolari costanti.

Discussione

Ci sono tre funzioni dell'Exec dedicate alla gestione delle risorse nel sistema dell'Amiga: AddResource, OpenResource e RemResource. Si noti che non esiste una funzione CloseResource.

La funzione OpenResource apre le risorse nello stesso modo in cui OpenDevice apre i dispositivi e OpenLibrary apre le librerie. I nomi delle risorse seguono le stesse regole dei nomi dei dispositivi.

Esistono cinque risorse nel sistema Amiga, alle quali è possibile riferirsi con le seguenti definizioni e nomi:

DISKNAME	oppure	disk.resource
MISCNAME	oppure	misc.resource
CIAANAME	oppure	ciaa.resource

CIABNAME	oppure	ciab.resource
POTGONAME	oppure	potgo.resource

Le strutture relative a queste risorse sono contenute nei file INCLUDE disk.h, misc.h, cia.h e potgo.h.

Procure

Sintassi di chiamata della funzione

```
result = Procure (semaphore, message)
D0               A0          A1
```

Scopo della funzione

Questa funzione serve per appropriarsi di un semaforo basato su messaggi. Se il semaforo è disponibile, Procure restituisce il valore TRUE (vero) e l'argomento message non viene utilizzato. Se invece il semaforo è già in uso, Procure restituisce il valore FALSE (falso). In questo caso, quando il semaforo si libera, il task che aveva chiamato Procure riceve alla sua reply port il messaggio che ha indicato come secondo argomento della funzione (se si era messo in attesa del messaggio, riottiene il controllo). L'arrivo del messaggio significa che il semaforo a suo tempo richiesto si è liberato. In caso d'insuccesso, quindi, Procure non sospende l'esecuzione del task ma fa in modo di avvisarlo quando arriva il suo turno di accesso.

Argomenti della funzione

semaphore Indirizzo della struttura Semaphore che costituisce il semaforo richiesto.

message Indirizzo di una struttura Message allocata dal task per definire un messaggio. Se Procure rileva che il semaforo indicato non è disponibile, accoda questo messaggio alla message port sm_MsgPort della struttura Semaphore, accodando così la richiesta di accesso del task. Ogni volta che un task libera un semaforo basato su messaggi, il sistema restituisce il primo messaggio che trova nella coda a quella

message port, avvisando il relativo task che il semaforo è di sua proprietà, fino a quando non lo libera. Quindi, questo indirizzo deve individuare il messaggio che il task riceverà se Procure restituisce il valore FALSE.

Discussione

Nell'Exec esistono due funzioni che riguardano i semafori basati su messaggi: Procure e Vacate. Nell'Amiga oltre a questo tipo di semafori esistono anche i semafori di segnalazione, per i quali i task devono impiegare le strutture SignalSemaphore e SemaphoreRequest. Si vedano a questo proposito le spiegazioni relative alla funzione AddSemaphore.

I semafori basati su messaggi sono costituiti da una struttura Semaphore, che contiene una sotto-struttura MsgPort e un parametro contatore. Questa struttura è definita come segue:

```
struct Semaphore {
    struct MsgPort sm_MsgPort;
    WORD sm_Bids;
};
```

Il parametro sm_MsgPort è il nome della sotto-struttura MsgPort che riceverà i messaggi di risposta relativi al semaforo e appartenenti al semaforo basato su messaggi definito dalla struttura Semaphore. Il parametro sm_Bids è un contatore del numero di messaggi in coda alla message port.

Proprio come si possono proteggere aree RAM con Forbid/Permit e ObtainSemaphore/ReleaseSemaphore, si può raggiungere lo stesso scopo ricorrendo ai semafori basati su messaggi e alla combinazione Procure/Vacate. Al pari dei semafori di segnalazione, i semafori basati su messaggi non sospendono lo scambio di controllo fra i task. Inoltre, i semafori basati su messaggi forniscono un vantaggio rispetto ai semafori di segnalazione: le richieste di accesso vengono accodate se il semaforo non è libero, e il task che ha avanzato la richiesta non viene sospeso. Quindi, grazie a questo comportamento asincrono e all'uso dei messaggi, ogni task può contemporaneamente aspettare che si liberino diversi semafori basati su messaggi ai quali ha tentato di accedere. Per ognuno di essi il task ha allocato un messaggio e ha chiamato la funzione Procure. Ora può esaminare periodicamente la reply port dove i messaggi relativi a quei semafori dovrebbero giungere.

È importante che il parametro mp_Flags della sotto-struttura MsgPort contenuta nella struttura Semaphore venga inizializzato a PA_IGNORE, dal momento che il sistema utilizza il parametro mp_SigTask per memorizzarvi l'indirizzo dell'ultima struttura Message che ha estratto e restituito al mittente tramite la funzione ReplyMsg (questo mittente è il task che possiede il semaforo). Inoltre, è fondamentale che nel messaggio che crea, il task indichi l'indirizzo della sua reply port.

Infine si noti che quando si alloca una struttura Semaphore per creare un nuovo semaforo, occorre inizializzarne il parametro sm_Bids a -1, per indicare non solo che non ci sono richieste di accesso in coda (il valore sarebbe zero), ma anche che il semaforo è libero. Se si desidera rendere pubblico il semaforo, si devono impostare anche i parametri ln_Name e ln_Pri della sotto-struttura Node contenuta nella sotto-struttura MsgPort e, tramite la funzione AddPort, si deve aggiungere alla lista delle message port di sistema la message port appartenente al semaforo. In questo modo il semaforo è facilmente rintracciabile, come qualsiasi altra message port pubblica.

PutMsg

Sintassi di chiamata della funzione

PutMsg (msgPort, message)
A0 A1

Scopo della funzione

Questa funzione inserisce il messaggio indicato come secondo argomento nella coda alla message port indicata come primo argomento. Questa operazione corrisponde a "inviare un messaggio", e il destinatario è la message port nella cui coda viene inserito. Si ricordi che "inserire" un messaggio all'interno di una coda in realtà comporta solo l'aggiornamento di due parametri puntatore presenti all'interno della struttura Node che intesta la struttura Message, e quindi che non corrisponde in nessun modo né a trasferire né a copiare la struttura Message da un'area di memoria all'altra. La struttura Message rimane sempre dov'è stata originariamente allocata.

Il corpo di un messaggio può occupare fino a 64K di RAM, e deve sempre iniziare con una struttura Message. Tutti i messaggi che transitano nell'Amiga iniziano con una struttura Message, seguita dal corpo vero e proprio del messaggio. Per quanto riguarda la forma del messaggio, cioè l'organizzazione dei dati che lo compongono, esistono alcune convenzioni. Se il messaggio serve per inoltrare informazioni a un task che non sia di sistema, l'organizzazione del messaggio è a nostra discrezione: l'importante, ovviamente, è che il mittente e il destinatario siano al corrente dell'organizzazione convenuta. Per inviare informazioni al sistema, invece, esistono opportuni messaggi predefiniti. Per esempio, se occorre inviare un messaggio al dispositivo TrackDisk, occorre attenersi all'organizzazione della struttura IOExtTD, nella quale il primo elemento è una struttura Message, mentre tutti i parametri che seguono costituiscono il messaggio predefinito.

Ricordiamo quali sono le "norme di comportamento" da seguire per evitare conflitti tra i task nella gestione dei messaggi. Il task mittente deve allocare la struttura del messaggio, inizializzarne i parametri di gestione e il contenuto vero e proprio, e inviarlo tramite la funzione PutMsg indicando l'indirizzo della message port destinataria. Se il mittente desidera ricevere il messaggio di ritorno, deve memorizzare nel parametro mn_ReplyPort della struttura Message l'indirizzo della sua message port adibita a reply port. Dal momento in cui il mittente chiama PutMsg, non deve più considerare di sua proprietà l'area di memoria occupata dal messaggio; questo significa che non deve assolutamente modificare il messaggio, che può invece essere soggetto ad alterazioni da parte del destinatario. Il mittente può ritenere di essere di nuovo in possesso del messaggio solo quando lo riceve in risposta alla sua reply port.

Quando il destinatario riceve il messaggio alla sua message port, lo esamina, eventualmente copiando le informazioni che lo interessano in apposite locazioni di memoria, lo modifica se desidera restituire a sua volta altre informazioni, e chiama la funzione ReplyMsg per restituirlo al mittente. Da questo momento in poi deve considerare inaccessibile l'area di memoria occupata dal messaggio: il messaggio non gli appartiene più.

Quando il mittente riottiene il messaggio che aveva inviato, può esaminare la risposta in esso contenuta, e poi può fare del messaggio quello che ritiene più opportuno; anche liberare la memoria occupata, eliminandolo completamente.

Queste norme non sono rigidissime (anche se sono vivamente consigliate, soprattutto nelle interazioni con i dispositivi di I/O e con il sistema in generale), e possono quindi essere modificate secondo particolari convenzioni. Per esempio, due task cooperanti potrebbero concordare che non hanno bisogno di risposte formali eseguite tramite la funzione ReplyMsg quando scambiano messaggi fra loro. Una nuova convenzione potrebbe essere che il mittente considera un messaggio completamente elaborato dal destinatario quando rileva che uno dei parametri del messaggio assume un particolare valore. Oppure, un'altra convenzione potrebbe essere che il destinatario di un messaggio quando lo riceve ne diventa proprietario a tutti gli effetti e che non lo deve restituire al mittente; in tal caso il mittente creerebbe il messaggio e il destinatario provvederebbe a liberarne la memoria o a riutilizzarla. Comunque, se non ci sono valide ragioni è consigliabile attenersi alle norme standard.

Argomenti della funzione

msgPort — Indirizzo della message port alla quale è destinato il messaggio.

message — Indirizzo della struttura Message che rappresenta il messaggio da inviare.

Discussione

Ci sono tre funzioni dell'Exec che riguardano la gestione dei messaggi: PutMsg, GetMsg e ReplyMsg. Queste funzioni permettono ai task di scambiarsi informazioni per mezzo di messaggi. Dato che i messaggi passano attraverso le message port, è importante anche tenere presente il ruolo delle funzioni di gestione delle message port: AddPort, FindPort, RemPort e WaitPort.

Per comprendere a fondo cosa accade quando una message port riceve un messaggio, è necessario analizzare il comportamento della funzione PutMsg. Le prime operazioni che compie sono impostare a NT_MESSAGE il parametro ln_Type della struttura Message, disabilitare gli interrupt, e inserire il messaggio in fondo alla coda della message port indicata come primo argomento. Poi, se rileva che il parametro mp_SigTask della message port in questione contiene il valore zero, restituisce semplicemente il controllo. Quindi, ogni volta che una message port creata con mp_SigTask azzerato riceve un messaggio, il messaggio viene accodato e non accade nient'altro. Se invece il parametro mp_SigTask contiene un valore diverso da zero, la funzione PutMsg prosegue con altre azioni che dipendono dal valore contenuto nel parametro mp_Flags della struttura MsgPort della message port ricevente. Vi sono quattro casi.

- Se mp_Flags contiene il valore PA_SIGNAL, PutMsg chiama la funzione Signal per generare il segnale assegnato alla message port. Se il task che l'ha assegnato è in attesa del segnale, riottiene il controllo. PutMsg ottiene il numero del bit di segnale dal parametro mp_SigBit e l'indirizzo della struttura Task del task dal parametro mp_SigTask, entrambi contenuti nella struttura MsgPort. Pertanto, questi due parametri devono essere opportunamente inizializzati. Dopo quest'operazione, PutMsg riabilita gli interrupt e restituisce il controllo.

- Se mp_Flags contiene il valore PA_SOFTINT, PutMsg chiama la funzione Cause indicando come argomento l'indirizzo contenuto nel parametro mp_SigTask. Questo indirizzo dev'essere quello di una struttura Interrupt opportunamente inizializzata per definire la routine di servizio dell'interrupt software causato dalla funzione Cause. Dopo quest'operazione, PutMsg riabilita gli interrupt e restituisce il controllo.

- Se mp_Flags contiene il valore PA_IGNORE non esegue nessun'altra operazione fuorché riabilitare gli interrupt e restituire il controllo. In questo caso il comportamento di PutMsg è identico al caso in cui mp_SigBit contiene il valore zero. Quindi, sia impostando a zero mp_SigTask, sia impostando mp_Flags a PA_IGNORE si fa in modo che l'arrivo di un messaggio non provochi nessuna azione collaterale, ma fra i due metodi esiste una differenza. Il secondo, infatti, permette di rendere passiva la message port (e poi di nuovo attiva, eventualmente) senza dover azzerare il parametro mp_SigTask e quindi senza dover salvare ogni volta il suo contenuto. Questo metodo è anche più veloce.

- Se mp_Flags contiene il valore PF_ACTION, PutMsg ritiene che il parametro mp_SigTask contenga l'indirizzo di una routine, e provvede a chiamarla tramite la funzione Assembly jsr. Quando riottiene il controllo, PutMsg riabilita gli interrupt e restituisce il controllo. Si noti che se il task è in attesa di un segnale dalla message port e quindi la sua esecuzione è sospesa, l'esecuzione della routine associata alla message port (che evidentemente è una parte del task) non corrisponde a risvegliare il task. Di solito, si fa in modo che sia questa routine a decidere se chiamare la funzione Signal per riattivare il task, basandosi magari sul contenuto del messaggio. Si noti anche che questa routine viene eseguita mentre gli interrupt sono disabilitati. Il suo compito potrebbe essere quello di mantenere ordinata la coda secondo i livelli di priorità dei messaggi (il sistema li lascia nell'ordine in cui sono arrivati). Per mantenere l'ordinamento a priorità, è sufficiente che la routine estragga l'ultimo messaggio in coda tramite la funzione RemTail, e lo reinserisca nella stessa coda utilizzando la funzione Enqueue, che si basa sulla priorità del nodo per stabilire dove inserirlo all'interno della lista. In questo modo, la gestione della lista non è più di tipo FIFO, anche se i nodi continuano a essere estratti dalla sommità e inseriti dal fondo.

Questi sono i possibili funzionamenti di PutMsg, e quindi le possibili configurazioni che può assumere una message port. Nella creazione delle message port, si tenga conto che il parametro mp_Flags è esso stesso un flag, e non un insieme di bit di flag. Pertanto, le costanti PA_SIGNAL, PA_SOFTINT, PA_IGNORE e PF_ACTION sono valori, e non numeri di bit, e non è quindi possibile attribuire a una message port più di uno dei quattro comportamenti alla volta.

I segnali

Nel sistema Amiga un segnale software non è altro che un bit di flag che sotto certe condizioni passa da zero a uno. Ogni task possiede un proprio set di 32 bit di segnale, dei quali soltanto 16 sono a sua disposizione, mentre gli altri sono riservati al sistema. Descriviamo ora le fasi che caratterizzano l'uso di un segnale.

Un task decide di utilizzare uno dei suoi segnali per comunicare con una delle sue message port. Per prima cosa deve allocare un bit di segnale; può farlo richiedendo un particolare bit (o il primo che risulta disponibile) tramite la funzione AllocSignal. Quest'operazione non fa altro che "marcare" il bit prescelto, cioè indicare al sistema e al task che il corrispondente segnale è stato allocato; è vero che essendo il task l'unico che può allocare i suoi segnali l'operazione di marcare quelli in uso sarebbe un compito che il task potrebbe svolgere personalmente senza difficoltà, ma il sistema Exec lo aiuta ugualmente mantenendo traccia dei segnali allocati.

A questo punto, il task dispone del numero del segnale prescelto. Per assegnarlo a una delle sue message port, non fa altro che memorizzare quel

numero nel parametro mp_SigBit, l'indirizzo della propria struttura Task nel parametro mp_SigTask, e impostare il flag PA_SIGNAL del parametro mp_Flags. Si osservi che il segnale poteva essere assegnato anche a una routine di interrupt, o perfino a un altro task.

A questo punto, il task può entrare in attesa che il segnale allocato e assegnato gli venga prima o poi inviato. Per farlo chiama la funzione Wait, indicando come argomento una maschera dei bit di segnale. Uno dei vantaggi dei segnali è proprio che un task può attenderne parecchi contemporaneamente. Se per esempio possiede tre message port e desidera disporsi in attesa di segnali da tutte e tre, basta che allochi e assegni un diverso segnale per ognuna, e può entrare in attesa con un'unica chiamata della funzione Wait.

Ora, non appena la message port in questione riceve un messaggio, il sistema fa in modo d'inviare il relativo segnale al task, cioè imposta nel parametro tc_SigRecvd della struttura Task il bit di segnale che era stato assegnato a quella porta. Il task non si rende conto di questa operazione, ma riottiene il controllo dalla funzione Wait.

Ma se era entrato in attesa di diversi segnali non può sapere quale di questi ha causato il suo risveglio. Deve allora confrontare il numero del bit di segnale restituito da Wait con quello contenuto nei parametri mp_SigBit delle message port coinvolte. Questa verifica gli consente di scoprire l'origine del segnale e comportarsi di conseguenza.

Questa descrizione dei segnali si può applicare senza restrizioni ad altri contesti. Per esempio, se TaskA alloca uno dei suoi segnali ed entra in attesa di riceverlo, un generico TaskB, conoscendo il numero del bit di quel segnale e l'indirizzo della struttura Task di TaskA, può originare il segnale e provocare il risveglio di TaskA chiamando semplicemente la funzione Signal dell'Exec.

Le message port

PutMsg colloca i messaggi in una lista di tipo FIFO (una coda) appartenente alla message port indicata come argomento. Ciascun task può disporre di un qualsiasi numero di message port, che possono essere pubbliche o private (ovvero non rintracciabili nel sistema). Le message port pubbliche vengono così definite perché appaiono all'interno della lista di sistema PortList, alla quale qualsiasi task può accedere; una message port può essere resa pubblica solo se il task le attribuisce un nome.

Grazie al sistema delle message port e dei messaggi, ogni task può avere a che fare informazioni provenienti dall'intero sistema, informazioni che si accodano alle sue message port in attesa di essere estratte ed elaborate.

Quando un messaggio giunge a una message port, spesso è preceduto nella coda da altri messaggi non ancora elaborati. Se il task per un certo periodo non entra in attesa di segnali da una sua message port, la relativa coda si allunga. L'arrivo di ogni messaggio causa la generazione del segnale assegnato alla porta, ma dal momento che il task non è in attesa non può rilevare l'arrivo di questi segnali. Inoltre, non essendo previsto un contatore, uno stesso segnale può arrivare più volte ma il task non avrà mai la possibilità di sapere quante volte il bit di segnale è stato attivato nella struttura Task del task.

A un certo momento, il task può decidere di mettersi in attesa di messaggi da una certa message port; chiama quindi la funzione Wait. Questa funzione rileva che uno dei bit di segnale allocati, quello assegnato alla message port in questione, è impostato, e restituisce subito il controllo. Al task risulta quindi che è giunto un messaggio (anche se potrebbero esserne già giunti parecchi). Questo significa che se il task preleva il primo messaggio dalla coda, potrebbe non trattarsi dell'ultimo messaggio pervenuto.

Un altro aspetto che occorre considerare nella gestione delle message port e dei messaggi è lo scambio di controllo fra i task, che si verifica periodicamente in accordo con le varie priorità dei task. Se per ipotesi i task presenti nel sistema avessero tutti eguale priorità, il tempo di CPU verrebbe ripartito fra loro in egual misura (per la precisione, in una situazione in cui i task avessero tutti la stessa priorità ciascuno riceverebbe 64 millisecondi di tempo della CPU, un intervallo temporale che viene definito "time slice", fetta di tempo). Quando un task riottiene il controllo dalla funzione Wait, al massimo dispone di 64 millisecondi prima di perderlo per ragioni che non dipendono dalle sue azioni. Considerando il tempo necessario per individuare quale message port ha generato il segnale e quello per prelevare tutti i messaggi in essa accodati, soprattutto nel caso di una coda particolarmente affollata, il successivo scambio di controllo può verificarsi quando il task non ha ancora svuotato interamente la coda; quindi alcuni messaggi dovrebbero essere estratti ed elaborati al successivo turno di CPU del task.

Il significato delle code alle message port

L'ordine in cui una serie di messaggi è accodata in una message port ricalca esattamente l'ordine con cui sono sopraggiunti; il parametro di priorità dei messaggi non viene mai cosiderato. Le code vengono gestite attraverso la sotto-struttura di tipo List mp_MsgList contenuta nella struttura MsgPort. Tutta la gestione dei messaggi è effettuata dalle funzioni dell'Exec, un lavoro di considerevole complessità quando nel sistema coesistono svariati task.

RawDoFmt

Sintassi di chiamata della funzione

```
RawDoFmt (formatString, dataStream, putCharFunction, putCharData)
          A0            A1          A2               A3
```

Scopo della funzione

Questa funzione formatta un flusso di caratteri in accordo alla sintassi prevista dal linguaggio C nella formattazione delle stringhe di testo durante le procedure di output (funzioni printf, fprintf e sprintf). Il flusso dei caratteri di input viene scandito carattere per carattere; ogni carattere viene ricopiato nel flusso di ouput. Ogni volta che la funzione incontra nel flusso di input il carattere "%", lo interpreta come l'inizio di una direttiva di formattazione; la sintassi di queste direttive dev'essere conforme a quella prevista dal linguaggio C. L'elaborazione di una direttiva causa l'inserimento della rappresentazione ASCII del dato nel flusso di output. La conversione in ASCII del dato viene eseguita secondo le disposizioni contenute nella direttiva di formattazione; se per esempio il dato è numerico, la sua rappresentazione ASCII potrà essere quella decimale o esadecimale. A ogni direttiva corrisponde quindi un dato, che dev'essere contenuto in un terzo flusso, detto flusso di dati. Dopo l'elaborazione di una direttiva, la copia dei caratteri dal flusso di input al flusso di output riprende normalmente fino alla direttiva successiva. L'utilità di una funzione come RawDoFmt risiede nella possibilità di definire una stringa di testo contenente campi indeterminati, che di volta in volta vengono ridefiniti in modo da ottenere una stringa completamente determinata, simile alle altre come testo base ma diversa in alcuni campi.

Argomenti della funzione

formatString — Indirizzo di una stringa di caratteri conforme al formato di output delle stringhe previsto dal linguaggio C per le funzioni printf, fprintf e sprintf. La stringa dev'essere a terminazione nulla, e l'abbiamo definita flusso dei caratteri di input.

dataStream — Indirizzo di un buffer contenente un flusso di dati non formattato, che sarà elaborato per inserire i dati nella stringa di output rispettando i tipi definiti nelle direttive di formattazione.

putCharFunction — Indirizzo di una funzione di formattazione dati che dev'essere strutturata per ricevere in D0 i caratteri e in A3 l'indice assoluto all'interno del buffer che deve riceverli. Questa funzione viene chiamata ogni volta che RawDoFmt genera un nuovo carattere da inviare in output. RawDoFmt non modifica mai il valore contenuto nel registro A3, che è quindi sotto il dominio del task e successivamente della funzione di formattazione.

putCharData Indirizzo del buffer destinato a ricevere il flusso dei dati di output generato dalla formattazione del flusso di input.

Discussione

Spesso si dispone di flussi di dati non formattati, per esempio un insieme di valori espressi su long word, e si desidera trasformarli in stringhe ASCII per inserirli all'interno di stringhe di testo. Questa trasformazione e integrazione di un insieme di dati all'interno di una stringa di testo è il compito svolto dalla funzione RawDoFmt. Essa, data una stringa di input contenente direttive di formattazione e un flusso di dati, genera una stringa di output uguale a quella di input, a eccezione delle direttive di formattazione, che vengono sostituite dai corrispondenti dati opportunamente trasformati in sequenze di caratteri ASCII.

La funzione RawDoFmt richiede come argomenti quattro indirizzi. Il primo, formatString, deve individuare la stringa di input contenente testo e direttive di formattazione. Le direttive della stringa devono attenersi al formato previsto dal C per le stringhe di output.

Il secondo indirizzo, dataStream, deve individuare un buffer contenente i dati da integrare nella stringa di ouput al posto delle corrispondenti direttive presenti nella stringa di input. Questi dati possono avere origine da qualsiasi sorgente.

Il terzo indirizzo, putCharFunction, deve individuare una funzione definita dall'utente per elaborare il flusso di output dei caratteri generato da RawDoFmt. Tale funzione riceve il controllo ogni volta che RawDoFmt ha pronto un carattere di output. Si è liberi di costruire questa funzione in accordo con le proprie necessità e si tenga conto che il registro A3 individua inizialmente l'inizio del buffer predisposto per ricevere la stringa formattata. La funzione putCharFunction dev'essere chiamata nel seguente modo:

```
putCharFunction (character, putCharData)
                D0: 0-8   A3
```

dove character è il carattere di output, e putCharData è la locazione di memoria passata dal task alla funzione RawDoFmt.

Il quarto e ultimo indirizzo, putCharData, deve individuare un buffer destinato a ricevere la stringa formattata; questo argomento viene passato alla funzione putCharFunction definita dal programmatore, la quale può trattarlo come desidera, per esempio come indice assoluto all'interno del buffer.

La funzione RawDoFmt può essere illustrata con un esempio. Si supponga di disporre di una serie di tre dati che si desidera conglobare all'interno della seguente stringa rispettando i formati indicati dalle tre direttive presenti.

```
"%d carattere: %c/hex: 0x%2x"
```

I tre dati sono tutti numerici: secondo le direttive indicate, il primo

dev'essere trasformato in una stringa ASCII che lo rappresenti in notazione decimale, il secondo dev'essere semplicemente inserito nella stringa (è infatti un codice ASCII), e il terzo dev'essere trasformato in una stringa ASCII che lo rappresenti in notazione esadecimale su due caratteri ASCII. L'intera stringa di input deve risiedere in un'area RAM, e il suo indirizzo dev'essere indicato come primo argomento.

ReleaseSemaphore

Sintassi di chiamata della funzione

ReleaseSemaphore (signalSemaphore)
A0

Scopo della funzione

Questa funzione svolge un compito opposto a quello svolto da ObtainSemaphore, dal momento che libera il semaforo di segnalazione del quale il task era entrato in possesso tramite la funzione ObtainSemaphore. Il semaforo diventa quindi disponibile per gli altri task. Se si è formata una lista di task in attesa di quel semaforo, il controllo passa al task che aveva chiamato ObtainSemaphore per primo, e gli altri salgono lungo la coda. Questo automatico scambio di possesso fra i task continua fino a quando la coda non è vuota. Questa coda è la lista individuata dal parametro ss_WaitQueue della struttura SignalSemaphore, e il numero di strutture SemaphoreRequest in essa presenti è indicato nel parametro ss_QueueCount.

Il task che ottiene l'accesso al semaforo, può nidificare l'accesso chiamando altre volte la funzione ObtainSemaphore. Il sistema ne tiene conto incrementando ogni volta il parametro ss_NestCount: il semaforo non risulta completamente liberato da un task fino a quando questo parametro non torna a zero (viene decrementato ogni volta che il task chiama ReleaseSemaphore). Quindi, il numero di chiamate a ReleaseSemaphore necessarie per liberare un semaforo è uguale al numero di chiamate effettuate a ObtainSemaphore. Dal momento che un semaforo di segnalazione è accessibile da qualsiasi sezione dei codici di un task, è ovvio concludere che non può costituire una forma di protezione interna.

Argomenti della funzione

signalSemaphore Indirizzo della struttura SignalSemaphore corrispondente al semaforo da rilasciare.

Discussione

L'Exec prevede sette funzioni che riguardano i semafori di segnalazione e due funzioni che riguardano la lista dei semafori di segnalazione. Queste funzioni sono illustrate nella spiegazione di AddSemaphore.

A ogni chiamata di ObtainSemaphore deve corrispondere una chiamata a ReleaseSemaphore. Il semaforo non torna libero fino a quando il numero di chiamate a ObtainSemaphore non eguaglia il numero di chiamate a ReleaseSemaphore. Questo vale per tutti i task che usano i semafori di segnalazione. Il parametro contatore di annidamento (ss_NestCount) della struttura SignalSemaphore conta il numero di volte che il semaforo è stato richiesto, ma non rilasciato, dal task che lo detiene.

Quando questo contatore si azzera, il semaforo è di nuovo libero, pronto per essere utilizzato da un altro task. Si osservi che il sistema va quasi sempre in crash se il numero dei rilasci supera quello delle chiamate nidificate. È importante notare la differenza fra gli scopi delle funzioni AddSemaphore e RemSemaphore, rispetto a quelli della coppia ObtainSemaphore e ReleaseSemaphore. AddSemaphore e RemSemaphore riguardano la lista dei semafori di segnalazione, mentre ObtainSemaphore e ReleaseSemaphore alterano il parametro ss_NestCount nella struttura SignalSemaphore.

ReleaseSemaphoreList

Sintassi di chiamata della funzione

ReleaseSemaphoreList (list)
A0

Scopo della funzione

Questa funzione svolge un compito opposto a quello svolto da ObtainSemaphoreList, dal momento che libera in un sol colpo tutti i semafori di segnalazione presenti nella lista indicata come argomento e che erano stati

ottenuti chiamando la funzione ObtainSemaphoreList. Si ricordi che il sistema non è preparato al tentativo di rilasciare più semafori di quelli che si erano ottenuti, e questo porterebbe al crash della macchina.

Argomenti della funzione

list Indirizzo della struttura List che intesta la lista dei semafori di segnalazione da liberare; le strutture SignalSemaphore in essa presenti sono concatenate fra loro tramite i primi elementi, ovvero sotto-strutture di tipo Node.

Discussione

L'Exec prevede sette funzioni che riguardano i semafori di segnalazione e due funzioni che riguardano la lista dei semafori di segnalazione. Queste funzioni sono illustrate nel corso della spiegazione di AddSemaphore.

ReleaseSemaphoreList, in coppia con ObtainSemaphoreList, nasce per aiutare il programmatore. Se diversi semafori di segnalazione sono stati ottenuti dal task e collocati in una lista, ReleaseSemaphoreList provvede a rilasciarli tutti in un sol colpo. Si noti che non è necessario che i semafori presenti nella lista siano stati ottenuti contemporaneamente tramite la funzione ObtainSemaphoreList; alcuni di essi possono essere stati ottenuti impiegando direttamente AttemptSemaphore o ObtainSemaphore.

RemDevice

Sintassi di chiamata della funzione

```
error = RemDevice (device)
D0                 A1
```

Scopo della funzione

Questa funzione rimuove un particolare dispositivo dalla lista dei dispositivi disponibili, ed è individuabile attraverso la struttura ExecBase. Una volta che il dispositivo è stato rimosso, non può essere più utilizzato finché non

torna disponibile tramite una chiamata alla funzione AddDevice, la quale provvede a inserire la struttura Device del dispositivo nella lista di sistema DeviceList. La funzione restituisce il valore zero se ha avuto successo; altrimenti restituisce un codice d'errore.

Argomenti della funzione

device Indirizzo della struttura Device che definisce il dispositivo da rimuovere.

Discussione

Ci sono otto funzioni dell'Exec che riguardano i dispositivi di I/O: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Si vedano anche le spiegazioni relative a queste funzioni.

RemDevice chiama la funzione Expunge del dispositivo, la quale si occupa di effettuare i necessari controlli e di rimuovere il dispositivo. Generalmente, questa funzione verifica se il dispositivo risulta ancora aperto, controllando il valore contenuto nel parametro lib_OpenCnt della struttura Library contenuta nella struttura Device. Se non risulta aperto da altri task, la funzione provvede a rimuoverlo dalla lista di sistema DeviceList. Se invece risulta ancora aperto, generalmente Expunge ne proroga la rimozione impostando il flag LIBF_DELEXP del parametro lib_Flags. In questo modo, quando viene chiamata la funzione CloseDevice e il controllo passa alla funzione Close del dispositivo, quest'ultima oltre a chiudere il dispositivo provvede anche a rimuoverlo tramite la funzione Expunge del dispositivo.

La struttura Device è identica alla struttura Library. Si veda la funzione AddLibrary.

RemHead

Sintassi di chiamata della funzione

```
node = RemHead (list)
DØ             AØ
```

Scopo della funzione

Questa funzione riceve come argomento l'indirizzo di una lista e rimuove il nodo che ne occupa la sommità. La funzione restituisce l'indirizzo della struttura Node rimossa. Se la lista è vuota, restituisce il valore zero.

Argomenti della funzione

list Indirizzo della struttura List dalla quale si desidera estrarre il nodo di testa.

Discussione

L'Exec possiede diverse funzioni previste per operare con nodi e liste doppie. Una completa esposizione di queste funzioni è contenuta nella spiegazione della funzione AddHead.

Le funzioni RemHead e RemTail si utilizzano in combinazione con AddHead e AddTail per gestire le liste. Quando si combina AddTail con RemHead per aggiungere e rimuovere nodi, si opera una gestione della lista di tipo FIFO (First In, First Out); AddTail aggiunge i nodi in fondo alla lista mentre RemHead li rimuove dalla sommità, in modo che il primo elemento inserito sia il primo a essere rimosso. Quando una lista viene gestita in questo modo viene definita "coda". Nell'Amiga, per esempio, tutte le liste che fanno capo alle message port sono delle code.

Quando invece si combinano AddHead e RemHead, si opera una gestione della lista di tipo LIFO (Last In, First Out); AddHead aggiunge nodi alla sommità della lista, e quindi RemHead rimuove sempre l'ultimo nodo aggiunto. Al contrario del caso precedente, nella gestione LIFO il primo elemento inserito è l'ultimo a essere rimosso. Quando una lista viene gestita in questo modo, viene definita "pila" (o catasta, o stack). Si possono usare queste funzioni anche in altre combinazioni. Per esempio, combinando AddTail e RemTail si opera ancora una gestione LIFO, ma la porta d'ingresso/uscita degli elementi corrisponde al fondo della lista.

RemIntServer

Sintassi di chiamata della funzione

RemIntServer (intNumber, interrupt)
DØ: Ø-4 A1

Scopo della funzione

Questa funzione rimuove il nodo di un interrupt server da una particolare catena di server, ma non rimuove il server dalla memoria. Se il server rimosso è l'ultimo della catena, gli interrupt per questa catena vengono disabilitati. Ogni catena di server è associata a uno e un solo bit di interrupt del chip Paula (da 0 a 14). La differenza fondamentale fra un interrupt server e un interrupt handler è che il primo appartiene a una comunità di routine di interrupt tutte associate allo stesso bit di interrupt, mentre il secondo è un'unica routine associata a un particolare bit di interrupt. Nel primo caso, quando si verifica un interrupt i server della catena ricevono il controllo uno dopo l'altro, iniziando da quello a più alta priorità per finire con quello a priorità più bassa. Ogni server ha la facoltà di decidere se i server di priorità inferiore riceveranno o meno il controllo, e deve sempre concludersi con l'istruzione RTS. Se prima di questa istruzione azzera il flag Z della CPU, il controllo viene ceduto al server successivo della catena, mentre se lo imposta, l'Exec non chiama i server successivi.

Argomenti della funzione

intNumber Bit (0-14) di interrupt del chip dedicato Paula (il chip 4703 dedicato alle periferiche e al suono).

interrupt Indirizzo della struttura Interrupt che definisce il server relativo al bit di interrupt indicato nel primo argomento. Questa struttura contiene in un parametro l'indirizzo del punto d'ingresso dei codici del server.

Discussione

Il sistema di interrupt dell'Amiga usa i registri del 68000 nel modo seguente:

1. Prima di cedere il controllo a un server, le routine dell'Exec salvano automaticamente i registri A0, A1, A4, A5, A6, D0 e D1.

2. All'interno dei codici di interrupt i registri A0, A1, A5, D0 e D1 possono essere modificati in qualsiasi modo senza che si debba ripristinare il loro contenuto originario.

3. I registri A4 e A6 devono essere considerati come registri speciali di sistema. Le convenzioni del sistema Exec prevedono che il registro A4 venga usato per puntare alle funzioni che si vogliono chiamare, e che il registro A6 venga usato per puntare alla struttura Library di qualsiasi libreria di sistema si voglia usare durante l'esecuzione della routine di interrupt.

4. Qualsiasi altro registro *deve* essere riportato al suo contenuto originario prima che il server esegua l'istruzione RTS. Se questo non accade, il comportamento del sistema non è prevedibile.

RemLibrary

Sintassi di chiamata della funzione

```
error = RemLibrary (library)
D0                  A1
```

Scopo della funzione

Questa funzione rimuove una particolare libreria dalla lista delle librerie disponibili; questa lista di sistema si chiama LibList ed è individuabile attraverso la struttura ExecBase. Una volta che la libreria è stata rimossa, non può essere utilizzata fino a quando non viene nuovamente resa disponibile tramite la funzione AddLibrary, la quale provvede a inserire la struttura Library della libreria nella lista di sistema LibList. La funzione restituisce il valore zero se si è svolta con successo; altrimenti restituisce un codice d'errore.

Argomenti della funzione

library Indirizzo della struttura Library che definisce la libreria da rimuovere.

Discussione

Nel sistema Amiga ci sono otto funzioni che riguardano le librerie: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Le librerie vengono identificate attraverso un nome e un numero di versione. Si vedano anche le spiegazioni relative a queste funzioni.

Il sistema Exec mantiene automaticamente una lista, LibList, delle librerie che sono state aggiunte al sistema e quindi rese disponibili. Ogni volta che si aggiunge una libreria tramite la funzione AddLibrary, la lista viene a contenere un nuovo nodo costituito dalla struttura Library che definisce la libreria. Allo stesso modo, ogni volta che si rimuove una libreria dal sistema tramite la funzione RemLibrary, dalla stessa lista viene estratto il nodo costituito dalla struttura Library che definiva quella libreria.

L'unico argomento previsto dalla funzione RemLibrary è l'indirizzo della struttura Library della libreria da rimuovere. In realtà la funzione RemLibrary non interagisce direttamente con la lista di sistema LibList. Invece chiama la funzione Forbid per sospendere lo scambio di controllo fra i task, chiama la funzione Expunge della libreria indicata dal codice chiamante (una delle funzioni standard che tutte le librerie devono possedere) e infine chiama la funzione Permit per riabilitare lo scambio di controllo fra i task. Tutto il lavoro di rimozione della libreria dalla lista di sistema LibList e dalla memoria è affidato alla funzione Expunge. Quindi, la rimozione di una libreria è affidata alla libreria stessa: il sistema non ha mezzi per rimuovere una libreria non predisposta per questa evenienza.

La funzione Expunge della libreria generalmente compie le seguenti operazioni. Verifica se la libreria risulta ancora aperta da qualcuno. Se lo è, imposta il flag LIBF_DELEXP del parametro lib_Flags per rimandare la rimozione della libreria, e restituisce il controllo. In questo modo, quando la funzione interna Close (chiamata da CloseLibrary) rileverà che la libreria non è più aperta per nessuno e che il flag LIBF_DELEXP è impostato, provvederà automaticamente a richiamare la funzione Expunge.

Se invece Expunge rileva che la libreria è completamente chiusa, chiama Remove per rimuovere la struttura Library dalla lista di sistema LibList, chiude le librerie che aveva aperto (generalmente la libreria DOS), libera la memoria occupata dal modulo di controllo della libreria, ed esce preoccupandosi di memorizzare in D0 il valore della segment list, in modo che l'AmigaDOS possa liberare tutta la memoria occupata dalla libreria.

Questi sono i comportamenti generali delle funzioni di libreria Expunge e Close, ed è opportuno tenerne conto quando si progetta una libreria.

Remove

Sintassi di chiamata della funzione

Remove (node)
A1

Scopo della funzione

Questa funzione rimuove una struttura Node da una lista a doppia concatenazione. La struttura Node è predisposta per mantenere i collegamenti di un nodo con quelli che lo seguono e lo precedono immediatamente. Tutte le strutture di sistema predisposte per essere elencate all'interno di liste iniziano con una struttura Node. Per un buon esito della funzione, è necessario che il nodo appartenga a una lista.

Argomenti della funzione

node	Indirizzo della struttura Node che si desidera rimuovere dalla lista.

Discussione

L'Exec ha diverse funzioni previste per operare con nodi e liste doppie. Una completa esposizione di queste funzioni è contenuta nella spiegazione della funzione AddHead.

Si noti che la funzione Remove non richiede d'indicare in quale lista si trova il nodo da rimuovere. Il nodo viene rimosso qualunque sia la lista a cui appartiene. Si deve sempre verificare che il nodo appartenga effettivamente a una lista.

RemPort

Sintassi di chiamata della funzione

RemPort (msgPort)
A1

Scopo della funzione

Questa funzione rimuove una message port pubblica dalla lista di sistema delle message port. Una volta che la message port è stata rimossa dalla lista, non è più pubblica, e pertanto non è più rintracciabile tramite il suo nome. Questo non significa però che la message port sia stata eliminata: RemPort, infatti, non disalloca la memoria occupata dalla message port.

Se la message port non si trova nella lista di sistema PortList, la funzione restituisce una condizione di errore. La message port deve quindi essere stata aggiunta alla lista delle message port di sistema con la funzione AddPort.

Argomenti della funzione

msgPort	Indirizzo della struttura MsgPort che definisce la message port.

Discussione

Nel sistema Amiga ci sono quattro routine del sistema Exec che riguardano esplicitamente la gestione delle message port: AddPort, FindPort, RemPort e WaitPort. Le prime tre riguardano esclusivamente le message port pubbliche, cioè quelle che sono state rese pubbliche tramite la funzione AddPort. WaitPort, invece, riguarda sia le message port pubbliche sia quelle private.

Tutti i task del sistema hanno la possibilità di avere un proprio insieme di message port, e di renderne pubbliche alcune. La differenza fra le message port private e quelle pubbliche è che le seconde sono rintracciabili da chiunque ne conosca il nome.

Ogni message port pubblica viene creata per conseguire un obiettivo specifico: normalmente per passare informazioni o segnali che sono vitali per lo svolgimento delle operazioni del task o del dispositivo. Se la lista di sistema

delle message port pubbliche si allunga eccessivamente, il numero delle message port può diventare troppo grande per essere gestito con efficienza dal sistema. È allora necessario rimuoverne alcune, operazione che se non altro aiuta a tenere sotto controllo ciò che accade nel sistema e libera memoria per altri scopi. Si può usare la funzione RemPort per rimuovere message port pubbliche e farle tornare private.

Per usare la funzione RemPort è necessario conoscere l'indirizzo della struttura MsgPort che definisce la message port da estrarre dalla lista di sistema. Se si conosce il suo nome, si può usare la funzione FindPort per ottenere questo indirizzo. Il nome della message port è l'unico argomento richiesto dalla funzione FindPort; si tratta della stringa individuata dal parametro ln_Name nella sotto-struttura Node contenuta nella message port. Questo parametro dev'essere sempre inizializzato prima di chiamare la funzione AddPort.

RemResource

Sintassi di chiamata della funzione

RemResource (resource)
A1

Scopo della funzione

Questa funzione rimuove una risorsa dalla lista delle risorse di sistema. Nessuna nuova apertura di questa risorsa può essere effettuata fino a quando essa non viene nuovamente aggiunta al sistema con la funzione AddResource.

Argomenti della funzione

resource	Indirizzo della struttura Resource da rimuovere; generalmente è il valore restituito dalla funzione OpenResource.

Discussione

Ci sono tre funzioni dell'Exec che sono direttamente interessate alla gestione delle risorse: AddResource, OpenResource e RemResource. Si noti che non esiste la funzione CloseResource. Si vedano anche le spiegazioni relative a queste funzioni. La struttura Resource è formalmente identica alla struttura Library. Si veda anche la funzione AddLibrary.

RemSemaphore

Sintassi di chiamata della funzione

RemSemaphore (signalSemaphore)
A1

Scopo della funzione

Questa funzione rimuove un semaforo dalla lista dei semafori di segnalazione del sistema. Dopo la rimozione, i tentativi di accedere al semaforo non avranno più successo.

Argomenti della funzione

signalSemaphore Indirizzo di una struttura SignalSemaphore opportunamente inizializzata.

Discussione

Nell'Exec ci sono sette funzioni che riguardano i semafori di segnalazione e due funzioni che riguardano la lista dei semafori di segnalazione. Queste funzioni sono illustrate nel corso della spiegazione di AddSemaphore.

I semafori di segnalazione sono mantenuti dal sistema in una particolare lista. Ogni volta che viene mandata in esecuzione AddSemaphore, la lista viene aggiornata per riflettere l'aggiunta di un altro semaforo di segnalazione contraddistinto da un particolare nome. Ogni volta che viene eseguita la funzione RemSemaphore, la lista dei semafori di segnalazione viene aggiornata

per riflettere la rimozione di un semaforo (attenzione: la funzione non disalloca la memoria occupata da quel semaforo).

La lista dei semafori di segnalazione viene gestita grazie alla sotto-struttura Node ss_Link nella struttura SignalSemaphore. Questa sotto-struttura Node fornisce il meccanismo per inserire e legare le nuove strutture SignalSemaphore nella lista dei semafori di segnalazione.

RemTail

Sintassi di chiamata della funzione

```
node = RemTail (list)
DØ             AØ
```

Scopo della funzione

Questa funzione rimuove il nodo di coda della lista indicata come argomento, e restituisce l'indirizzo della struttura Node corrispondente a quel nodo. Se la lista è vuota, restituisce il valore zero.

Argomenti della funzione

list	Indirizzo della struttura List che intesta la lista dalla quale si desidera estrarre il nodo di coda.

Discussione

Il sistema Exec ha diverse funzioni previste per operare con nodi e liste doppie. Una completa trattazione di queste funzioni è contenuta nella spiegazione della funzione AddHead.

Le funzioni RemHead e RemTail si usano in combinazione con AddHead e AddTail per gestire una lista a ordinamento speciale. Quando si combina AddTail con RemHead, si produce una lista di tipo FIFO (First In, First Out); AddTail aggiunge un nodo in fondo alla lista e RemHead rimuove il primo nodo dalla lista. Quando si combina AddHead con RemHead, si produce una lista di tipo LIFO (Last In, First Out) che è equivalente a una struttura di dati definita pila (o catasta, o stack); AddHead aggiunge un nodo in testa alla lista e

RemHead rimuove il primo nodo della lista. Si possono usare queste funzioni anche in combinazioni diverse per gestire le liste in altro modo. Per esempio, combinando AddTail e RemTail si produce ancora una lista di tipo LIFO, nella quale però i nodi vengono aggiunti e rimossi dal fondo della lista.

RemTask

Sintassi di chiamata della funzione

RemTask (taskCS)
A1

Scopo della funzione

Questa funzione rimuove un task dal sistema. Le risorse di memoria impiegate dal task dovrebbero essere liberate prima di chiamarla. Quando un task ne rimuove un altro, riottiene ovviamente il controllo; il task rimosso, invece, viene interrotto e non può più riottenere il controllo; la sua struttura Task non è infatti più presente in nessuna lista di sistema per la gestione dei task. Questo brusco modo d'interrompere l'esecuzione di un task può causare dei problemi, ed è quindi sconsigliato. È invece molto meglio che il task sia in grado di procedere autonomamente alla propria rimozione, chiamando appunto la funzione RemTask dall'interno della sua routine di chiusura. In questo secondo caso, tutti i codici successivi non possono essere eseguiti, quindi RemTask dev'essere chiamata come ultima operazione.

A prescindere dal fatto che il task da rimuovere sia lo stesso task che effettua la chiamata, la funzione RemTask libera anche qualsiasi pool di memoria che risulti agganciato al task tramite il parametro tc_MemEntry della struttura Task che lo definisce. Se il task indicato non risulta presente in nessuna lista di sistema dei task, il risultato è una condizione di errore.

Argomenti della funzione

taskCS — Indirizzo della struttura Task che definisce il task da rimuovere. Se questo argomento vale zero, la funzione rimuove la struttura Task che definisce il task chiamante.

Discussione

Ci sono quattro routine dell'Exec che riguardano specificamente la gestione dei task: AddTask, FindTask, RemTask e SetTaskPri. Si vedano anche le spiegazioni relative a queste funzioni.

Come inibire l'esecuzione dei task

Ci sono due metodi per inibire l'esecuzione dei task. Primo, tramite la funzione SetTaskPri si può assegnare al task una bassa priorità, così che i task a priorità più alta gli sottraggano la maggior parte dei cicli di CPU disponibili. Questo metodo si basa sul fatto che per avere accesso alla CPU un task a bassa priorità deve attendere che i task a priorità più alta entrino tutti in stato di attesa. In questo caso, il primo task fra quelli che hanno la priorità appena inferiore ottiene il controllo.

Secondo, si possono usare le funzioni di sistema Forbid e Permit. Queste routine sono in genere impiegate dal software sistema, e servono per disabilitare e riabilitare lo scambio di controllo fra i task. Chiamando queste due funzioni, si può sospendere la gestione multitasking durante l'esecuzione di un dato segmento di codice. Conviene usare Forbid e Permit anche nel caso che un task debba modificare strutture di dati accessibili da altri task.

ReplyMsg

Sintassi di chiamata della funzione

ReplyMsg (message)
A1

Scopo della funzione

Questa funzione serve per restituire al mittente un messaggio pervenuto nella propria message port. ReplyMsg di solito viene chiamata quando il task destinatario di un messaggio ha finito di elaborarlo e vuole restituirlo al mittente, così che il messaggio possa essere riutilizzato, disallocato o altro. Il mittente di un messaggio è in genere il task che ha allocato il messaggio e lo ha inizializzato. È questa la fase in cui il mittente deve decidere se richiedere la successiva restituzione del messaggio, cioè se memorizzare o meno nel parametro mn_ReplyPort l'indirizzo della propria reply port. Infatti, dopo che ha

chiamato la funzione PutMsg, non deve più accedere al messaggio per nessuna ragione, fino a quando non lo riottiene nella sua reply port.

Il destinatario, a sua volta, dopo aver ricevuto il messaggio lo deve leggere, eventualmente modificarne i parametri per formulare una risposta, e infine chiamare la funzione ReplyMsg. Da questo momento in avanti il messaggio non è più di sua proprietà, e quindi non deve più accedervi per nessuna ragione. Il destinatario può chiamare la funzione ReplyMsg anche quando all'interno del messaggio non è indicato alcun mittente, cioè quando il paramentro mn_ReplyPort contiene un indirizzo nullo; in questo caso, infatti, la funzione restituisce semplicemente il controllo.

Argomenti della funzione

message Indirizzo della struttura Message che intesta il messaggio. Normalmente, ReplyMsg imposta il parametro ln_Type del messaggio a NT_REPLYMSG, ma se si tratta di un messaggio in cui il parametro mn_ReplyPort vale zero, memorizza nel parametro ln_Type il valore NT_FREEMSG.

Discussione

Nel sistema Amiga, le routine dell'Exec che riguardano esplicitamente i messaggi sono tre: PutMsg, GetMsg e ReplyMsg. Queste routine gestiscono i messaggi in partenza e in arrivo alle message port dei task. Siccome i messaggi usano le message port, è importante anche tener presente il ruolo delle routine di gestione delle message port. Queste routine sono AddPort, FindPort, ReplyPort e WaitPort. Si vedano anche le spiegazioni relative a queste routine.

I messaggi possono essere scambiati tra le message port di un solo task o tra message port appartenenti a task diversi. Nel primo caso il task mittente e quello destinatario coincidono. Nel secondo caso mittente e destinatario rimangono distinti.

Ogni task può disporre di una reply port, oltre alla message port che impiega per ricevere messaggi dagli altri task. Lo scopo della reply port è di ricevere in risposta i messaggi inoltrati; quando un messaggio torna alla reply port, il mittente ha la certezza che sia giunto a destinazione e che sia stato esaminato.

Il messaggio restituito alla reply port del mittente può anche costituire una risposta formulata dal destinatario, e quindi può contenere importanti informazioni per la prosecuzione dell'attività. Molto spesso capita che il task mittente di un messaggio non possa proseguire nelle sue attività fino a quando non riceve risposta dal destinatario. In questo caso, il mittente entra in attesa della risposta, per non appropriarsi inutilmente di cicli della CPU.

Se si considerano tutti i task, tutte le message port e le reply port create

da questi task, si inizia a capire quanto possono diventare complesse le interazioni fra i task.

Il meccanismo dei messaggi è molto flessibile. Consente a uno stesso task di creare e inoltrare nel sistema un numero indefinito di messaggi, di controllare che vengano restituiti in maniera asincrona, cioè senza entrare in attesa, o sincrona, cioè entrando in attesa.

Si ricordi infine che qualsiasi message port o reply port può essere predisposta per causare un evento quando riceve un messaggio nella sua coda.

SendIO

Sintassi di chiamata della funzione

SendIO (iORequest)
A1

Scopo della funzione

Questa funzione serve per inoltrare una richiesta di I/O a un dispositivo dell'Amiga. La richiesta dev'essere costituita almeno da una struttura IORequest opportunamente inizializzata, anche se in genere questo tipo di struttura costituisce solo il primo elemento di più estese stutture di I/O, come la struttura estesa standard IOStdReq, o la struttura estesa non standard IOExtTD. La struttura IORequest è in pratica composta da una struttura Message d'intestazione, seguita da una serie di parametri che costituiscono un messaggio base comune a tutti i dispositivi di I/O dell'Amiga. Le versioni estese di questa struttura fanno poi seguire al messaggio base altri parametri che variano a seconda del dispositivo a cui è diretta la richiesta di I/O. Quindi, la struttura IORequest è a tutti gli effetti un messaggio predefinito.

Esiste comunque un'importante differenza nella gestione delle richieste di I/O rispetto alla generica gestione dei messaggi. Le richieste di I/O oltre a contenere l'indirizzo della reply port del mittente, devono anche contenere nei parametri io_Device e io_Unit l'indirizzo della struttura Device del dispositivo e l'identificatore dell'unità del dispositivo indirizzata. Queste informazioni identificano univocamente il destinatario del messaggio. Per questa loro particolare natura, pur essendo messaggi a tutti gli effetti, le richieste di I/O non vengono inoltrate con la funzione PutMsg, ma con le funzioni SendIO (asincrona), DoIO (sincrona) e BeginIO (sincrona o asincrona), come viene ampiamente spiegato nel secondo volume.

La funzione SendIO permette d'inoltrare le richieste di I/O in modo

asincrono. Il task riottiene subito il controllo anche se il dispositivo non ha nemmeno iniziato a elaborare la richiesta di I/O. Una particolarità di questa funzione, comune anche alla funzione DoIO, è che azzera il parametro io_Flags della struttura di I/O, e quindi se il task desidera inoltrare una richiesta con alcuni flag impostati, deve utilizzare necessariamente la funzione BeginIO, che accede direttamente all'interno del dispositivo e non fa quindi parte della libreria Exec.

Argomenti della funzione

iORequest Indirizzo della struttura IORequest in parte inizializzata dalla funzione OpenDevice (parametri io_Device e io_Unit).

Discussione

Ci sono otto funzioni dell'Exec che riguardano direttamente i dispositivi di I/O: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Ciascuna di queste funzioni opera con i dispositivi. Si vedano anche le spiegazioni relative a queste funzioni.

In generale, le richieste ai dispositivi di I/O si dividono in due categorie. Nella prima ricadono le richieste di I/O che devono essere soddisfatte perché il task possa proseguire nella propria esecuzione. Queste richieste, definite sincrone, devono essere inoltrate tramite la funzione DoIO, la quale dopo aver effettuato l'invio mette il task in attesa della risposta.

Nella seconda categoria ricadono invece le richieste di I/O asincrone, cioè quelle la cui risposta non è così urgente da non permettere al task di proseguire nell'esecuzione di altre mansioni. Tutte le richieste di questo tipo devono essere inoltrate tramite la funzione SendIO, la quale consente al task di svolgere altre funzioni dopo l'invio della richiesta di I/O.

Oltre a DoIO e SendIO, per comunicare con i dispositivi di I/O i task possono impiegare BeginIO e AbortIO, due funzioni previste dalla libreria di ogni dispositivo. La prima non ha un corrispondente nella libreria Exec, e quindi chiamandola si cede direttamente il controllo alle routine del dispositivo, mentre la seconda è presente anche nella libreria Exec, e ha l'unica funzione di effettuare la chiamata all'omonima funzione presente nella libreria del dispositivo.

BeginIO è necessaria quando si desidera impostare uno o più flag nella richiesta di I/O. SendIO, infatti, azzera sempre tutti i flag, mentre DoIO imposta sempre il flag IOF_QUICK per cercare di far elaborare la richiesta nella modalità veloce, il cosiddetto QuickIO.

Quando il task riottiene il controllo da SendIO, deve prepararsi a ricevere nella sua reply port la richiesta di I/O che ha inoltrato, ovvero la risposta. Solo all'arrivo della risposta il task può ritenere che la richiesta sia stata

completamente elaborata. Questa interazione task-dispositivo viene detta asincrona. Per verificare se la richiesta è stata elaborata e restituita, il task può chiamare periodicamente la funzione CheckIO, oppure entrare in attesa tramite la funzione WaitIO.

Quando il task riottiene il controllo da DoIO, può ritenere che la richiesta di I/O sia stata completamente elaborata dal dispositivo, e non deve sondare la sua reply port in quanto questa operazione è già stata compiuta da DoIO. Questa funzione cerca sempre di ottenere il QuickIO, ma se il dispositivo decide di elaborare la richiesta normalmente è la funzione stessa a preoccuparsi di entrare in attesa della risposta alla reply port del task.

Quando il task riottiene il controllo da DoIO, se aveva richiesto il QuickIO e il flag risulta ancora impostato, significa che la modalità veloce di esecuzione è stata accolta dal dispositivo. In questo caso, la risposta non viene accodata alla reply port del task, dal momento che il solo fatto che il task abbia riottenuto il controllo significa che la richiesta è stata completamente elaborata. Se invece il task ha richiesto il QuickIO con BeginIO, ma rileva che il flag IOF_QUICK è stato azzerato, significa che la richiesta è stata accodata all'unità, e che verrà restituita alla reply port del task solo quando sarà elaborata.

Il vantaggio del QuickIO consiste nel fatto che (se viene accolto) la richiesta di I/O viene soddisfatta immediatamente, anche se altre richieste la precedono nella coda all'unità indirizzata, permettendo così la prosecuzione dell'elaborazione. Spetta al dispositivo decidere quando il QuickIO è possibile; in caso d'impossibilità, la richiesta di I/O viene accodata normalmente, come se fosse stata inviata con SendIO oppure con BeginIO e IOF_QUICK azzerato.

SetExcept

Sintassi di chiamata della funzione

```
oldSignals = SetExcept (newSignals, signalMask)
D0                      D0          D1
```

Scopo della funzione

Questa funzione serve per modificare lo stato dei bit di exception nel parametro tc_SigExcept del task. La modifica avviene nel seguente modo. Il task deve indicare nell'argomento signalMask la maschera dei bit interessati dalla nuova impostazione; i bit a 1 di questa maschera indicano alla funzione quali sono i bit del parametro tc_SigExcept su cui intervenire. Nell'argomento newSignals deve invece indicare i nuovi valori (1 o 0) assegnati ai bit di exception indicati dalla maschera; in questo argomento la funzione prenderà

in considerazione solo gli stati dei bit che possiedono il corrispondente bit della maschera signalMask impostato a 1, cioè solo i bit interessati al cambiamento di stato. Quindi, tramite questi due argomenti il task riesce ad agire su particolari insiemi bit del parametro tc_SigExcept e a impostarli a nuovi valori. Per quanto riguarda invece i bit del parametro tc_SigExcept sui quali la funzione non agisce (quelli che nel parametro signalMask risultano azzerati), essi mantengono i valori orginali che avevano già prima che la funzione venisse eseguita.

Riassumendo, SetExcept consente di modificare lo stato di alcuni particolari bit del parametro tc_SigExcept preservando lo stato degli altri. Il risultato è una nuova configurazione di bit nel parametro tc_SigExecept.

Compiute queste operazioni, la funzione verifica se nel parametro tc_SigRecvd esistono bit impostati a 1, indicanti che il task ha ricevuto dei segnali. Se qualcuno di questi bit è fra quelli cha causano exception, cioè risulta impostato anche nel parametro tc_SigExcept, la funzione imposta il flag TC_EXCEPT nel parametro tc_Flags della struttura Task e compie due operazioni diverse a seconda che il task risulti in attesa o in esecuzione, tenendo presente che il primo caso si può verificare solo se la funzione SetExcept viene eseguita dal sistema o da una routine di interrupt del task.

1. Se il task risulta in stato di attesa, la struttura Task viene estratta dalla lista di sistema TaskWait e viene inserita nella lista TaskReady, in modo che il task rientri fra quelli eseguibili. Successivamente, se il task risulta essere il primo della lista, viene eseguita la funzione Reschedule per causare uno scambio di controllo fra i task e cedergli così il controllo della CPU: il task torna attivo, ma avendo il flag TC_EXCEPT impostato entra in exception, cioè viene eseguita la routine di exception il cui indirizzo è contenuto nel parametro tc_ExceptCode della struttura Task. Se invece il task non risulta il primo della lista, la funzione restituisce il controllo al codice chiamante e il task entrerà in exception solo quando verrà il suo turno di CPU.

2. Se invece il task risulta eseguibile, SetExcept non fa altro che chiamare la funzione Reschedule: il task torna attivo, ma avendo il flag TC_EXCEPT impostato entra anche in questo caso in exception.

   Può invece capitare che nel parametro tc_SigRecvd alcuni bit siano impostati, ma che nessuno sia un bit di exception. Allora la funzione va a verificare se il task si trova in attesa almeno di uno di questi bit. Se non lo è, la funzione restituisce semplicemente il controllo al codice chiamante, mentre se risulta in attesa di uno di questi bit, la funzione rimuove la struttura Task dalla lista di sistema TaskWait, la inserisce nella lista TaskReady, e se il task è il primo della lista chiama la funzione Reschedule per cedergli il controllo; se non è il primo della lista riceverà il controllo al suo prossimo turno di CPU. Al contrario di quanto accade quando i segnali ricevuti causano exception, in questo caso il task riceve il controllo esattamente dal punto in cui era entrato in attesa.

Infine, la funzione restituisce sempre in oldSignals la configurazione dei bit presente nel parametro tc_SigExcept prima che venisse modificata. A questo proposito, se il task chiama SetExcept indicando entrambi gli argomenti a zero, il contenuto del parametro tc_SetExcept non viene alterato, ma semplicemente restituito al codice chiamante.

Argomenti della funzione

newSignals Argomento da 32 bit nel quale gli stati dei bit interessati all'operazione (si veda il prossimo argomento) vengono copiati nei corrispondenti bit del parametro tc_SigExcept. Gli stati relativi ai bit non interessati all'operazione vengono ignorati.

signalMask Maschera da 32 bit nella quale i bit a 1 sono quelli che nel parametro tc_SigExcept sono interessati all'operazione, cioè quelli che devono assumere gli stati dei corrispondenti bit dell'argomento newSignals. I bit che in questa maschera risultano azzerati corrispondono ai bit del parametro tc_SigExcept non interessati all'operazione, e che pertanto mantengono i loro valori originali.

Discussione

Vediamo un esempio che chiarisca come intervengono i due argomenti della funzione e il valore contenuto nel parametro tc_SigExcept, nella creazione della nuova configurazione dei bit di exception. Supponendo di prendere in considerazione soltanto gli otto bit meno significativi, se il parametro tc_SigExcept contiene il valore %10100100 e il nostro task desidera modificare i bit 0, 1, 4, 5 in modo che i primi tre siano a 1 e il bit 5 sia a zero, occorre chiamare la funzione SetExcept indicando:

```
newSignals = %11011011
signalMask = %00110011
```

In questo modo otteniamo i seguenti risultati:

```
oldSignals =    %10100100
tc_SigExcept = %10010111
```

Si noti che nell'argomento newSignals sono stati volutamente impostati a 1 anche alcuni dei bit non coinvolti nell'operazione, cioè azzerati nel parametro signalMask, per mostrare la loro ininfluenza.

Le exception appartengono a due categorie. Exception del 68000, che vengono eseguite nel modo supervisor, e quelle private del task, che vengono eseguite nel modo user. Le exception del 68000 sono raggruppate a loro volta in tre categorie:

1. Reset, errore d'indirizzamento, errore sul bus.

2. Trace, interrupt, istruzione illegale, violazione di privilegio nell'esecuzione delle istruzioni.

3. Istruzioni TRAP, TRAPV e CHK, insieme alla condizione di divisione per zero.

Una violazione di privilegio è il tentativo di eseguire un'istruzione privilegiata quando la CPU è in modo user; un'istruzione viene infatti definita "privilegiata" se è eseguibile solo in modo supervisor.

L'istruzione TRAP può causare anche exception, e consente dal modo user di cedere il controllo a uno dei 16 indirizzi di memoria contenuti all'interno degli autovettori che vanno dal numero 32 (indirizzo 0x00000080) al numero 47 (indirizzo 0x000000BC). Questi 16 autovettori sono contenuti nel primo kilobyte di memoria nel caso del 68000, e nel kilobyte indirizzato dal registro VBR nel caso delle CPU superiori. Quest'area di memoria è la tavola dei vettori di exception considerata dalla CPU.

Le routine di trap memorizzate in questi 16 indirizzi dovrebbero essere brevi, così da consentire una rapida elaborazione e restituzione del controllo al task in esecuzione.

La discussione che segue riguarda le exception del 68000. Le elaborazioni delle exception del 68000 si svolgono in modo supervisor in quattro fasi:

1. Il 68000 esegue internamente una copia del registro di stato. Dopo la copia, nel registro di stato vengono impostati a 1 il bit del modo supervisor (S) e a zero il bit del modo trace (T), rispettivamente per forzare il 68000 a funzionare in modo supervisor, e per disabilitare la modalità d'esecuzione "trace", qualora fosse attiva al verificarsi dell'exception, in modo che la routine di gestione dell'handler possa completarsi alla normale velocità d'esecuzione.

2. Il 68000 determina il numero dell'autovettore relativo all'exception. Questo numero viene poi usato internamente per generare l'indirizzo dell'autovettore nel quale è contenuto l'indirizzo della routine preposta alla gestione dell'exception. Il 68000 prevede 16 autovettori per le exception provocate dall'istruzione TRAP.

3. Il 68000 salva lo stato della CPU: il program counter e il valore del registro di stato vengono collocati nel supervisor stack.

4. Il 68000 preleva l'indirizzo contenuto nell'autovettore precedentemente identificato, e lo memorizza nel program counter. Il 68000 riprende poi

la decodifica e l'esecuzione delle istruzioni a partire da quella nuova posizione nello spazio indirizzabile. Procede fino a quando non incontra l'istruzione RTE (ritorno da exception).

Le exception private dei task

L'Amiga gestisce le exception private dei task nel modo user. Si può pensare a un'exception privata di un task come a un task all'interno di un task. Il meccanismo di sistema usato per elaborare le exception preserva l'intero insieme dei registri che il 68000 impiega con il primo task, affinché possano essere ancora utilizzabili quando il task riacquista il controllo della CPU, cioè quando la routine di exception ha terminato il suo lavoro. Quando si verifica un'exception di questo tipo, la CPU forza in ogni caso il modo user, e quindi il task in esecuzione deve possedere uno user stack sufficientemente capiente per rispondere alle necessità di stack dell'exception. Alcune routine di exception richiedono infatti molto spazio.

La routine di exception dev'essere pronta a ricevere nel registro A1 l'indirizzo della propria area dati, e nel registro D0 la maschera dei bit di exception relativi ai segnali che hanno causato l'exception. Grazie a quest'ultima informazione, la routine può rilevare la causa dell'exception. Il sistema inoltre organizza lo user stack nel seguente modo (partendo dall'indirizzo inferiore verso quello più alto):

User stack

PC (program counter)
SR (registro di stato a 16 bit)
Da D0 a D7
Da A0 a A7
(il resto dello stack)

Una volta che si è entrati nell'elaborazione dell'exception, si deve innanzitutto impostare il registro D0 con la maschera di eventi di exception del task. Quest'operazione imposta i bit di segnale che causeranno nuove exception una volta che il controllo tornerà al task.

Infine, è necessaria l'esecuzione dell'istruzione RTS del 68000 per completare l'elaborazione dell'exception. Il task riottiene il controllo dal punto in cui era stato interrotto dal segnale di exception.

SetFunction

Sintassi di chiamata della funzione

```
oldFunc = SetFunction (library, funcOffset, funcEntry)
DØ                     A1       AØ.W        DØ
```

Scopo della funzione

SetFunction consente di modificare, all'interno della tavola dei vettori di salto di una libreria, l'indirizzo contenuto nel vettore relativo a una particolare funzione. Quest'operazione serve per sostituire una funzione di una libreria con un'altra. SetFunction si preoccupa di effettuare l'intervento in modo che in seguito il sistema non dichiari non valida la libreria, cioè aggiorna opportunamente il checksum della libreria. Se tutto va a buon fine, il vettore che si trova all'offset funcOffset nella tavola dei vettori della libreria viene a contenere un'istruzione di salto che cede il controllo alla nuova funzione; l'indirizzo memorizzato in questa istruzione jmp $xxxxxx è ovviamente quello indicato dal task come argomento funcEntry. SetFunction restituisce l'indirizzo della funzione che viene estromessa dalla libreria.

Argomenti della funzione

library — Indirizzo della struttura Library che definisce la libreria nella cui tavola dei vettori di salto desideriamo intervenire.

funcOffset — Offset del vettore di salto che si desidera modificare. Dev'essere un valore negativo relativo all'indirizzo base della libreria.

funcEntry — Indirizzo della nuova funzione che dev'essere inserita nella libreria.

Discussione

Ci sono otto funzioni che riguardano le librerie nel sistema Amiga: AddLibrary, CloseLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLi-

brary, SetFunction e SumLibrary. Le librerie vengono identificate con i loro nomi e numeri di versione. Si vedano anche le spiegazioni relative a queste funzioni.

I vettori di libreria

Ciascuna libreria deve sempre avere almeno le quattro funzioni standard OPEN, CLOSE, EXPUNGE e EXTFUNC, e le istruzioni di salto a queste funzioni devono necessariamente trovarsi nei primi quattro vettori della tavola della libreria. Quindi, la funzione Open ha offset -6, Close ha offset -12, Expunge ha offset -18 e Extfunc ha offset -24. Al di fuori di questi vettori, gli altri possono individuare in memoria qualsiasi tipo di funzione senza alcun vincolo.

La funzione Open viene chiamata automaticamente come parte della funzione OpenLibrary quando si apre una libreria. Qualunque task può chiamare la funzione OpenLibrary, e quindi la libreria può risultare aperta simultaneamente da diversi task.

Tutti i task hanno accesso ai contenuti di una libreria, tanto alle funzioni quanto al modulo di controllo (il modulo di controllo è costituito dalla tavola dei vettori, più la struttura Library, più l'area dati). Ogni volta che un task apre la libreria, la funzione OPEN della libreria provvede a incrementare il parametro lib_OpenCnt della struttura Library conservata nella libreria. Ciò permette di sapere in ogni istante quanti task possiedono la libreria aperta.

La funzione Close viene mandata in esecuzione quando un task chiama la funzione CloseLibrary per chiudere la libreria. Generalmente, decrementa il valore contenuto nel parametro lib_OpenCnt e verifica se è arrivato a zero. Se è ancora diverso da zero, restituisce il controllo, perché evidentemente ci sono ancora dei task che stanno accedendo alla libreria. Se invece il parametro vale zero, significa che tutti i task hanno chiuso la libreria. A questo punto la funzione controlla lo stato del flag LIBF_DELEXP nel parametro lib_Flags della struttura Library: se è impostato chiama automaticamente la funzione Expunge per rimuovere dal sistema l'intera libreria, mentre se non lo è restituisce il controllo. Questo flag indica che è stata prorogata la chiusura della libreria, e generalmente viene impostato dalla funzione Expunge, chiamata dalla funzione RemLibrary, quando rileva che non è possibile rimuovere la libreria in quanto c'è ancora almeno un task che la tiene aperta.

La funzione Expunge in pratica è l'antitesi della funzione MakeLibrary e della funzione AddLibrary. Infatti, rimuove la struttura Library dalla lista di sistema delle librerie e poi libera tutta la memoria occupata dal modulo di controllo e dalle funzioni della libreria.

La funzione Extfunc, infine, dice al sistema come uscire da una delle funzioni della libreria.

L'uso di SetFunction per cambiare i vettori di libreria

Nell'argomento "library" il task deve indicare l'indirizzo della struttura Library relativa alla libreria nella quale desidera cambiare un vettore di salto.

Si ricordi che si tratta dell'indirizzo base della libreria. In memoria individua il primo parametro della struttura Library, ed è preceduto dalla tavola dei vettori di salto.

Nell'argomento funcOffset il task deve indicare l'offset (rispetto all'indirizzo base della libreria) al quale corrisponde la posizione del vettore di salto della funzione da modificare. Dev'essere necessariamente un numero negativo e multiplo di sei (sei sono infatti i byte occupati da un vettore di salto). L'offset deve anche essere, in valore assoluto, inferiore o uguale al valore contenuto nel parametro lib_NegSize della struttura Library, impostato durante la creazione del modulo di controllo e indicante il limite inferiore del modulo stesso come offset negativo rispetto all'indirizzo base della libreria.

Nell'argomento funcEntry il task deve indicare l'indirizzo della funzione che prenderà il posto di quella vecchia.

SetFunction, oltre che sostituire la funzione indicata con un'altra, ricalcola automaticamente il checksum della libreria, e quindi non è necessario che il task chiami anche la funzione SumLibrary.

SetIntVector

Sintassi di chiamata della funzione

```
oldInterrupt = SetIntVector (intNumber, interrupt)
D0                           D0: 0-4    A1
```

Scopo della funzione

Questa funzione fornisce un meccanismo per impostare i vettori di interrupt del sistema, cioè per associare nuovi handler a particolari interrupt. Più precisamente, la funzione svolge le seguenti operazioni.

Tramite intNumber identifica nella struttura ExecBase, all'interno dell'array IntVects, la struttura IntVector relativa al numero di interrupt indicato dal task (IntVects raccoglie 16 strutture IntVector, una per ogni livello di interrupt riconosciuto dall'Amiga; queste strutture possono definire sia handler sia catene di server, a seconda del tipo di interrupt). Successivamente, se l'argomento interrupt indicato dal task vale zero, imposta a -1 i parametri iv_Data e iv_Code della struttura IntVector. Se invece il task indica nell'argomento interrupt l'indirizzo di una struttura Interrupt opportunamente inizializzata per definire un handler, SetIntVector memorizza questo indirizzo nel parametro iv_Node della struttura IntVector, e copia i valori contenuti nei parametri is_Data e is_Code della struttura Interrupt rispettivamente nei parametri iv_Data e iv_Code della struttura IntVector. Infine, la funzione

restituisce sempre l'indirizzo della struttura Interrupt associata a quel numero di interrupt, cioè l'indirizzo che era contenuto nel parametro iv_Node della struttura IntVector prima che il task chiamasse SetIntVector.

Quando si verifica un interrupt, le tre linee di collegamento IPL0, IPL1 e IPL2 della CPU assumono una combinazione di stati che indica alla CPU la richiesta di interrupt e la sua priorità (la CPU riconosce sette livelli di priorità; la priorità 0 corrisponde all'assenza di interrupt, mentre la priorità 7 è quella più alta). Se la priorità della richiesta avanzata è maggiore del livello di priorità in cui la CPU sta funzionando, questa sospende le operazioni che sta svolgendo, abilita il modo supervisor, salva lo stato di tutti i registri, e cede il controllo all'indirizzo contenuto nell'autovettore corrispondente alla priorità di interrupt della richiesta (questi autovettori sono sette e si trovano nella tavola delle exception, a partire dall'indirizzo 0x00000064). La routine di interrupt che riceve il controllo trasforma il livello di priorità della CPU nel corrispondente livello di interrupt considerato dall'Amiga (per fare questa trasformazione accede ai registri di interrupt del chip Paula, per scoprire l'origine dell'interrupt). Infine, localizza nella struttura ExecBase, all'interno dell'array IntVects, la struttura IntVector relativa a quel livello di interrupt, e cede il controllo alla routine di interrupt indicata dal parametro iv_Code.

Per quanto riguarda i registri della CPU, la routine di interrupt che riceve il controllo deve considerare quanto segue:

D0	Alterabile, in ingresso non contiene un valore significativo.
D1	Alterabile, in ingresso (salvo per l'interrupt di livello 15) contiene una maschera di bit ottenuta effettuando l'operazione logica AND fra il registro INTENAR (registro a sola lettura i cui bit impostati indicano gli interrupt abilitati) e il registro INTREQR (registro a sola lettura i cui bit impostati identificano le richieste di interrupt), entrambi del chip Paula; il bit a 1 di questa maschera identifica il livello di interrupt.
A0	Alterabile, in ingresso (salvo per l'interrupt di livello 15) contiene l'indirizzo base dei registri dei chip dedicati dell'Amiga (0xDFF000).
A1	Alterabile, in ingresso contiene l'indirizzo del segmento dati di interrupt, lo stesso indirizzo contenuto nel parametro iv_Data della struttura IntVector.
A5	Alterabile, in ingresso contiene l'indirizzo della routine di interrupt, lo stesso indirizzo contenuto nel parametro iv_Code della struttura IntVector.
A6	Alterabile, in ingresso contiene l'indirizzo base della libreria Exec.

Tutti gli altri registri devono essere preservati.

Argomenti della funzione

intNumber Il numero di interrupt (0-15). Il chip Paula gestisce i livelli dallo 0 al 14, mascherandoli o meno a seconda di quanto è indicato nei suoi registri. Il livello 15, invece, giunge direttamente alla CPU (corrisponde alla priorità 7) dalla porta d'espansione dell'Amiga, e pertanto non è mascherabile.

interrupt Indirizzo della struttura Interrupt che definisce il nuovo handler per il livello di interrupt indicato da intNumber, cioè che indica il suo punto d'ingresso e l'indirizzo del segmento dati. È una buona idea dare un nome alla struttura Node contenuta nella struttura Interrupt, così gli altri task del sistema possono identificare quale task detiene il controllo dell'interrupt.

Discussione

Nella routine di un handler sono generalmente svolte in progressione le operazioni seguenti:

1. La routine disabilita il particolare interrupt del chip Paula a cui è associata, azzerando il relativo bit di interrupt nel registro INTENA del chip. In questo modo, durante l'elaborazione dell'interrupt, Paula ignorerà tutte le richieste dello stesso livello di interrupt che l'hardware potrebbe nel frattempo inoltrare.

2. Azzera opzionalmente il bit di interrupt che indica la richiesta nel registro INTREQ del chip Paula. Questa operazione può anche essere svolta quando la routine di interrupt volge al termine.

3. Esegue i propri codici di gestione.

4. Attiva nuovamente l'interrupt reimpostando il relativo bit di abilitazione nel registro INTENA, in modo che Paula torni a considerarne le richieste.

Si noti che le fasi 2 e 4 vengono svolte per evitare il rischio che il codice cada in un ciclo senza uscita, elaborando ripetizioni dello stesso interrupt che si verificano a tempi troppo ravvicinati. Questo previene il verificarsi di overflow nel supervisor stack.

Si veda la spiegazione della funzione AddIntServer per un'illustrazione dei bit di interrupt del chip Paula e una spiegazione del loro significato.

SetSignal

Sintassi di chiamata della funzione

```
oldSignals = SetSignal (newSignals, signalMask)
DØ                      DØ          D1
```

Scopo della funzione

Questa funzione serve per modificare lo stato dei bit di segnale nel parametro tc_SigRecved del task. La modifica avviene nel modo seguente. Il task deve indicare nell'argomento signalMask la maschera dei bit interessati alla nuova impostazione; i bit a 1 di questa maschera indicano alla funzione i bit del parametro tc_SigRecvd su cui intervenire. Nell'argomento newSignals il task deve invece indicare i nuovi valori (1 o 0) che devono assumere i bit di segnale indicati nella maschera; in questo argomento la funzione prenderà in considerazione solo gli stati dei bit che possiedono il corrispondente bit della maschera signalMask impostato a 1, cioè solo i bit interessati dal cambiamento di stato. Quindi, tramite questi due argomenti il task riesce ad agire su un particolare insieme di bit del parametro tc_SigRecvd e a impostarli a nuovi valori. I bit del parametro tc_SigRecvd sui quali la funzione non agisce, invece, (quelli che nel parametro signalMask risultano azzerati) mantengono i valori che avevano prima che la funzione venisse eseguita.

Per riassumere, SetSignal consente di modificare lo stato di alcuni particolari bit nel parametro tc_SigRecvd preservando lo stato degli altri. Il risultato è una nuova configurazione di bit nel parametro tc_SigRecvd.

Compiute queste operazioni, la funzione verifica se nel parametro tc_SigRecvd esistono dei bit a 1, indicanti che il task ha ricevuto dei segnali. Se qualcuno di questi bit è fra quelli cha causano exception, cioè risulta impostato anche nel parametro tc_SigExcept, la funzione imposta il flag TC_EXCEPT nel parametro tc_Flags della struttura Task e compie due operazioni diverse a seconda che il task risulti in attesa o in esecuzione. Si tenga presente che il primo caso può verificarsi solo se la funzione SetSignal viene eseguita dal sistema o da una routine di interrupt del task.

1. Se il task risulta in stato di attesa, la struttura Task viene estratta dalla lista di sistema TaskWait e viene inserita nella lista TaskReady, in modo che il task torni a essere eseguibile. Successivamente, se il task risulta essere il primo della lista viene eseguita la funzione Reschedule per causare uno scambio di controllo fra i task e cedergli così il controllo della CPU: il task torna attivo, ma avendo il flag TC_EXCEPT impostato entra in exception, cioè viene eseguita la routine di exception il cui

indirizzo è contenuto nel parametro tc_ExceptCode della struttura Task. Se invece il task non è il primo della lista, la funzione restituisce il controllo al codice chiamante. Il task entrerà in exception solo quando verrà il suo turno di CPU.

2. Se invece il task risulta eseguibile, SetSignal non fa altro che chiamare la funzione Reschedule: il task torna attivo, ma avendo il flag TC_EXCEPT impostato entra anche in questo caso in exception.

Può invece capitare che nel parametro tc_SigRecvd alcuni bit siano impostati, ma che nessuno sia un bit di exception. Allora la funzione va a verificare se il task si trova in attesa almeno di uno di questi bit. Se non lo è la funzione restituisce il controllo al codice chiamante, altrimenti la funzione rimuove la struttura Task dalla lista di sistema TaskWait, la inserisce nella lista TaskReady, e se il task è il primo della lista chiama la funzione Reschedule per cedergli il controllo; se non è il primo della lista riceverà il controllo al suo prossimo turno di CPU. Diversamente da quanto accade quando i segnali causano exception, in questo caso il task riceve il controllo esattamente nel punto in cui era entrato in attesa.

Infine, la funzione restituisce sempre in oldSignals la configurazione dei bit presente nel parametro tc_SigRecvd prima che venisse modificato. A questo proposito, se il task chiama SetSignal indicando entrambi gli argomenti a zero, il contenuto del parametro tc_SetRecvd non viene alterato, ma semplicemente restituito al codice chiamante.

Argomenti della funzione

newSignals Argomento da 32 bit nel quale gli stati dei bit interessati all'operazione (si veda il prossimo argomento) vengono copiati nei corrispondenti bit del parametro tc_SigRecvd. Gli stati relativi ai bit che non sono interessati all'operazione vengono ignorati.

signalMask Maschera da 32 bit nella quale i bit a 1 sono quelli che nel parametro tc_SigRecvd devono assumere gli stati assunti dai corrispondenti bit dell'argomento newSignals. I bit che in questa maschera risultano azzerati corrispondono ai bit del parametro tc_SigRecvd che non sono interessati all'operazione, e che pertanto mantengono i loro valori precedenti.

Discussione

Vediamo un esempio che chiarisca come intervengono i due argomenti della funzione e l'originale valore contenuto nel parametro tc_SigRecvd nella creazione della nuova configurazione dei bit di exception. Supponendo di prendere in considerazione soltanto gli otto bit meno significativi, se il parametro tc_SigRecvd contiene il valore %10100100 e il nostro task desidera modificare i bit 0, 1, 4, 5 di modo che i primi tre siano a 1 e il bit 5 sia a zero, occorre chiamare la funzione SetSignal indicando:

```
newSignals = %11011011
signalMask = %00110011
```

In questo modo si ottengono i seguenti risultati:

```
oldSignals =    %10100100
tc_SigRecvd = %10010111
```

Si noti che nell'argomento newSignals sono stati volutamente impostati a 1 anche alcuni bit non coinvolti nell'operazione, cioè non impostati nel parametro signalMask, per mostrare la loro ininfluenza.

Ogni task possiede due specie di segnali: segnali di comunicazione fra task e segnali di exception. Nell'insieme costituiscono un totale di 32 bit di segnale disponibili per ciascun task. I 16 bit inferiori (0-15) sono riservati al sistema; i 16 bit superiori (16-32) possono essere usati dal task. Alcuni sotto-insiemi di questi 32 segnali possono causare exception. I segnali che possono produrre exception sono quelli impostati nel parametro tc_SigExcept della struttura Task.

Talvolta si presenta la necessità di porre un task in uno stato noto, usando le funzioni SetSignal e SetExcept per specificare direttamente i segnali ai quali il task risulterà sensibile. Ciò si ottiene nel modo seguente:

```
oldSignals = SetExcept (Ø, ØxFFFFFFFF);
/* imposta a zero l'intera maschera di exception */
oldSignals = SetSignal (Ø, ØxFFFFFFFF);
/* imposta a zero l'intera maschera di segnali */
```

La prima istruzione imposta a zero il parametro tc_SigExcept della struttura Task per escludere l'elaborazione di qualsiasi exception; nessun segnale ricevuto dal task potrà così causare exception. La seconda istruzione imposta a zero il parametro tc_SigRecvd della struttura Task. In questo modo tutti i segnali che un task può aver ricevuto vengono eliminati, dal momento che ne vengono azzerati i bit di segnale. La situazione può in seguito essere riportata alla condizione primitiva eseguendo nel task le appropriate chiamate a SetSignal e a SetExcept. Per quanto riguarda tc_SigRecvd, si tenga conto che qualsiasi message port o task può inoltrare un nuovo segnale al task in questione, e quindi impostare un bit di questo parametro.

Si ricordi che il parametro tc_SigRecvd tiene conto di tutti i segnali ricevuti dal task, ma di cui il task non è ancora entrato in attesa tramite la funzione Wait. Ogni volta che un messaggio giunge a una message port di questo task alla quale è stato assegnato un segnale, viene impostato il corrispondente bit di segnale nel parametro tc_SigRecvd. Se il task ne è in attesa il bit viene azzerato, e il task riottiene il controllo.

La funzione SetSignal fornisce un modo per controllare dinamicamente il modo in cui un task risponde al suo ambiente esterno; esso riconosce o ignora gli eventi esterni a seconda di quanto è richiesto dalla funzione SetSignal.

Tramite SetSignal si può fare in modo che il task si comporti in molti modi diversi. Per esempio, si può simulare l'arrivo di un segnale inviato dalla tastiera.

La funzione SetSignal, comunque, può essere pericolosa perché altera il normale comportamento del task. In genere i segnali vengono assegnati al task per una sola ragione: modificare il suo comportamento non appena si verifica un certo evento. La funzione SetSignal consente di modificare arbitrariamente tale comportamento, ma in molti casi non risulta una buona politica.

Il codice per rilevare lo stato dei i segnali è:

```
oldSignals = SetSignal (Ø,Ø);
```

In questo modo la variabile oldSignals viene a contenere una copia del valore contenuto nel parametro tc_SigRecvd della struttura Task.

Se invece desiderassimo azzerare il bit di segnale relativo alla pressione della combinazione di tasti Ctrl-C, dovremmo far eseguire al nostro task l'istruzione:

```
oldSignals = SetSignal (Ø, SIGBREAKF_CTRL_C);
```

Potrebbe poi nascere l'esigenza, a un certo punto nell'esecuzione del programma, di verificare se è stata premuta la combinazione di tasti Ctrl-C. L'istruzione da usare è allora:

```
if (SetSignal(Ø,Ø) & SIGBREAKF_CTRL_C)
    puts("Combinazione Ctrl-C premuta");
```

SetSR

Sintassi di chiamata della funzione

```
oldSR = SetSR (newSR, mask)
DØ             DØ     D1
```

Scopo della funzione

Questa funzione fornisce il mezzo per modificare, in modo controllato e dall'interno del task, lo stato di alcuni particolari flag nel registro di stato della CPU. SetSR agisce soltanto sui bit del registro di stato che sono a 1 nell'argomento mask (la maschera dei bit interessati alla funzione) e restituisce il contenuto dell'intero registro di stato della CPU.

Argomenti della funzione

newSR Nuovi valori dei flag del registro di stato (SR) della CPU.

mask Maschera dei flag che devono assumere gli stati indicati nell'argomento newSR.

Discussione

I flag del registro di stato della CPU 68000 sono i seguenti:

0 Carry
1 Overflow
2 Zero
3 Negative
4 Extend
5 Non usato
6 Non usato
7 Non usato
8 Primo bit di maschera interrupt
9 Secondo bit di maschera interrupt
10 Terzo bit di maschera interrupt
11 Non usato
12 Non usato
13 Supervisor state
14 Non usato
15 Modo Trace

Il registro di stato a 16 bit del 68000 è diviso in due metà: il byte utente (i bit da 0 a 7) e il byte di sistema (i bit da 8 a 15).

Si noti che ci sono tre bit del registro di stato che riportano il livello di priorità di interrupt in cui la CPU sta funzionando (i livelli di interrupt della CPU sono sette, e sono quindi esprimibili su tre bit). Se si verifica un interrupt con un livello di priorità superiore al valore conservato in questa maschera di tre

bit, la CPU sospende quanto sta facendo e inizia l'elaborazione dell'interrupt a più alta priorità.

Se invece si verifica un interrupt con una priorità inferiore al valore indicato dalla maschera, il nuovo interrupt dovrà attendere fino a quando la sequenza di elaborazione non termina. Si vedano le spiegazioni delle funzioni AddIntServer, Cause, e RemIntServer per una trattazione delle relazioni tra i sette livelli di interrupt del 68000 e i 15 interrupt effettivamente usati dall'Amiga.

Per leggere il contenuto del registro di stato si chiama:

```
current = SetSR (Ø, Ø)
```

Per impostare a 3 il livello di interrupt del processore si usa invece:

```
oldSR = SetSR ($Ø3ØØ, $Ø7ØØ)
```

Ciò consente di selezionare i bit 8, 9 e 10 (grazie a $0700) e impostare i bit 8 e 9 a 1, mentre il bit 10 viene azzerato. La variabile oldSR ora contiene il valore precedente del registro di stato. Per reimpostare gli interrupt del processore al livello primitivo si usa l'istruzione:

```
oldSR = SetSR (oldSR, $Ø7ØØ)
```

Ciò consente di selezionare i bit 8, 9 e 10 (grazie a $0700) e impostare i bit 8, 9 e 10 ai loro vecchi valori.

SetTaskPri

Sintassi di chiamata della funzione

```
oldPriority = SetTaskPri (taskCS, newPriority)
DØ: Ø-8                   A1      DØ: Ø-8
```

Scopo della funzione

Questa funzione cambia la priorità di un task senza riguardo per lo stato in cui si trova (in attesa, eseguibile, in esecuzione...), e restituisce il valore della vecchia priorità. Dal momento che comporta una chiamata alla funzione Reschedule, l'esecuzione della funzione SetTaskPri può anche causare uno scambio di controllo fra i task.

Argomenti della funzione

taskCS Indirizzo della struttura Task relativa al task di cui si desidera modificare il livello di priorità.

newPriority La nuova priorità che la funzione assegna al task.

Discussione

Ci sono quattro routine dell'Exec che riguardano specificamente le gestione dei task nel sistema Amiga: AddTask, FindTask, RemTask e SetTaskPri. Si vedano anche le spiegazioni relative a queste funzioni.

L'Amiga è una macchina multitasking, in grado di gestire l'esecuzione parallela di più task indipendenti. Il sistema è predisposto per celare a ogni task in esecuzione la presenza degli altri. In altre parole, nessun task è tenuto a sapere dell'esistenza degli altri: è il sistema che si preoccupa di coordinarli e ripartire fra loro il tempo della CPU.

Nei sistemi multitasking il coordinamento fra i task viene gestito dallo scheduler, un sofisticato algoritmo che sovraintende a tutti i task presenti nel sistema. Nell'Amiga questo algoritmo è di tipo *preemptive* quando considera task dotati di priorità differenti, cioè sfrutta il diritto di prelazione che un task a più alta priorità fa valere su un task a priorità inferiore. Quando ci sono invece diversi task tutti con la stessa priorità, l'algoritmo diventa di tipo *round-robin*, cioè a ripartizione circolare del tempo di CPU.

In pratica, lo scambio di controllo fra i task avviene sulla base delle loro priorità. Quando si verifica uno scambio, lo scheduler seleziona il task eseguibile a più alta priorità e lo riattiva nel punto in cui era stato sospeso. Questo task detiene il controllo della CPU fino a quando non si verifica una delle seguenti condizioni: un task a priorità superiore diventa eseguibile; il task esaurisce completamente la partizione di tempo assegnatagli e c'è un task della stessa priorità pronto per essere eseguito; il task entra volontariamente in attesa che si verifichi un evento esterno per continuare. In tutti questi casi perde il controllo, cioè viene sospeso, ma nei primi due continua a essere classificato come task eseguibile, cioè pronto a riottenere il controllo in ogni istante, mentre nella terza eventualità diventa un task in attesa, e quindi non eseguibile.

Quindi in ogni istante la situazione che si viene a creare è caratterizzata da un task *in esecuzione*, da una serie di task *eseguibili* e quindi elencati nella lista di sistema TaskReady, e da una serie di task *in attesa* e quindi elencati nella lista di sistema TaskWait. Quando si verifica uno degli eventi di cui un task è in attesa, il task viene estratto dalla lista TaskWait (nella quale i nodi non sono disposti in ordine di priorità) e inserito nella lista TaskReady (nella quale invece i nodi sono disposti in ordine di priorità, con il nodo a priorità maggiore in testa alla lista), tornando a essere eseguibile.

Quello descritto è un comportamento di tipo "preemptive", ma che diventa

di tipo round-robin quando esistono task eseguibili che hanno la stessa priorità del task in esecuzione.

Da questa descrizione risulta evidente l'importanza della priorità nella gestione dei task. Se per esempio un task chiamasse per se stesso la funzione SetTaskPri per autoassegnarsi una priorità molto elevata, e successivamente non entrasse mai in attesa, non permetterebbe l'esecuzione di nessun altro task.

Si possono variare le priorità di tutti i task, fuorché quelle dei task di sistema, che vengono impostate dalle routine dell'Exec Schedule e Reschedule e sono comprese fra -20 e +20.

In genere i task hanno tutti priorità zero, cioè hanno il parametro ln_Pri (della sotto-struttura Node della struttura Task) azzerato. Tuttavia la priorità può essere impostata a valori compresi fra +127 e -128.

Task a uguale priorità ricevono eguali partizioni di tempo della CPU. Il tempo di suddivisione è attualmente impostato a 64 millisecondi. Per dare un'idea dell'entità di questo intervallo di tempo, si noti che è grossomodo equivalente al tempo necessario per riempire quattro quadri video.

Quando un task viene sostituito, il sistema Exec salva automaticamente tutti i registri nello stack del task. La sola eccezione è il valore contenuto nel registro USP (User Stack Pointer), che viene invece salvato nel parametro tc_SPReg della struttura Task.

La maggior parte delle volte, lo scambio di controllo si verifica per effetto di un interrupt. Per esempio, supponiamo che il task, interagendo con il dispositivo TrackDisk, richieda di leggere una certa quantità di dati presenti sul disco. Per farlo invia una richiesta di I/O e quindi entra in attesa della risposta. A un certo punto, il dispositivo segnala al task che i dati che lo riguardano sono pronti, un'operazione che svolge all'interno di una routine di interrupt. Questo comporta il passaggio del task dallo stato di attesa allo stato di eseguibile, cioè viene estratto dalla lista di sistema TaskWait e inserito, secondo l'ordine di priorità, nella lista di sistema TaskReady. Quando l'elaborazione dell'interrupt termina, lo scheduler riesamina la situazione, e cede il controllo al task che occupa la prima posizione nella lista TaskReady, che può anche essere il task appena ridiventato eseguibile.

Da questa descrizione risulta evidente che nella programmazione dell'Amiga i loop vanno evitati: quando un task non può proseguire se non si verifica un particolare evento, deve entrare in attesa escludendosi dall'uso della CPU fino a quando quell'evento non si verifica.

Signal

Sintassi di chiamata della funzione

```
oldSignals = Signal (taskCS, signals)
               A1      DØ
```

Scopo della funzione

Questa funzione invia a un task la maschera di segnali indicata nell'argomento signals. È uno strumento per l'invio di segnali che si affianca a quello più automatico delle message port. Se il task è in attesa di uno dei segnali inviati con questa funzione, torna eseguibile e riottiene il controllo alla prima occasione. Per attivare il task, il sistema usa la routine privata di sistema Reschedule (sempre che le altre condizioni siano soddisfatte).

Una descrizione più dettagliata del comportamento di questa funzione ne aiuta la comprensione. Signal effettua prima di tutto un'operazione logica OR fra il contenuto dell'argomento signals e quello del parametro tc_SigRecvd della struttura Task indicata come primo argomento, e memorizza quindi il risultato nel parametro tc_SigRecvd.

Compiuta quest'operazione, la funzione verifica se nel parametro tc_SigRecvd esistono dei bit a 1, i quali indicherebbero l'arrivo di segnali. Se almeno uno di questi bit è fra quelli cha causano exception, cioè risulta impostato anche nel parametro tc_SigExcept, la funzione imposta il flag TC_EXCEPT nel parametro tc_Flags della struttura Task e compie due operazioni diverse a seconda che il task risulti in attesa o in esecuzione, tenendo presente che il primo caso può verificarsi soltanto se nell'argomento taskCS della funzione il task indica l'indirizzo della propria struttura Task.

1. Se il task risulta in stato di attesa, la struttura Task viene estratta dalla lista di sistema TaskWait e viene inserita nella lista TaskReady, in modo che il task rientri fra quelli eseguibili. Subito dopo viene eseguita la funzione Reschedule, e se il task è al primo posto nella lista riceve il controllo della CPU: il task torna attivo, ma avendo il flag TC_EXCEPT impostato entra in exception, cioè viene eseguita la routine di exception il cui indirizzo è contenuto nel parametro tc_ExceptCode della struttura Task. Se invece il task non risulta il primo della lista, la funzione restituisce il controllo al codice chiamante e il task entrerà in exception solo quando verrà il suo turno di CPU.

2. Se invece il task risulta eseguibile, Signal non fa altro che chiamare la funzione Reschedule: il task torna attivo, ma avendo il flag TC_EXCEPT impostato entra anche in questo caso in exception.

Può anche capitare che nel parametro tc_SigRecvd alcuni bit siano impostati, ma che nessuno sia un bit di exception. Allora la funzione va a verificare se il task si trova in attesa almeno di uno di questi bit. Se non lo è, la funzione restituisce il controllo al codice chiamante; in caso affermativo, invece, la funzione rimuove la struttura Task dalla lista di sistema TaskWait, la inserisce nella lista TaskReady, e se il task è il primo della lista chiama la funzione Reschedule per cedergli il controllo; se non è il primo della lista riceverà il controllo al suo prossimo turno di CPU. Al contrario di quanto accade quando i segnali ricevuti causano exception, questa volta l'esecuzione del task riprende esattamente dal punto in cui il task era entrato in attesa del segnale.

Infine, la funzione restituisce sempre in oldSignals la configurazione di bit presente nel parametro tc_SigRecvd prima delle modifiche. Si osservi che se il task chiama Signal indicando il valore zero come secondo argomento, il contenuto del parametro tc_SigRecvd non viene alterato, ma viene semplicemente restituito al codice chiamante.

Si può inviare un segnale a un task qualunque sia il suo stato (in esecuzione, eseguibile o in attesa) e, come appare evidente dalla precedente spiegazione, senza badare al fatto che causi exception.

Argomenti della funzione

taskCS — Indirizzo della struttura Task del task che deve ricevere i segnali.

signals — Valore da 32 bit nel quale i bit a 1 sono quelli relativi ai segnali che devono essere inviati al task indicato nel primo argomento.

Discussione

Ci sono sei funzioni dell'Exec che riguardano i segnali: AllocSignal, SetExcept, FreeSignal, SetSignal, Signal e Wait. In un certo senso, anche le funzioni GetMsg, PutMsg e ReplyMsg possono essere incluse nella categoria delle funzioni di gestione dei segnali.

La funzione Signal viene considerata di basso livello. Il suo scopo principale è quello di fornire supporto a funzioni di livello più alto, come PutMsg. Generalmente un task non ha bisogno d'inviare segnali a se stesso, mentre può aver bisogno d'inoltrare segnali ad altri task per riattivarli. Comunque, a parte le interazioni task-task, sono il sistema e i dispositivi a fare l'uso maggiore dei segnali, sia internamente sia nelle interazioni con i task,

attraverso il meccanismo delle message port.

I segnali vengono per lo più impiegati nella gestione delle message port e dei messaggi. Tuttavia, sia la funzione Signal sia la funzione SetSignal possono fornire un utile meccanismo per eseguire il debug dei task e dei programmi.

Nella trattazione dei segnali è opportuno ricordare che il sistema non prevede l'accodamento di segnali dello stesso tipo (cioè relativi allo stesso bit di segnale) inviati a un task. Se il task non ne è in attesa, dopo il primo segnale ricevuto gli altri dello stesso tipo vengono persi. Se in seguito il task entra in attesa di quel particolare segnale, riottiene subito il controllo, ma ai suoi occhi è stato inviato un solo segnale. È quindi opportuno che segnali dello stesso tipo vengano causati da eventi analoghi, e magari dalla stessa sorgente. Un esempio è il meccanismo di gestione dei messaggi. Quando a una message port giungono in rapida successione numerosi messaggi, il task riceve altrettanti segnali; se durante l'arrivo di questi messaggi il task non è in attesa di segnali da quella message port, quando poi chiama la funzione Wait riottiene subito il controllo ma ai suoi occhi risulta l'arrivo di un solo segnale: se su questa base prelevasse solo un messaggio dalla message port e richiamasse la funzione Wait, tutti gli altri messaggi pervenuti rimarrebbero inutilmente in coda. Proprio per evitare questa situazione si consiglia sempre di creare i task in modo che quando riottengono il controllo grazie a un segnale entrino in un loop che svuoti completamente la coda della message port.

SumLibrary

Sintassi di chiamata della funzione

SumLibrary (library)
A1

Scopo della funzione

Se la libreria indicata ha il flag LIBF_CHANGED impostato, nel parametro lib_Flags della struttura Library, SumLibrary calcola un nuovo checksum di libreria e lo memorizza nel parametro lib_Sum della struttura Library. Il checksum è la somma di tutte le word di cui è composta la tavola dei vettori di salto della libreria.

Se invece la libreria indicata ha il flag LIBF_CHANGED azzerato, la funzione calcola ugualmente il checksum di libreria ma poi lo confronta con il vecchio valore presente nel parametro lib_Sum della struttura Library. Se sono uguali la funzione restituisce semplicemente il controllo, mentre se sono diversi

chiama la funzione Alert per far entrare in condizione di alert il sistema; nel compiere questa operazione, indica come codice d'errore AN_LibChkSum (0x81000003) che appare sullo schermo come avvertimento.

Occorre infine notare che SumLibrary non esegue nulla e restituisce semplicemente il controllo quando rileva che il flag LIBF_SUMUSED del parametro lib_Flags è azzerato. Questo accade quando il creatore della libreria decide che per la sua libreria non dev'essere calcolato il checksum.

Argomenti della funzione

library Indirizzo della struttura Library relativa alla libreria sulla quale dev'essere calcolato il checksum.

Discussione

Ci sono otto funzioni dell'Exec che riguardano le librerie: AddLibrary, MakeFunctions, MakeLibrary, OpenLibrary, RemLibrary, SetFunction e SumLibrary. Si vedano anche le spiegazioni relative a queste funzioni.

Dato che il sistema Exec è strettamente correlato alle librerie, esso tenta di mantenere coerenti le informazioni contenute in ogni libreria. Lo strumento per farlo è la possibilità di controllo tramite il checksum. Attualmente non esiste un task di sistema che periodicamente effettui il checksum su tutte le librerie che hanno il flag LIBF_SUMUSED impostato, ma in future versioni del sistema operativo potrebbe essere previsto, così da mantenere un controllo costante sull'integrità delle tavole dei vettori di salto delle librerie.

Se si fa uso della funzione MakeLibrary oppure della routine SetFunction è perché s'intende cambiare il contenuto di una libreria, e ogni volta che si altera una libreria si dovrebbe calcolare e memorizzare per essa un nuovo valore di checksum. SetFunction chiama SumLibrary automaticamente, ma MakeLibrary no, o almeno non direttamente.

Va ricordato infine che la funzione SumLibrary viene anche chiamata automaticamente dalla funzione AddLibrary, la quale, lo ricordiamo, serve per rendere disponibile una libreria al sistema.

SuperState

Sintassi di chiamata della funzione

```
oldSysStack = SuperState ()
DØ
```

Scopo della funzione

Questa funzione provoca l'attivazione del modo supervisor della CPU, permettendo al task di mandare in esecuzione istruzioni privilegiate eseguibili soltanto in quel modo. Si noti che, pur attivando il modo supervisor della CPU, la funzione forza il sistema a continuare a usare come stack quello del task (lo user stack) anziché il supervisor stack. Per questa ragione il task, dopo aver chiamato la funzione SuperState, ha ancora accesso a tutti i valori memorizzati nello user stack, come per esempio gli indirizzi di ritorno delle chiamate alle subroutine.

La chiamata a SuperState comporta però che anche il sistema degli interrupt (che funzionando in modo supervisor generalmente si avvale del supervisor stack) utilizzi lo user stack. Questo significa che se un task chiama la funzione SuperState dev'essere certo di disporre di uno stack abbastanza capiente per essere utilizzato anche dal sistema degli interrupt.

La funzione restituisce un puntatore al supervisor stack, che il task dovrà in seguito utilizzare per la chiamata alla funzione UserState, necessaria per riattivare il modo user della CPU.

SuperState non ha alcun effetto se viene chiamata quando il modo supervisor è già attivo, e restituisce un valore nullo.

Argomenti della funzione

Questa funzione non prevede argomenti.

Discussione

Ci sono due funzioni dell'Exec che consentono di operare lo scambio tra i modi di funzionamento della CPU: le funzioni SuperState e UserState.

I microprocessori Motorola 68000/10/20/30 hanno due possibili modi di funzionamento: il modo user (utente) e il modo supervisor (supervisore). Il flag

S del registro di stato (SR) determina in quale di questi modi la CPU deve funzionare; se il bit S è impostato a 1 il 68000 si trova in modo supervisor, quello che permette di eseguire tutte le istruzioni previste dalla CPU. Nel modo user, invece, la CPU si rifiuta di eseguire alcune istruzioni, che vengono quindi definite "privilegiate", cioè eseguibili soltanto in modo supervisor. Con il 68000 le istruzioni privilegiate sono:

ANDI to SR	*And* immediato con il registro di stato.
EORI to SR	*Or* esclusivo immediato con il registro di stato.
ORI to SR	*Or* immediato con il registro di stato.
MOVE to SR	Muove nel registro di stato.
MOVE USP	Muove nel registro puntatore allo user stack.
RESET	Esegue il reset dei dispositivi esterni.
RTE	Ritorno da exception.
STOP	Carica il valore dell'operando nel registro di stato e si ferma.

Mentre il 68000 si trova nel modo supervisor, in pratica tutte le istruzioni che usano implicitamente o esplicitamente lo stack accedono al supervisor stack.

Per la maggior parte del tempo il sistema opera nel modo user, in particolar modo durante l'esecuzione dei task del programmatore. Tutte le elaborazioni delle exception sono svolte nel modo user senza riguardo allo stato del bit S quando l'exception si verifica. Quando un segnale causa un'exception, il sistema prima di cedere il controllo alla routine di exception del task predispone due registri della CPU nel seguente modo:

A0	Contiene l'indirizzo dell'area dati della routine di exception. Questa è l'area di memoria che viene usata dalla routine di exception per memorizzare i propri dati.
D0	Contiene una maschera di bit di segnale, così che la routine di exception possa rilevare la causa dell'exception.

Sempre prima di eseguire il codice di exception, il sistema salva lo stato del task nello user stack. Lo schema con cui inserisce questi dati sullo stack (cioè lo stack frame), partendo dall'indirizzo inferiore e salendo verso l'alto, è il seguente:

Schema dello stack del task

Program counter (PC)
Registro di stato (16 bit, SR)
Da D0 a D7
Da A0 a A7
(parte rimanente dello stack)

In uscita dall'elaborazione dell'exception si deve impostare D0 al nuovo valore della maschera di abilitazione per le exception; in questo nuovo valore i bit di segnale impostati sono quelli che causeranno la successiva exception. Si noti che questa nuova maschera può essere la stessa che la routine di exception ha ricevuto in D0. Infine, la routine di exception deve eseguire l'istruzione RTS per completare l'elaborazione.

I task possono richiedere che determinati eventi asincroni causino exception. In pratica, mentre perché un normale segnale venga rilevato ed elaborato dal task è necessario che il task ne sia in attesa, gli eventi che causano exception provocano automaticamente e in ogni condizione il congelamento del task e l'esecuzione della relativa routine di exception. In questo senso, le routine di exception sono molto simili alle routine di interrupt, con la differenza che sono dedicate a un particolare task. Invece di entrare in attesa di un segnale, il task lo definisce come segnale che provoca exception tramite la routine SetExcept: in questo modo, quando quel segnale sopraggiunge, il task viene interrotto per mandare in esecuzione la sua routine di exception, la quale deve rilevare l'origine dell'exception e comportarsi di conseguenza.

Nel sistema Amiga oltre alle exception appena descritte (che abbiamo definito asincrone) esistono anche le trap, cioè exception sincrone provocate esclusivamente da particolari operazioni compiute dai codici del task (errore d'indirizzamento, istruzione illegale, divisione per zero, violazione di privilegio, esecuzione dell'istruzione TRAP # della CPU...).

Le trap, a differenza delle exception, vengono elaborate nel modo supervisor. Questo significa che, quando si verifica una trap, lo scambio di controllo fra i task viene sospeso, cosa che non accade con l'elaborazione delle exception. Quando la CPU rileva una trap e ne avvia l'elaborazione, accede alla tavola delle exception e cede il controllo all'indirizzo contenuto nell'autovettore relativo a quella trap. Nell'Amiga, questi indirizzi individuano particolari routine dell'Exec, le quali prendono il controllo, determinano qual è il task in attività e decidono se la trap può essere elaborata da quel task. In caso affermativo, il controllo viene ceduto alla routine di gestione delle trap prevista dal task. Si ricordi che l'indirizzo di questa routine dev'essere memorizzato nel parametro tc_TrapCode della struttura Task.

Una volta che il 68000 è entrato nel modo supervisor, il supervisor stack viene a contenere la struttura di dati della trap avvenuta e il suo numero. Si consulti il manuale di riferimento del 68000 per una spiegazione degli stack frame relativi alle trap e dei numeri di trap.

Per ritornare da una trap si deve rimuovere il numero di trap dal supervisor stack ed eseguire l'istruzione RTE.

Si ricordi che le routine di trap dovrebbero essere realizzate in modo da risultare molto corte, dal momento che durante la loro esecuzione lo scambio di controllo fra i task è sospeso.

TypeOfMem

Sintassi di chiamata della funzione

```
attributi = TypeOfMem (indirizzo)
DØ                    A1
```

Scopo della funzione

Questa funzione restituisce un valore descrittivo del tipo di memoria presente all'indirizzo indicato. Il valore restituito è una long word in cui sono posti a 1 i flag appropriati (per esempio MEMF_CHIP o MEMF_FAST) oppure vale zero se l'indirizzo non fa parte della memoria RAM.

Argomenti della funzione

indirizzo	L'indirizzo del quale si desiderano conoscere gli attributi.

Discussione

Nella libreria Exec esistono nove funzioni di gestione della memoria: AllocAbs, Allocate, AllocEntry, AllocMem, AvailMem, Deallocate, FreeEntry, FreeMem e TypeOfMem. Si vedano anche le spiegazioni relative a queste funzioni.

TypeOfMem viene generalmente usata per sapere se un dato indirizzo si trova nella chip RAM. Nei modelli attuali la chip RAM rappresenta il primo megabyte di memoria, a differenza dei modelli meno recenti in cui costituiva i primi 512K. Versioni future dell'Amiga porteranno probabilmente questa quantità a due megabyte.

Se l'indirizzo indicato non fa parte della lista di sistema (per esempio si trova in ROM o in aree non utilizzate), la funzione restituisce il valore zero.

UserState

Sintassi di chiamata della funzione

UserState (sysStack)
DØ

Scopo della funzione

Questa funzione forza la CPU 68000 a riattivare il modo user di funzionamento, e dev'essere eseguita soltanto quando la CPU sta funzionando nel modo supervisor. In genere viene utilizzata dopo una chiamata a SuperState, che invece attiva il modo supervisor mantenendo come stack quello utente.

Argomenti della funzione

sysStack — Valore del puntatore al supervisor stack (SSP). Questo valore è generalmente quello restituito dalla funzione SuperState, e serve per ripristinare il contenuto del registro SSP, precedentemente modificato da SuperState.

Discussione

Ci sono due funzioni dell'Exec che consentono di operare lo scambio tra i due modi di funzionamento del microprocessore 68000: SuperState e UserState. Si consulti anche la spiegazione della funzione SuperState.

Il modo user del 68000 è a un livello di privilegio inferiore rispetto a quello del modo supervisor. Se il bit S del registro di stato è impostato a 0, significa che la CPU sta operando in modo user. La maggior parte delle istruzioni svolgono il loro lavoro in modo identico nelle due modalità. Tuttavia, alcune istruzioni che hanno importanti effetti sul sistema sono "privilegiate": possono essere usate solamente nel modo supervisor. In particolare i programmi non hanno la possibilità di usare le istruzioni STOP o RESET nel modo user.

Il 68000 è progettato per assicurare che un programma non possa entrare nel modo supervisor accidentalmente. Questo si ottiene rendendo privilegiate

tutte le istruzioni che possono modificare il registro di stato. Sono istruzioni privilegiate anche MOVE to USP (muove un indirizzo nel puntatore allo user stack) e MOVE from USP (preleva l'indirizzo contenuto nel puntatore allo stack utente). Quando il 68000 si trova nel modo user e sta elaborando istruzioni di programma, soltanto le trap hanno la facoltà di cambiare il livello di privilegio.

Il ritorno dal modo supervisor al modo user si può ottenere utilizzando una delle quattro istruzioni che seguono:

- RTE (ritorno da exception).
- MOVE word to SR (muove un nuovo valore nel registro di stato).
- ANDI to SR (*And* immediato di un valore con il registro di stato).
- EORI to SR (*Or* esclusivo immediato di un valore con il registro di stato).

Si veda anche il manuale di riferimento del Motorola 68000.

Vacate

Sintassi di chiamata della funzione

Vacate (semaphore)
AØ

Scopo della funzione

Questa funzione serve per rilasciare una struttura Semaphore precedentemente bloccata con una chiamata alla funzione Procure. Ricordiamo che la struttura Semaphore serve per definire un semaforo basato su messaggi. La struttura Semaphore è in pratica una message port alla quale la funzione Procure accoda i messaggi indicati dai task quando tentano di ottenere il semaforo e questo non è disponibile. Vacate non fa altro che prelevare il primo messaggio presente nella coda alla struttura Semaphore e restituirlo al mittente. Il task che l'aveva indicato nella chiamata della funzione Procure lo riceve nella sua reply port, e può quindi assumere che il semaforo sia diventato di sua proprietà.

Argomenti della funzione

semaphore Indirizzo della struttura Semaphore che definisce il semaforo basato su messaggi. Al suo interno, la sotto-struttura MsgPort consente di accodare i messaggi dei task che desiderano appropriarsi del semaforo.

Discussione

L'Exec contiene due funzioni che riguardano i semafori basati su messaggi: Procure e Vacate. Quando un task chiama la funzione Procure e ottiene il semaforo (eventualmente dopo un periodo d'attesa), deve poi chiamare la funzione Vacate per rilasciare quel semaforo, in modo che ridiventi disponibile per il task successivo. In questo senso, Procure e Vacate sono molto simili alle funzioni ObtainSemaphore e ReleaseSemaphore, che riguardano i semafori di segnalazione.

Wait

Sintassi di chiamata della funzione

```
signals = Wait (signalSet)
D0              D0
```

Scopo della funzione

Questa funzione permette a un task di entrare in attesa di uno o più segnali. La funzione fa entrare in attesa il task, e gli restituisce il controllo soltanto quando nel sistema una delle sue message port, un altro task o una routine di interrupt inoltrano al task sospeso uno dei segnali di cui è in attesa. Quando la funzione Wait rileva l'arrivo di uno dei segnali attesi dal task, azzera il relativo bit di segnale nel parametro tc_SigRecvd della struttura Task e restituisce il controllo. Il task, analizzando il valore restituito dalla funzione, può rilevare l'origine del segnale e comportarsi di conseguenza.

Può capitare che un task riceva uno o più segnali prima di mettersi in attesa. In questo caso il task riottiene il controllo non appena chiama la funzione Wait. Si ricordi che potrebbe essere giunto anche più di un segnale, quindi il

task deve aspettarsi che ci possa essere anche più di un messaggio da elaborare.

Questa funzione non può essere chiamata quando la CPU sta funzionando in modo supervisor. Il motivo è che i segnali possono causare lo scambio di controllo fra i task, e quando la CPU funziona in modo supervisor lo scambio di controllo fra i task è sospeso.

Argomenti della funzione

signalSet In questo argomento da 32 bit, il task deve impostare a 1 i bit corrispondenti ai segnali per i quali desidera entrare in attesa.

Discussione

Ci sono sei funzioni dell'Exec che riguardano i segnali: AllocSignal, FreeSignal, SetExcept, SetSignal, Signal e Wait. Anche le funzioni GetMsg, PutMsg e ReplyMsg possono essere incluse nella categoria delle funzioni di gestione dei segnali. I task comunicano tra loro mediante i messaggi e le message port. Se il task assegna un bit di segnale a una message port, quando nella coda a quella message port giunge un messaggio il task riceve il segnale. Se era entrato in attesa tramite la funzione Wait, riottiene il controllo. Ma le origini dei segnali non sono soltanto le message port. I task possono anche inviarsi segnali vicendevolmente, e anche le routine di interrupt possono inviare segnali.

Ci sono alcune cose da sottolineare riguardo alla trasmissione dei segnali. Ogni task dispone di un insieme di 32 bit di segnale; quelli da 0 a 15 sono usati dalle routine di sistema, e quindi il task li deve considerare già allocati e assegnati. Gli altri 16 (da 16 a 31) sono a completa disposizione del task, che può allocarli ed eventualmente assegnarli a message port, task, routine di interrupt...

I segnali vengono molto spesso usati per gestire in maniera efficiente lo scambio dei messaggi. Quando un task deve entrare in attesa di messaggi da un certo numero di message port in suo possesso, utilizza la funzione Wait per entrare in attesa di questi segnali senza impegnare inutilmente la CPU.

La funzione Wait, insieme con le funzioni SetSignal e Signal, serve anche per effettuare il debug dei programmi. Si può utilizzare Signal quando si desidera inoltrare un segnale al task e SetSignal quando da una routine di interrupt, di trap o di exception del task si devono inoltrare al task diversi segnali simultaneamente. In pratica, all'interno del nostro programma si può far precedere ogni istruzione Wait da una chiamata a Signal o SetSignal, in modo che quando Wait viene chiamata restituisca subito il controllo; Signal e SetSignal simulano in questo modo l'invio di segnali da parte di un altro task o dispositivo. Una volta che il task è stato completamente controllato, si

rimuovano le istruzioni Signal e SetSignal inserite nel sorgente per effettuare il debug.

WaitIO

Sintassi di chiamata della funzione

```
error = WaitIO (iORequest)
D0              A1
```

Scopo della funzione

Questa funzione attende che venga completata una particolare richiesta di I/O, cioè che venga restituita la struttura di I/O che il task ha utilizzato per formulare la richiesta al particolare dispositivo di I/O. Se la richiesta di I/O è già stata elaborata, la funzione restituisce il controllo immediatamente.

Il funzionamento di WaitIO è il seguente. Prima di tutto verifica lo stato del flag IOF_QUICK contenuto nel parametro io_Flags della struttura di I/O indicata come argomento: se risulta impostato, copia in D0 il contenuto del parametro io_Error e restituisce subito il controllo, perché la richiesta è stata evidentemente inoltrata con la funzione BeginIO e con una richiesta di QuickIO, e il QuickIO è stato accordato. Se invece il flag risulta azzerato, WaitIO entra in un ciclo all'interno del quale chiama la funzione Wait indicando il bit di segnale assegnato alla reply port del task; ogni volta che riottiene il controllo (è giunto un messaggio nella reply port) controlla se il parametro ln_Type della richiesta che sta aspettando contiene la costante NT_REPLYMSG. Se l'esito è negativo rientra nel ciclo e chiama nuovamente Wait, mentre se è affermativo significa che il messaggio giunto nella coda alla reply port del task è la richiesta che il task aveva indicato nell'argomento iORequest. In questo secondo caso, WaitIO provvede a estrarre la richiesta dalla coda alla reply port del task, memorizza nel registro D0 il contenuto del parametro io_Error e restituisce il controllo.

WaitIO dev'essere usata con cautela perché non restituisce il controllo fino a quando la richiesta di I/O non è stata soddisfatta e restituita al mittente (cioè alla reply port del task). Se esiste la possibilità che questa situazione non si verifichi, è più sicuro usare la funzione Wait, che ha più probabilità di restituire il controllo, dal momento che permette di entrare in attesa di diversi segnali.

Argomenti della funzione

iORequest Indirizzo della struttura di I/O che si è inviata a un dispositivo per formulare una richiesta.

Discussione

Ci sono otto funzioni nell'Exec direttamente correlate con i dispositivi di I/O del sistema Amiga: AddDevice, CloseDevice, OpenDevice, RemDevice, CheckIO, DoIO, SendIO e WaitIO. Si vedano anche le spiegazioni relative a queste funzioni, e il volume *Programmare l'Amiga Vol. II*.

Un task può inviare a un dispositivo dell'Amiga le proprie richieste di I/O in diversi modi. Se desidera inoltrare la specifica richiesta in modo sincrono, deve impiegare la funzione DoIO, la quale non restituisce il controllo fino a quando la richiesta non viene completamente elaborata dal dispositivo. Se invece intende compiere altre operazioni mentre la richiesta di I/O attende di essere elaborata dal dispositivo, deve impiegare la funzione SendIO, la quale restituisce subito il controllo.

DoIO, a differenza di SendIO, tenta sempre di farsi accordare il QuickIO. Oltre a queste due funzioni, il task può chiamare direttamente la funzione BeginIO del dispositivo. Questo metodo permette d'impostare a proprio piacimento il contenuto del parametro io_Flags della richiesta di I/O.

Ha senso chiamare WaitIO soltanto dopo aver chiamato SendIO o BeginIO, dal momento che DoIO chiama internamente la funzione Wait.

Per quanto riguarda i rischi che comporta l'uso della funzione WaitIO quando non è certa la restituzione della richiesta da parte del dispositivo, in *Programmare l'Amiga Vol. II*, in occasione della spiegazione della funzione WaitIO, è illustrato un metodo che permette di far riprendere l'esecuzione del task quando WaitIO mantiene il controllo troppo a lungo.

WaitPort

Sintassi di chiamata della funzione

```
message = WaitPort (msgPort)
D0                 A0
```

Scopo della funzione

Questa funzione attende che ci sia almeno un messaggio nella coda relativa alla message port indicata come argomento. Se necessario, la funzione chiama a sua volta la funzione Wait per entrare in attesa del segnale assegnato a quella message port. Se nella message port è già presente anche un solo messaggio, la funzione restituisce il controllo immediatamente. Il valore restituito dalla funzione è sempre l'indirizzo della prima struttura Message giunta nella coda. Si tenga presente che WaitPort non rimuove il messaggio in questione dalla coda.

Argomenti della funzione

msgPort	Indirizzo della struttura MsgPort che definisce la message port da controllare.

Discussione

Ci sono quattro funzioni dell'Exec direttamente correlate alla gestione delle message port nel sistema Amiga: AddPort, FindPort, RemPort e WaitPort. AddPort, FindPort e RemPort riguardano esclusivamente le message port pubbliche, cioè quelle che vengono elencate nella lista di sistema PortList e sono dotate di nome. WaitPort riguarda sia le message port pubbliche che quelle private. Strettamente correlate alle routine che riguardano le message port sono le routine PutMsg, GetMsg e ReplyMsg, le quali servono per gestire i messaggi che vengono scambiati attraverso le message port.

Si noti che la funzione WaitPort restituisce l'indirizzo della prima struttura Message che giunge alla particolare message port, senza riguardo per le caratteristiche del messaggio. Non si può chiedere a questa funzione di attendere un messaggio con determinate caratteristiche. Si noti inoltre che la funzione restituisce soltanto l'indirizzo della prima struttura Message in coda, anche se sono giunti diversi messaggi in rapida successione.

WaitPort svolge un compito analogo alla funzione GetMsg, con l'importante differenza che GetMsg funziona in modo asincrono, cioè non mette mai il task in attesa, anche quando nella message port indirizzata non esistono messaggi in coda, mentre WaitPort è una funzione sincrona.

Funzioni addizionali dell'Exec

Oltre alle funzioni già trattate in questo capitolo, ci sono 12 funzioni addizionali che in genere vengono usate dal sistema ma che possono essere utili, in determinate circostanze, anche al programmatore. Abbiamo inserito qui, per completezza, una breve descrizione di ciascuna di queste funzioni. Si noti che, contrariamente alla maggior parte delle funzioni descritte nel corso del capitolo, solo due tra le funzioni aggiuntive possiedono argomenti.

Debug(0)

Chiama il debugger di sistema, che per default è ROM-Wack. Comunque, qualsiasi task può alterare il vettore relativo alla funzione Debug della libreria Exec, in modo che venga invocato un debugger proprio anziché ROM-Wack quando un task chiama la funzione Debug. La funzione da utilizzare per effettuare questo intervento nella tavola dei vettori della libreria Exec è SetFunction.

Disable()

Permette di sospendere la gestione degli interrupt condotta dal sistema. Implicitamente, quindi, sospende anche la gestione multitasking dei task, come se fosse stata chiamata anche la funzione Forbid(). Dopo aver chiamato la funzione Disable, il task ha la certezza che il sistema non può svolgere nessun compito parallelamente all'esecuzione dei codici che seguono la chiamata a Disable. Per riattivare gli interrupt, e quindi anche lo scambio di controllo fra i task, il task deve chiamare la funzione Enable. È importante che il task esegua tante chiamate a Enable quante sono le chiamate a Disable, dal momento che queste due funzioni tengono aggiornato un contatore di annidamento all'interno della struttura Exec (IDNestCnt). Si presti attenzione al fatto che dopo la chiamata alla funzione Disable se il task entra per qualche ragione in attesa, gli interrupt e la gestione multitasking vengono ripristinati temporaneamente fino a quando il task non riottiene il controllo, momento in cui tornano a essere disabilitati. Il task può entrare in attesa in molti modi, comunicando con un dispositivo, chiamando una funzione della libreria DOS, eseguendo le funzioni di I/O da console del C, chiamando esplicitamente funzioni di attesa come Wait...

Dispatch()

Questa funzione viene impiegata dal sistema per cedere il controllo al primo task presente nella lista di sistema TaskReady, nel corso di uno scambio

di controllo fra task. Si noti che se non esistono task pronti per essere eseguiti, la funzione incrementa di 1 il contatore DispCount contenuto all'interno della struttura ExecBase e attende che la CPU entri in exception.

Enable()

Viene impiegata dal sistema per riabilitare gli interrupt precedentemente disabilitati tramite una chiamata a Disable. In realtà questa funzione riabilita gli interrupt soltanto quando rileva che il parametro IDNestCnt della struttura ExecBase, ha raggiunto il valore zero. Infatti, quando questo parametro giunge a zero significa che l'ultima chiamata a Enable pareggia il numero delle chiamate a Disable. Questa gestione permette di annidare le chiamate di abilitazione e disabilitazione.

Exception()

Viene impiegata dal sistema per avviare l'esecuzione della routine di exception di un task. Se ne serve per esempio la funzione Dispatch quando rileva che il task che deve riottenere il controllo ha il flag TF_EXCEPT del parametro tc_Flags impostato.

ExitIntr()

Viene impiegata dal sistema per uscire correttamente da una routine di interrupt.

Forbid()

Permette di disabilitare lo scambio di controllo fra i task. Ciò fornisce al task un modo per proteggere le strutture di dati alle quali accede. Si noti tuttavia che i semafori forniscono un modo migliore per ottenere lo stesso risultato, anche se pretendono un'uniformità di convenzioni fra i task. Si tenga conto che Forbid non sospende gli interrupt, e mantiene aggiornato il contatore di annidamento TDNestCnt presente all'interno della struttura ExecBase.

Permit()

Permette di riabilitare lo scambio di controllo fra i task, precedentemente sospeso tramite Forbid. Come Forbid, anche Permit mantiene aggiornato il contatore di annidamento TDNestCnt presente all'interno della struttura ExecBase. In realtà, Permit non riattiva la gestione multitasking fino a quando il numero di chiamate effettuate a Forbid non è uguale al numero di chiamate effettuate a Permit.

Reschedule()

Viene impiegata dal sistema per ristabilire l'ordine di priorità di esecuzione per un task. Questa funzione permette a un task di abbandonare direttamente il controllo della CPU 68000 senza dover entrare in attesa di un evento (come l'arrivo di un segnale da una delle sue message port).

Schedule()

Viene impiegata dal sistema per stabilire l'ordine di priorità di esecuzione per un task. Ha un comportamento molto simile a quello della funzione SetTaskPri.

Supervisor()

Permette di porre la CPU 68000 in modo supervisor e di fargli eseguire una particolare routine. Prima di chiamarla occorre memorizzare nel registro A5 l'indirizzo della routine da eseguire nel modo supervisor, ed essere certi che nel registro A6 sia contenuto l'indirizzo base della libreria Exec. Il seguente esempio in Assembly ne mostra l'impiego.

```
        movea.l 4,A6
        lea 1$,A5
        jsr _LVOSupervisor(A6)
1$:
        ... ;inizio della routine
        rte
```

Switch()

Viene impiegata dal sistema per effettuare lo scambio di controllo fra due task. Questa funzione fornisce il modo per avviare l'elaborazione di un task senza attendere messaggi, segnali o interrupt.

Le funzioni di disegno e di gestione video della libreria Graphics

Introduzione

Questo capitolo riguarda le funzioni grafiche dell'Amiga, fuorché quelle relative al testo e alle animazioni che verranno discusse a parte. La libreria che le raccoglie è la libreria Graphics, contenuta nel ROM Kernel. Le funzioni presentate in questo capitolo possono essere suddivise in due gruppi principali: le funzioni video e le funzioni di disegno. In generale, le prime servono per definire e inizializzare i bitplane, le bitmap e le varie strutture di supporto alla gestione delle bitmap. Le funzioni di disegno, invece, attivano o azzerano i bit delle bitmap rendendo possibile disegnare sul video.

Un *bitplane* è un'area di memoria costituita da un qualsiasi numero di byte. Una *bitmap*, detta anche *raster*, è un insieme di bitplane; nell'Amiga una bitmap può essere composta da un minimo di un bitplane fino a un massimo di sei. Il numero dei bitplane che definiscono una bitmap dipende dal modo video che la caratterizza. La posizione di ciascun bit, all'interno di un bitplane, è in relazione con un particolare pixel dello schermo video su cui appare la bitmap.

Come mostra la Figura 2.1, i bitplane possono essere considerati come "strati" della bitmap; i bit sovrapposti dei bitplane vengono combinati per produrre un valore binario: questo valore viene poi associato al pixel del video che corrisponde alla stessa posizione nella bitmap, e determina il numero del registro colore che l'hardware associa a quel pixel. Il colore assegnato a ciascun registro viene controllato dal programma.

Le funzioni video

Le funzioni video effettuano allocazioni di memoria per i bitplane; molte di queste aree RAM, comunque, vengono impostate dalle routine di sistema a valori predeterminati. Tuttavia, a questo punto, nelle bitmap non esiste nessuna effettiva informazione di disegno; si tratta sostanzialmente di aree RAM pulite. Le funzioni video provvedono anche a creare in RAM le strutture che permettono la gestione dello schermo video, chiamando le funzioni di allocazione della memoria previste dalla libreria Exec. Una volta che la memoria e le strutture sono state impostate, tutto è pronto per definire il modo in cui le informazioni di disegno saranno disposte nella bitmap.

Quando si progetta un'applicazione, è necessario stabilire l'esatta composizione di tutti gli schermi video che occorrono. Nel corso del libro, gli schermi video verranno chiamati *view*, al fine di ricordare che per definirli occorre sempre una struttura View. Definire una view significa stabilire le caratteristiche e il numero delle diverse viewport che la compongono. Una *viewport* è un'area rettangolare dello schermo video, che ne occupa l'intera larghezza ma non necessariamente l'intera altezza.

Il risultato di questa procedura è una view composta da una o più viewport

rettangolari. Per rendere più chiari i concetti di view e viewport, prendiamo come esempio Intuition, il maggior "consumatore" delle funzioni della libreria Graphics.

Tutto ciò che in un dato istante appare sullo schermo video dell'Amiga è definito view, e in ogni istante viene visualizzata una e una sola view. Quindi, una view definisce tutto ciò che appare sullo schermo video. Una view è composta da un certo numero di viewport, ovvero fasce orizzontali di schermo dotate di caratteristiche proprie (palette e risoluzione orizzontale). In ambiente Intuition, i vari schermi sovrapposti non sono altro che viewport della stessa view. Intuition le gestisce per simulare la sovrapposizione degli schermi che l'utente varia a piacimento agendo con il mouse. A sua volta, una viewport può contenere un qualsiasi numero di layer, che in ambiente Intuition vengono impiegati per creare le finestre.

Se si vuole uno schermo video che non cambia nel tempo, è sufficiente caricare nell'hardware video la definizione della view (cioè le istruzioni per il

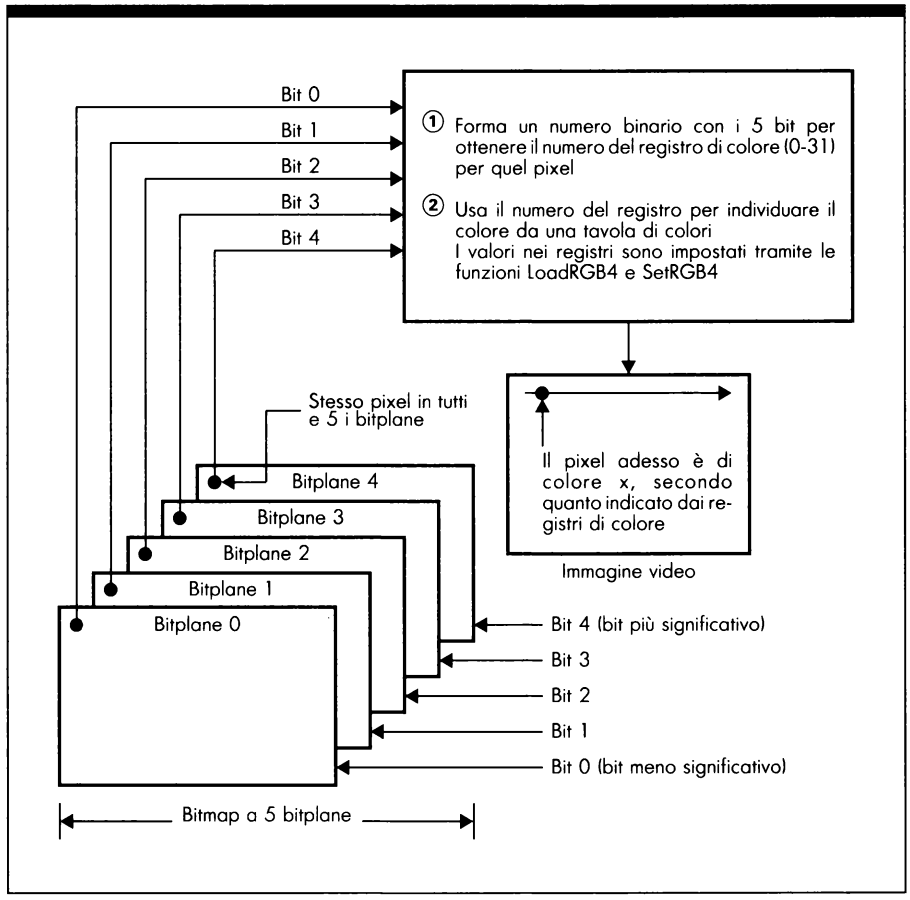


Figura 2.1: *Bitplane, bitmap, registri di colore e colori dei pixel*

Copper) una sola volta. L'hardware trasforma le istruzioni Copper in un quadro video che si ripete automaticamente 50 volte al secondo nel sistema PAL (60 volte al secondo nel sistema NTSC). In questo modo, grazie alla persistenza dell'immagine sulla retina, vediamo un'immagine video stabile, che non muta nel tempo.

Se invece si desidera uno schermo video in evoluzione, si devono definire diverse view e al momento opportuno caricare le loro istruzioni Copper nell'hardware video. Per disegnare sul video si devono quindi identificare e definire tutte le viewport che, opportunamente legate fra loro, costituiscono le diverse view di cui abbiamo bisogno. Generalmente, comunque, si continua a modificare la stessa view.

Le funzioni video possono essere suddivise in otto categorie: le funzioni per le bitmap e per le superbitmap, la funzione per inizializzare la struttura RastPort, le funzioni per le view, per le viewport, per definire le regioni, per i layer e per il controllo dell'hardware. Queste ultime si dividono a loro volta in diversi sotto-gruppi: le funzioni di controllo del Copper, le funzioni di controllo del Blitter e le funzioni per il controllo e il rilevamento di posizione del pennello elettronico nella scansione dei quadri video. Si veda anche il paragrafo relativo alle strutture grafiche.

Le funzioni di disegno

Le informazioni che costituiscono i disegni vengono memorizzate nelle bitmap attraverso le funzioni di disegno, ovvero la seconda grande categoria di funzioni grafiche del ROM Kernel di cui parliamo in questo capitolo. Il processo di scrittura delle informazioni nelle bitmap consiste in definitiva nell'impostare determinati bit nei bitplane che costituiscono la bitmap.

Le funzioni di disegno consentono di operare a vari livelli. Si possono trattare intere bitmap, regioni di bitmap, rettangoli all'interno di regioni, aree, linee, poligoni e così via. Si possono perfino impostare o azzerare direttamente specifici bit nelle bitmap. Quest'ultimo modo di operare rappresenta l'approccio di livello più basso e più versatile, ma anche quello più impegnativo; nella maggior parte dei casi, comunque, le funzioni di disegno di alto livello consentono di evitare di scendere a un controllo delle bitmap così diretto.

In genere le funzioni di disegno vengono prima di tutto utilizzate per impostare il modo di disegno, il pennello, i confini delle regioni su cui si desidera intervenire e così via; il disegno vero e proprio viene in un secondo momento. Il disegno viene così influenzato dal modo in cui sono stati fissati i parametri. Questo va fatto per tutte le parti della prima bitmap, dopo di che si può ripetere l'operazione per la successiva.

Le funzioni di disegno sono raggruppabili in sette categorie: le funzioni di accesso in lettura e scrittura ai colori del pixel, le funzioni di controllo del colore per i pennelli, le funzioni di controllo del colore RGB, le funzioni di controllo dei modi di disegno, le funzioni per il riempimento delle aree e le funzioni di disegno nelle regioni.

Le macro grafiche

In aggiunta alle funzioni video e di disegno, il sistema di programmazione della grafica mette a disposizione 16 macro-istruzioni: CINIT, CMOVE, CWAIT, CEND, SetOPen, SetDrPt, SetWrMsk, SetAfPt, ON_DISPLAY, OFF_DISPLAY, ON_SPRITE, OFF_SPRITE, RASSIZE, BNDRYOFF, ON_VBLANK e OFF_VBLANK. Ad eccezione della macro RASSIZE, definita nel file INCLUDE graphics/gfx.h, le definizioni precise di queste macro sono contenute nel file INCLUDE graphics/gfxmacros.h.

Le macro CINIT, CMOVE, CWAIT e CEND consentono al programmatore di definire un insieme d'istruzioni per il Copper. Esse possono essere impiegate per controllare l'hardware video in modo estremamente dettagliato. Le macro SetOPen, SetDrPt, SetWrMsk e SetAfPt consentono di accedere facilmente ai parametri della struttura RastPort.

Per queste macro-istruzioni in C è stata creata una sintassi molto simile a quella delle funzioni della libreria Graphics, così che i programmatori le possano impiegare proprio come se fossero funzioni che non restituiscono valori. Queste macro-istruzioni servono per semplificare al programmatore la scrittura dei sorgenti. In questo senso appaiono come dei rattoppi all'attuale versione della libreria Graphics, in attesa che in future versioni vengano tramutate in effettive funzioni della libreria.

Collegamenti tra le strutture grafiche

La chiave per definire un programma grafico (che adotti il single-playfield, il dual-playfield, il double-buffer, l'Hold And Modify e ogni altro modo video), consiste nell'identificare gli appropriati collegamenti tra le cinque maggiori strutture previste dalla libreria Graphics. Si tratta delle strutture RastPort, View, ViewPort, RasInfo e BitMap. Una volta che questi collegamenti sono stati definiti, si possono impostare i valori dei loro parametri. Anche per altri insiemi di strutture esistono collegamenti da instaurare (per esempio, le strutture Layer, AreaInfo...), ma non sono altrettanto vincolanti.

I collegamenti richiesti per le cinque strutture citate è illustrato nella Figura 2.2. I cinque rettangoli più grandi disposti verticalmente rappresentano le cinque strutture in questione. I rettangolini alla loro destra rappresentano le altre strutture a cui puntano le cinque principali. Le frecce del diagramma rappresentano i parametri-puntatori; i nomi di questi parametri, contenuti nelle strutture da cui le frecce si diramano, appaiono lungo ciascuna freccia.

La figura mostra che la struttura RastPort è collegata soltanto alla struttura BitMap, fra le cinque principali. Inoltre, la struttura RastPort punta alle strutture Layer, TmpRas, AreaInfo, GelsInfo e TextFont, e all'area dati denominata AreaPtrn.

La struttura View serve per definire una view. Dal momento che una view può essere composta da una o più viewport separate verticalmente sullo schermo, la struttura View possiede un puntatore che identifica la prima

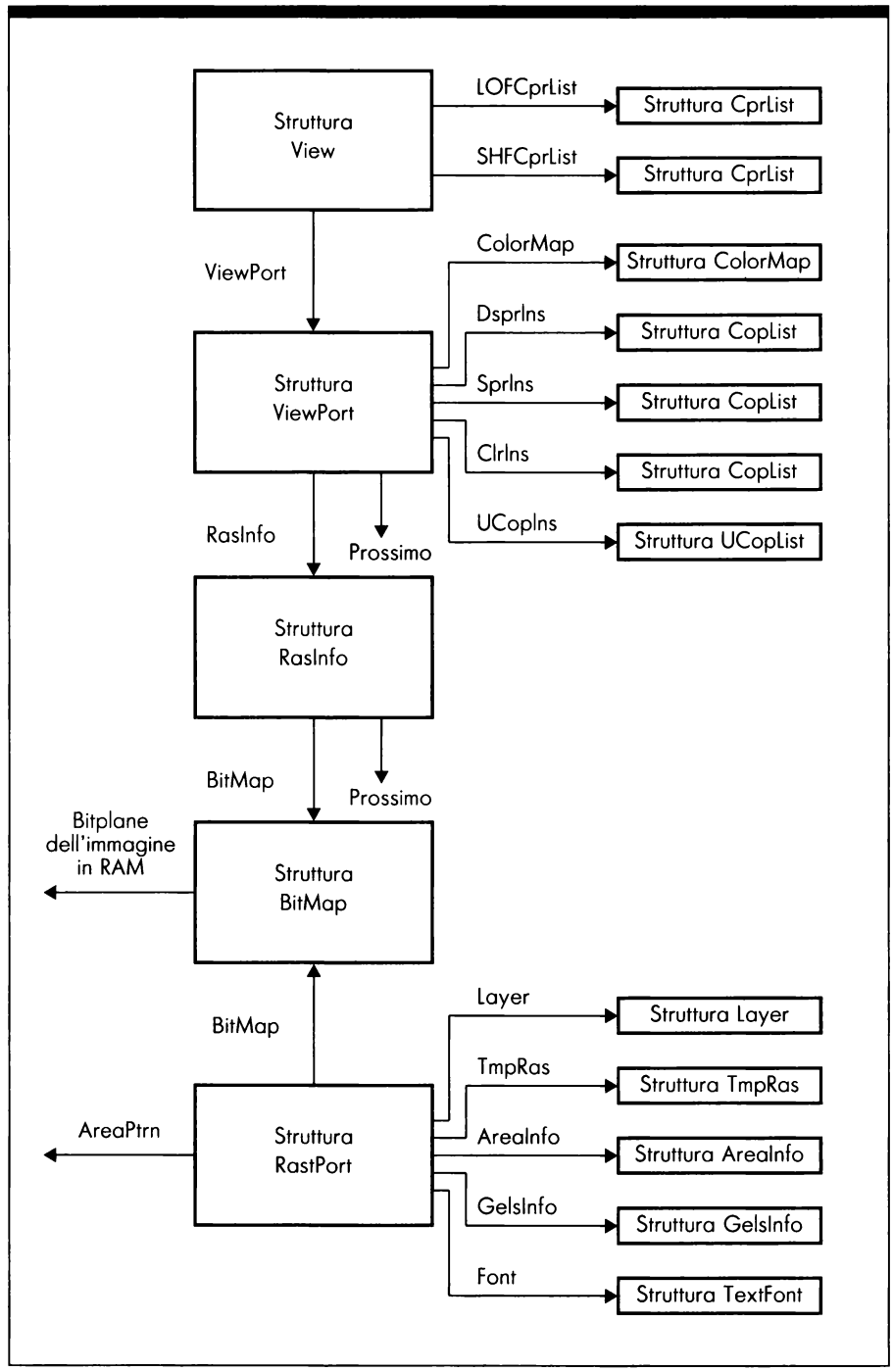


Figura 2.2: *Legami fra le principali strutture previste dalla libreria Graphics*

struttura ViewPort di una lista semplice di strutture ViewPort. Inoltre, contiene due puntatori che individuano in memoria due strutture CprList, ovvero le liste d'istruzioni del Copper per definire la view. Si noti che la struttura RastPort e la struttura View non sono direttamente collegate.

La lista di strutture ViewPort collegata alla struttura View è di tipo semplice: ogni nodo (cioè ogni struttura ViewPort) contiene infatti un solo puntatore (di nome Node) a un'altra struttura ViewPort, che quindi funge da puntatore al nodo successivo della lista.

Oltre a individuare la successiva struttura dello stesso tipo, ciascuna struttura ViewPort punta ad altre cinque strutture: il parametro ColorMap punta a una struttura ColorMap, i parametri DspIns, SprIns e ClrIns puntano a una struttura CopList, e infine il parametro UCopIns punta a una struttura UCopList. La struttura ColorMap definisce la tavola colori della viewport. Ciascuna viewport, infatti, può avere una tavola dei colori indipendente da quella delle altre; la tavola dei colori è in pratica la palette (tavolozza) dalla quale si possono attingere i colori per disegnare all'interno della corrispondente viewport. Le altre quattro strutture definiscono l'insieme completo delle istruzioni per il Copper relative a quella viewport.

La struttura ViewPort contiene anche un puntatore che serve per individuare in memoria una struttura di tipo RasInfo. Questa struttura contiene un puntatore a un'altra struttura RasInfo, che viene agganciata alla prima soltanto quando si usa il modo video dual-playfield (si veda l'appendice B). La struttura RasInfo risulta collegata anche alla struttura BitMap che punta ai bitplane in RAM. L'insieme di questi bitplane definisce la bitmap che costituisce la viewport.

Ciascun modo video impiega questi collegamenti in diverse combinazioni. Per esempio il modo single-playfield, il modo Hold And Modify e il modo Extra Half-Brite, uniscono a ogni struttura ViewPort (collegata a una particolare struttura View) una struttura RastPort, una struttura RasInfo, e una struttura BitMap. Invece, il modo double-buffer unisce a ogni struttura ViewPort una struttura RastPort, due strutture RasInfo, e due strutture BitMap.

Le strutture grafiche

Ci sono diverse importanti strutture con le quali lavorano le funzioni grafiche. In generale, per ciascuna struttura esiste una corrispondente funzione d'inizializzazione (per esempio le funzioni InitBitMap e InitRastPort). Queste funzioni impostano a valori di default alcuni parametri delle corrispondenti strutture indicate dai task. A loro volta, i programmi inizializzano altri parametri delle stesse strutture. Una volta che i parametri di una struttura sono stati completamente inizializzati, la struttura è pronta per svolgere un determinato compito. Per esempio, la struttura RastPort controlla tutti gli aspetti del disegno all'interno di una bitmap, detta anche "raster". In altre parole, agendo sui parametri della struttura RastPort relativa a una bitmap, si determina come appariranno i disegni che eseguiremo in quella bitmap.

L'importanza della struttura RastPort è tale che ne riportiamo qui l'intero

contenuto, corredandolo con una breve spiegazione dei suoi parametri.

```
struct RastPort {
        struct Layer *Layer;
        struct BitMap *BitMap;
        USHORT *AreaPtrn;
        struct TmpRas *TmpRas;
        struct AreaInfo *AreaInfo;
        struct GelsInfo *GelsInfo;
        UBYTE Mask;
        BYTE FgPen;
        BYTE BgPen;
        BYTE AOlPen;
        BYTE DrawMode;
        BYTE AreaPtSz;
        BYTE linpatcnt;
        BYTE dummy;
        USHORT Flags;
        USHORT LinePtrn;
        SHORT cp_x, cp_y;
        UBYTE minterms[8];
        SHORT PenWidth;
        SHORT PenHeight;
        struct TextFont *Font;
        UBYTE AlgoStyle;
        UBYTE TxFlags;
        UWORD TxHeight;
        UWORD TxWidth;
        UWORD TxBaseline;
        WORD TxSpacing;
        APTR *RP_User;
        ULONG longreserved[2];
        UWORD wordreserved[7];
        UBYTE reserved[8];
};
```

Il parametro Layer punta a una struttura Layer che consente alla struttura RastPort di controllare i parametri di disegno per le bitmap dei layer. La struttura Layer viene usata direttamente dalle funzioni LockLayerRom, UnlockLayerRom, CopySBitMap e SyncSBitMap; viene inoltre impiegata dalle funzioni della libreria Layers.

Il parametro BitMap punta a una struttura BitMap che consente alla struttura RastPort di controllare i parametri di disegno per una bitmap. La struttura BitMap punta a una bitmap in RAM e ne definisce le dimensioni. Viene impostata attraverso la funzione InitBitMap; altre funzioni utilizzano indirettamente la struttura BitMap.

Il parametro AreaPtrn punta in RAM a una regione dati contenente la matrice grafica (o retino) di riempimento delle aree. Questa regione dati

definisce i pixel che costituiscono la matrice usata dalle funzioni AreaEnd e Flood. Per impostare il blocco di memoria che definisce la matrice per i riempimenti di aree si può usare la macro-istruzione SetAfPt, definita nel file INCLUDE gfxmacros.h.

Il parametro TmpRas punta a una struttura TmpRas, utilizzata per controllare un buffer ausiliario. Questo buffer mantiene aree temporanee più piccole della bitmap nel corso del riempimento di aree effettuato dalle funzioni AreaDraw, AreaMove e AreaEnd. La struttura TmpRas viene inizializzata attraverso la funzione InitTmpRas.

Il parametro AreaInfo punta a una struttura AreaInfo che controlla un buffer ausiliario, il cui compito è conservare i vertici che definiscono un insieme di aree destinate a essere riempite tramite la funzione AreaEnd. La struttura AreaInfo viene inizializzata attraverso la funzione InitArea.

Il parametro GelsInfo punta a una struttura GelsInfo, utilizzata per legare alla definizione della bitmap di schermo i dettagli relativi alle animazioni di sprite virtuali e bob (blitter object). La struttura GelsInfo viene usata direttamente dalle funzioni di animazione della libreria Graphics.

Il parametro Mask è una maschera che indica quali bitplane di una bitmap sono interessati dalle operazioni di disegno. Il parametro Mask non è controllato direttamente da nessuna funzione della libreria Graphics. Può essere impostato sia usando una semplice istruzione di assegnazione dei parametri di struttura, sia usando la macro-istruzione SetWrMsk.

Il parametro FgPen mantiene il colore associato alla penna di primo piano (la penna relativa al tratto), cioè il colore con cui vengono disegnati i pixel di primo piano nella bitmap, e viene impostato tramite la funzione SetAPen.

Il parametro BgPen mantiene il colore associato alla penna di sfondo, cioè il colore con cui vengono disegnati i pixel dello sfondo, e viene impostato tramite la funzione SetBPen.

Il parametro AOlPen è il valore della penna utilizzata per disegnare i bordi delle aree nella bitmap. Questa penna è impiegata per contornare le aree prodotte dalle funzione AreaEnd. Il parametro AOlPen viene usato anche nelle operazioni di riempimento che coinvolgono la funzione Flood. Viene impostato tramite la macro-istruzione SetOPen.

Il parametro DrawMode deve contenere il modo di disegno, e viene impostato tramite la funzione SetDrMd.

Il parametro AreaPtSz deve contenere il numero delle word impiegate per definire la matrice grafica di riempimento utilizzata da funzioni come AreaEnd e Flood. Viene impostato tramite la macro-istruzione SetAfPt.

Il parametro linpatcnt contiene il valore del contatore relativo alla matrice di continuità per il disegno delle linee; serve alle funzioni Draw e PolyDraw durante il disegno delle linee tratteggiate. Viene controllato direttamente dal sistema.

Il parametro dummy è un byte fittizio inserito per tenere adeguatamente allineata la struttura RastPort nei vari aggiornamenti del software.

Il parametro Flags è un insieme di flag usati dal software sistema per controllare alcuni dettagli interni del processo di creazione del disegno. Non viene alterato direttamente da nessuna funzione della libreria Graphics. I flag di questo parametro sono definiti nel file INCLUDE rastport.h.

Il parametro LinePtrn occupa una word e contiene la matrice di continuità che le funzioni Draw e PolyDraw impiegano per disegnare le linee. In pratica, nel disegno di una linea queste funzioni tracciano ripetutamente la word della matrice di continuità. Pertanto, i bit a 1 e a 0 della matrice definiscono i pixel attivi e quelli non attivi di una linea tratteggiata. Per definire il valore contenuto in questo parametro si può usare la macro-istruzione SetDrPt.

I parametri cp_x e cp_y sono le coordinate x e y del pixel da rappresentare nel sistema di coordinate della bitmap. In pratica, definiscono la posizione in cui si verificherà la prossima operazione di disegno. Ogni volta che una funzione (AreaDraw, AreaMove, AreaEnd, Draw, Flood, Move o PolyDraw) muove la penna, questi valori vengono alterati. Altre funzioni di disegno sono interessate indirettamente a tali valori. Anche le funzioni di gestione dei testi contenute nella libreria Graphics modificano i valori in questione nel momento in cui disegnano un testo nella bitmap.

Il parametro minterms[8] è un insieme di otto byte utilizzati per controllare le operazioni logiche del Blitter. Questi byte sono impostati e controllati dalle funzioni BltBitMap, ClipBlit e da altre funzioni che coinvolgono il Blitter.

I parametri PenWidth e PenHeight mantengono la larghezza e l'altezza della penna espresse in pixel. Non esistono funzioni della libreria Graphics che controllino direttamente questi parametri.

Il parametro Font punta alla struttura TextFont utilizzata dalle funzioni di gestione dei testi contenute nella libreria Graphics.

I parametri AlgoStyle, TxFlags, TxHeight, TxWidth, TxBaseline e TxSpacing contengono i valori da utilizzare nel disegno dei testi. Il parametro AlgoStyle occupa un byte e viene usato per i cambiamenti di stile (nero, corsivo, e così via...) nella definizione della fonte-carattere per il testo. TxFlags contiene il valore del parametro Flags; è correlato al parametro Flags della struttura TextFont. TxHeight, TxWidth, TxBaseline e TxSpacing contengono i valori di altezza e larghezza dei caratteri, la linea di base e l'interlinea. Anch'essi sono correlati ai parametri della struttura TextFont. Per una più estesa trattazione dell'argomento si veda il capitolo 4.

Il parametro RP_User punta a una reply port (una message port) del task che sta impiegando la struttura RastPort per eseguire operazioni di disegno. Questo parametro non viene controllato direttamente da nessuna funzione della libreria Graphics.

I parametri longreserved[2], wordreserved[7] e reserved[8] sono riservati per future evoluzioni o ampliamenti della struttura RastPort.

Per comprendere l'importanza della struttura RastPort, la parola chiave è *controllo*; RastPort è la principale struttura usata per il controllo delle operazioni di disegno in una bitmap (in molte pubblicazioni dedicate all'Amiga, i termini *raster* e *raster bitmap* vengono utilizzati come sinonimi di *bitmap*; per evitare inutili confusioni ai lettori, in questo libro utilizzeremo sempre e soltanto il termine *bitmap*).

Una volta che i parametri della struttura RastPort sono stati impostati dalle opportune funzioni (come per esempio SetAPen e SetBPen) o direttamente dal task grazie alla chiamata di una funzione di disegno (come per esempio AreaEnd o RectFill), questi parametri controllano che una determinata parte della bitmap sia stata effettivamente disegnata, rilevando in che modo i suoi

bit sono stati attivati o azzerati e in che modo le alterazioni vengono interpretate dall'hardware video. La struttura RastPort cambia dinamicamente durante il processo di generazione del disegno.

Oltre alla struttura RastPort, le funzioni video e di disegno coinvolgono le seguenti strutture:

- la struttura RasInfo contiene due parametri che definiscono la posizione dell'angolo superiore sinistro della bitmap che definisce una viewport all'interno di una bitmap più estesa. Inoltre contiene un puntatore a una struttura BitMap. RasInfo va impostata con tre semplici istruzioni di programma.

- La struttura BitMap indica al sistema dove si trovano nella chip RAM i bitplane che costituiscono la bitmap, e quali caratteristiche hanno (il loro numero, le loro dimensioni...). La struttura BitMap viene inizializzata attraverso la funzione InitBitMap.

- La struttura ViewPort contiene i parametri per controllare la composizione video delle viewport di una specifica view di schermo. La struttura ViewPort viene inizializzata dalla funzione InitVPort.

- La struttura View definisce una view, che in pratica è uno schermo video dotato delle caratteristiche definite dai parametri della struttura View. Un programma a volte deve visualizzare in tempi diversi più di una view, ognuna delle quali dev'essere definita da una struttura View. Ogni view può essere composta da diverse viewport, ognuna definita da un'opportuna struttura ViewPort. La struttura View viene inizializzata tramite la funzione InitView.

- La struttura Region contiene i parametri che controllano la definizione di una regione. Una *regione* è una sotto-sezione di bitmap interessata da una certa operazione di disegno. La struttura Region viene inizializzata dalla funzione NewRegion. Una volta che una regione è stata inizializzata, si possono usare le funzioni AndRectRegion, OrRectRegion e XorRectRegion per estenderne la definizione.

- La struttura Rectangle contiene i parametri per definire i rettangoli all'esterno dei quali le funzioni di disegno non devono agire. In pratica, queste delimitazioni rettangolari indicano alle funzioni di disegno dove effettuare il clip. Il clip è quell'operazione che disabilita le operazioni di disegno al di fuori di una particolare delimitazione rettangolare. La struttura Rectangle viene inizializzata dalle istruzioni per definire i punti di confine dei rettangoli di delimitazione (clipping-rectangle).

- La struttura Layer contiene i parametri per definire le bitmap dei layer. I *layer* sono utilizzati da Intuition per realizzare schermi multi-finestra, in cui le diverse parti della presentazione video provengono da diverse bitmap di layer, eventualmente sovrapposte. Le strutture Layer

possiedono una priorità video che indica quale layer appare "in superficie", quale appena sotto e così via). La struttura Layer viene inizializzata da due funzioni della libreria Layers: CreateUpFrontLayer e CreateBehindLayer.

- La struttura ColorMap consente di definire i valori dei registri di colore del sistema hardware nella visualizzazione di una particolare viewport. La struttura ColorMap viene inizializzata dalla funzione GetColorMap.

- La struttura CopList contiene i parametri per controllare l'uso e la definizione di un insieme d'istruzioni esplicite intermedie e di basso livello per il Copper. Queste istruzioni sono necessarie per definire la rappresentazione video di una viewport. La struttura CopList viene definita e inizializzata dalla funzione MakeVPort. Ciascuna struttura ViewPort possiede un insieme di tre puntatori a strutture CopList per definire le liste di istruzioni intermedie del Copper per i playfield e gli sprite che devono apparire sul quadro video.

- La struttura CprList contiene i parametri per controllare l'uso e la definizione dell'insieme finale d'istruzioni hardware esplicite di basso livello per il Copper. Viene inizializzata dalle routine di sistema.

- La struttura AreaInfo contiene i parametri per definire le operazioni di riempimento di aree effettuate dalle funzioni AreaDraw, AreaMove e AreaEnd. Viene inizializzata dalla funzione InitArea.

- La struttura TmpRas contiene parametri utili durante le operazioni delle funzioni AreaDraw, AreaMove e AreaEnd. Viene inizializzata dalla funzione InitTmpRas.

- La struttura ClipRect collega fra loro i rettangoli di delimitazione usati per definire le regioni su cui dev'essere effettuato il clip durante il disegno, e per aggiornare le bitmap dei layer. Viene utilizzata indirettamente dalle funzioni di gestione delle regioni, e – in modo più esteso – dalle funzioni della libreria Layers.

- La struttura UCopList viene usata per mantenere i collegamenti tra le varie liste Copper definite dal programmatore. Queste liste vengono create dalle istruzioni di controllo del Copper CINIT, CMOVE, CWAIT e CEND, a cui la struttura UCopList si riferisce direttamente.

- La struttura Layer_Info viene usata per controllare la bitmap di un layer quando più di un task può accedervi per operazioni di disegno. La struttura viene utilizzata indirettamente dalle funzioni LockLayerRom e UnlockLayerRom, e – in modo più esteso – dalle funzioni della libreria Layers.

sistemi di coordinate

Vi sono tre sistemi di coordinate da imparare a conoscere, prima di poter programmare con le funzioni della libreria Graphics: il sistema di coordinate della bitmap, il sistema di coordinate della view e il sistema di coordinate della viewport. Si dovrebbe esaminare le funzioni una per una, per essere sicuri del sistema di coordinate che adottano.

Il sistema di coordinate della bitmap è il più esteso. In questo sistema i valori x e y possono entrambi arrivare a 1024. Vi ricorrono la maggior parte delle funzioni di disegno, tra cui le funzioni di riempimento di area (AreaDraw, AreaMove e AreaEnd) e quelle di disegno di linee (Draw, Move e PolyDraw).

Il sistema di coordinate della view è quello che segue dal punto di vista dell'estensione. Le coordinate massime per questo sistema dipendono dal modo video scelto. Per esempio, limitandoci al sistema PAL se si sceglie la bassa risoluzione con il modo video non-interlace, i massimi valori di x e y sono 320 e 256. Se si sceglie l'alta risoluzione in modo video interlace, i valori massimi diventano rispettivamente 640 e 512. L'hardware del video dell'Amiga, in effetti, è limitato a valori massimi di 368 e 312 in bassa risoluzione e modo non-interlace, 736 e 624 in alta risoluzione e modo interlace. Tuttavia, in overscan, cioè uscendo dai limiti 320 x 256 e 640 x 512, non si può essere certi che il monitor riesca a visualizzare l'intera view: questi limiti servono proprio per garantire con qualsiasi monitor un buon grado di centratura dell'immagine.

Il sistema di coordinate della viewport è il più ristretto. Le coordinate massime di questo sistema dipendono ancora una volta dal modo video scelto. Inoltre dipendono dalle dimensioni della viewport più grande. Se si definisce una viewport che copre l'intero schermo video, essa risulterà grande quanto la view in cui è contenuta, e quindi le sue coordinate massime saranno quelle della view.

La creazione della view

Per creare uno schermo video completo sono richiesti sette passi fondamentali. Primo, si deve usare la funzione AllocRaster per predisporre abbastanza memoria da contenere tutti i bitplane della bitmap. Ci si deve inoltre assicurare che esista abbastanza memoria per le seguenti strutture:

- tutte le strutture ViewPort della view.
- Tutte le strutture View con le quali interagisce il task.
- Tutte le strutture RastPort.
- Tutte le strutture BitMap.
- Tutte le strutture ColorMap.

- Tutte le strutture Layer.
- Tutte le strutture Layer_Info.
- Tutte le strutture RasInfo.
- Tutte le strutture Region.
- Tutte le strutture Rectangle.
- Tutte le strutture CopList.
- Tutte le strutture CprList.
- Tutte le strutture UCopList.
- Tutte le strutture TmpRas.

Questa lista contiene la maggior parte delle strutture relative alla grafica. Le esigenze di memoria per le strutture variano naturalmente a seconda del task; molti task grafici, infatti, non utilizzano tutte le strutture elencate.

Secondo, si devono definire tutte le strutture ViewPort che saranno combinate nella struttura View finale. Ciascuna delle strutture ViewPort costituisce un nodo di una lista di strutture ViewPort. Ogni nodo di questa lista possiede una variabile di puntamento alla struttura ViewPort che segue nella lista. L'insieme delle strutture ViewPort definisce la struttura View complessiva che verrà impiegata per definire l'immagine finale sullo schermo.

Terzo, si deve impostare la struttura RasInfo, che dice al sistema dove rintracciare in RAM l'inizio della bitmap da usare con la struttura ViewPort. Il sistema viene inoltre informato di come dev'essere collocata l'area video che rappresenta la viewport, rispetto alla definizione completa dell'area di disegno in memoria (la bitmap).

Quarto, si deve inizializzare la struttura View usando la funzione InitView. Si collega la prima struttura ViewPort alla struttura View. Si noti che siccome le strutture ViewPort sono legate tra loro, questa operazione collegherà alla struttura View anche le altre strutture ViewPort.

Quinto, si deve usare la funzione MakeVPort per creare le effettive istruzioni video di cui si servirà il Copper per definire ciascun quadro della presentazione video. Altre istruzioni video per il Copper possono provenire da sorgenti diverse.

Sesto, si devono conglobare tutte le istruzioni Copper per formare una lista completa d'istruzioni per il Copper. Il Copper ricorre a questa lista per controllare il pennello elettronico nel suo spostamento sullo schermo, così da generare ogni elemento della presentazione video. Le istruzioni per il Copper provengono da tre sorgenti: dalla funzione MakeVPort, dalle routine di animazione, dalla struttura UCopList prevista dal programmatore. La funzione MrgCop le unisce insieme (effettua un *merge*) per definire la presentazione video di un singolo quadro.

Si osservi che queste istruzioni video possono cambiare di quadro in quadro. Un intero insieme d'istruzioni (o una parte di esso) può cambiare durante l'intervallo di vertical-blanking (il periodo di tempo durante il quale il pennello elettronico - completato un quadro - si spegne per riportarsi in alto a sinistra e iniziare il successivo; fra l'inizio di un quadro e l'inizio del successivo trascorre un cinquantesimo di secondo nel sistema PAL e un sessantesimo nel sistema NTSC). Il motivo per cui gli interventi sul contenuto dello schermo vengono eseguiti durante il periodo di vertical-blanking è che in questo modo si evitano fastidiosi sfarfallii dello schermo.

L'ultima fase consiste nell'impiegare la funzione LoadView per caricare nell'hardware video la view rappresentata dalle istruzioni Copper. Questa è l'operazione che produce effettivamente l'apparizione dell'immagine sullo schermo. Fino a quel momento tutte le informazioni video esistono soltanto in memoria, sotto forma di bitmap e istruzioni Copper. La funzione LoadView prende le istruzioni raccolte nella lista Copper e le invia all'hardware video di sistema.

AllocRaster

Sintassi di chiamata della funzione

```
bitplane_Pointer = AllocRaster (width, height)
D0                              D0    D1
```

Scopo della funzione

Questa funzione chiama le apposite routine di sistema al fine di allocare memoria sufficiente per un bitplane. Se l'allocazione riesce, restituisce l'indirizzo dell'area di memoria RAM allocata, altrimenti restituisce il valore zero.

Argomenti della funzione

width	Numero di bit in direzione x nel sistema di coordinate della bitmap.
height	Numero di bit in direzione y nel sistema di coordinate della bitmap.

Discussione

Ci sono quattro funzioni della libreria Graphics che agiscono specificamente con le bitmap: AllocRaster, FreeRaster, InitTmpRas e ScrollRaster. Si vedano anche le spiegazioni relative a queste funzioni.

La *bitmap* è formata da un certo numero di bitplane, cioè da un certo numero di aree di memoria, la cui quantità dipende dal numero di colori che si desiderano. In molte pubblicazioni i termini *raster*, *bitmap* e *raster bitmap* sono usati come sinonimi; in questo volume useremo sempre il termine bitmap.

AllocRaster serve per allocare lo spazio di memoria destinato ai bitplane che si devono creare per la bitmap, ma crea un solo bitplane per volta. Non prevede argomenti puntatori e perciò non lavora direttamente con nessuna struttura; restituisce semplicemente l'indirizzo dell'area di RAM allocata. Quindi la funzione AllocRaster va chiamata tante volte quante sono i bitplane necessari per la bitmap.

AllocRaster lavora con due argomenti: width e height. Per esempio, se si desidera allocare un bitplane per creare uno schermo video in bassa risoluzione senza interlace, si dovrà indicare il valore 320 nell'argomento width e il valore 256 nell'argomento height (200 nel sistema NTSC).

Il parametro width viene automaticamente convertito al multiplo di 16 immediatamente superiore alla grandezza indicata, dal momento che ogni riga dev'essere costituita da un numero pari di byte. Ad esempio la chiamata:

```
plane = AllocRaster (100, 100);
```

allocherà un'area di memoria di 1400 byte: 100 corrisponde a 112 pixel; 112 pixel/8 bit = 14 byte; 14 byte x 100 righe = 1400 byte. Si veda anche la descrizione della macro RASSIZE che esegue automaticamente questo calcolo.

Si può usare un breve ciclo FOR in linguaggio C per allocare gli spazi di memoria necessari per diversi bitplane, chiamando la funzione AllocRaster a ogni esecuzione del ciclo. In questo modo si può allocare memoria per tutti i bitplane necessari per la bitmap (fino a 6 nel modo Hold And Modify). Il ciclo iterativo FOR sarà del tipo:

```
for (i = 0; i<depth; i++) {
     MyBitMap.Planes[i] = (PLANEPTR) AllocRaster (width, height);
}
```

La Tavola 2.1 mostra il numero di byte richiesti dalle diverse combinazioni di modi video. Si possono usare queste informazioni per valutare le esigenze di memoria video.

Il sistema hardware dell'Amiga possiede 32 registri per controllare i colori sullo schermo video. Ciascuno di questi registri contiene 12 bit, e può quindi rappresentare 4096 valori diversi, cioè 4096 variazioni cromatiche. Ogni tonalità è ottenuta per sintesi additiva dei tre colori fondamentali rosso, verde e blu. In ogni registro di colore, quattro bit controllano la sfumatura del rosso, quattro bit quella del verde e quattro bit quella del blu. Quindi, per ogni colore

Tavola 2.1: *Memoria richiesta dai modi video*

Dimensioni dell'immagine video	Modo video	Numero di byte per bitplane
320 x 256	Bassa risoluzione	10K = 10240 byte
320 x 512	Bassa risoluzione con interlace	20K = 20480 byte
640 x 256	Alta risoluzione	20K = 20480 byte
640 x 512	Alta risoluzione con interlace	40K = 40960 byte

fondamentale si può scegliere fra 16 valori d'intensità.

Quando si definisce la palette di un'immagine video, si inizializza una tavola colormap, sotto forma di array di word, nella quale i primi 12 bit di ogni word vengono copiati nei registri di colore per rendere attiva la palette. Il colore 1 è associato al registro di colore 1, il colore 2 al registro di colore 2 e così via.

Il colore zero è riservato allo sfondo. Qualsiasi area dello schermo video su cui non siano presenti oggetti è considerata sfondo. Il colore dello sfondo viene sfruttato anche per l'area del bordo, al di fuori della finestra di schermo. Il colore 1 costituisce spesso la scelta alternativa per un playfield a due colori. Si veda l'appendice A per la definizione di playfield. Ecco il numero dei bitplane corrispondenti al numero di colori impiegati per uno schermo single-playfield:

Numero di colori	Numero di bitplane richiesti
2	1
Da 3 a 4	2
Da 5 a 8	3
Da 9 a 16	4
Da 17 a 32	5

A seconda del numero di bitplane impiegati, a ogni pixel possiamo assegnare 2^n valori variabili fra 0 e $2^n - 1$, dove n è il numero di bitplane. Questi numeri indicano all'hardware quali sono i registri di colore dai quali deve prelevare i colori da associare ai rispettivi pixel di schermo. Così, se in uno schermo a 2 bitplane in corrispondenza di un pixel i bit di entrambi i bitplane sono a zero, l'hardware visualizzerà quel pixel con il colore contenuto nel registro colore 0. Se invece i due bit sono entrambi a uno, l'hardware visualizzerà il pixel con il colore contenuto nel registro 3.

AndRectRegion

Sintassi di chiamata della funzione

AndRectRegion (region, rectangle)
AØ A1

Scopo della funzione

Questa funzione effettua l'operazione logica AND bidimensionale tra un rettangolo di delimitazione (clipping-rectangle) e una regione, in modo che successivamente le funzioni di disegno accedano soltanto alla parte di regione compresa nel rettangolo di delimitazione. Il risultato è una nuova regione composta solo dalla parte del rettangolo di delimitazione comune anche alla regione precedente.

Argomenti della funzione

region — Indirizzo della struttura Region che definisce la regione che si desidera intersecare con il rettangolo di delimitazione. Questa struttura, dopo l'esecuzione della funzione, conterrà la definizione della nuova regione.

rectangle — Indirizzo della struttura Rectangle che definisce il rettangolo di delimitazione.

Discussione

Ci sono sei funzioni di disegno della libreria Graphics che riguardano specificamente le regioni e i rettangoli di delimitazione: AndRectRegion, ClearRegion, DisposeRegion, NewRegion, OrRectRegion e XorRectRegion. Un *rettangolo di delimitazione* è una delimitazione all'interno di una regione al di fuori della quale le funzioni di disegno non devono disegnare; il rettangolo di delimitazione serve per indicare alle funzioni della libreria Graphics i confini oltre i quali effettuare il clip dei disegni. Per esempio, se la finestra di Intuition (e quindi il layer) su cui desideriamo disegnare è solo parzialmente in

vista, le parti in vista si possono sicuramente ricondurre ad alcuni rettangoli di delimitazione, al di fuori dei quali le funzioni di disegno non devono disegnare per non compromettere il contenuto del layer di livello superiore. Ognuno di questi rettangoli di delimitazione è definito da una struttura Rectangle, strutture che collegate fra loro possono fare capo a una struttura ClipRect valida per tutto il layer. La definizione di tutti questi rettangoli costituisce una regione, che evidentemente può avere qualsiasi forma geometrica ottenibile tramite la composizione di rettangoli. La regione, una volta agganciata al layer tramite la funzione InstallClipRegion della libreria Layers, individuerà le parti della bitmap del layer all'interno delle quali le funzioni di disegno possono disegnare, e quindi indicherà implicitamente anche le parti della bitmap sulle quali le funzioni di disegno non devono disegnare.

La funzione AndRectRegion permette di delimitare una particolare porzione rettangolare di regione, trasformandola in una nuova regione; se la nuova regione viene agganciata al layer, le successive operazioni di disegno agiranno soltanto sulla nuova regione della bitmap.

Per comprendere meglio il concetto di rettangolo di delimitazione si supponga di voler disegnare un quadrato pieno in un layer. Ci sono almeno due modi per raggiungere questo scopo. Il primo è quello più ovvio: si decide in che posizione nel layer si vuole questo quadrato e s'impiegano le funzioni AreaDraw, AreaMove e AreaEnd (o la funzione RectFill) per definirlo e disegnarlo. Si usano le funzioni di disegno (SetAPen, SetBPen...) per preparare la struttura RastPort a controllare adeguatamente i colori con cui verrà effettuato il disegno del quadrato e dei bordi. Quando infine verranno eseguite le funzioni AreaEnd o RectFill, il quadrato verrà disegnato nella bitmap.

In alternativa, si possono usare insieme le funzioni AndRectRegion e OrRectRegion su di una regione definita attraverso due rettangoli di delimitazione per restringere l'esecuzione del disegno a una precisa porzione della bitmap. Per fare ciò si definisce, per prima cosa, una struttura Rectangle impiegando quattro semplici istruzioni che definiscano i vertici di un rettangolo di delimitazione grande quanto l'intera bitmap del layer.

Si chiama poi la funzione NewRegion per creare una nuova regione di misura zero. Quindi si invoca la funzione OrRectRegion per includere questo grande rettangolo di delimitazione nella nuova regione. L'operazione fa in modo che la regione venga portata alle dimensioni del rettangolo di delimitazione, cioè quelle dell'intera bitmap.

Usando la stessa procedura, si definisce poi un secondo rettangolo di delimitazione ma di dimensioni pari al quadrato in questione. Si chiama quindi la funzione AndRectRegion. In questo modo si delimita quel rettangolo all'interno della regione, cosicché in seguito disporremo soltanto di quella ristretta porzione della bitmap. Ora abbiamo una definizione di regione e dobbiamo agganciarla al layer inserendo l'indirizzo della struttura Region nel parametro DamageList della struttura Layer (il vecchio indirizzo contenuto in questo parametro lo salviamo temporaneamente, per poi ripristinarlo quando avremo terminato di disegnare nel layer).

A questo punto occorre eseguire immediatamente una chiamata alla funzione BeginUpdate della libreria Layers, la quale normalmente predispone il layer perché si possa accedere soltanto alle parti danneggiate da altri layer

ed eventualmente tornate alla luce. Dal momento che al posto della lista delle aree danneggiate abbiamo inserito provvisoriamente l'indirizzo della nostra regione, la funzione predispone il layer perché le funzioni di disegno accedano solo a essa. Questo significa che tutte le operazioni di disegno che vengono effettuate prima di eseguire la funzione EndUpdate, sempre della libreria Layers, disegneranno solo all'interno della regione rettangolare che abbiamo creato (per le spiegazioni delle funzioni BeginUpdate e EndUpdate si veda il capitolo 5). Inoltre, il sistema continua a tenere conto anche in questo caso del fatto che il layer al quale accediamo potrebbe essere parzialmente coperto, preoccupandosi di effettuare un clip automatico sulla nostra regione qualora sconfini geometricamente nelle parti del layer coperte da altri layer.

Ora, come in una normale procedura di disegno, si usano le funzioni di disegno (SetAPen, SetBPen...) per preparare la struttura RastPort del layer, così che essa possa controllare i colori per il disegno, i bordi e così via per il rettangolo da far apparire. Si usa poi la funzione RectFill per riempire il layer con il colore che si desidera. Normalmente l'intera bitmap del layer riceverebbe questo colore (salvo le parti coperte da altri layer), ma avendo definito un'opportuna regione di clip, soltanto l'area della bitmap da essa circoscritta sarà riempita con il colore prescelto.

Si può estendere questo processo di disegno indiretto impiegando ulteriormente la funzione AndRectRegion. Per esempio, prima di collegare la regione di clip al layer e chiamare la funzione BeginUpdate, si può definire un altro rettangolo di delimitazione. Si può poi applicare ancora la funzione AndRectRegion fra l'ultima regione ottenuta e il rettangolo di delimitazione per creare una nuova regione.

Il percorso da seguire partendo da questo punto dipende dalla posizione di uno dei due rettangoli in relazione a quella dell'altro. Come prima ipotesi assumiamo che il secondo rettangolo di delimitazione si trovi nell'identica posizione del primo all'interno della bitmap. In questo caso, quando si chiama la funzione BeginUpdate il risultato sarà lo stesso della prima volta; la definizione di regione non cambia perché l'operazione di AND è avvenuta fra una regione e un rettangolo di delimitazione delle stesse dimensioni e nella stessa posizione.

Facciamo ora l'ipotesi che il secondo rettangolo di delimitazione non si sovrapponga al primo; che abbia dimensioni diverse e che sia stato collocato in una posizione del tutto diversa all'interno della bitmap. In questo caso, chiamando la funzione AndRectRegion si otterrà una regione nulla, dal momento che la regione e il rettangolo indicati come argomenti hanno intersezione nulla; se agganciamo alla struttura Layer questa regione e chiamiamo la funzione BeginUpdate, tutte le funzioni di disegno che successivamente manderemo in esecuzione non eseguiranno più alcun disegno sul layer.

Il terzo caso, mostrato dalla Figura 2.3, è il più interessante, dato che illustra il vero modo di operare della funzione AndRectRegion. Qui si assume che il secondo rettangolo di delimitazione si sovrapponga parzialmente al primo. Tutte le operazioni di disegno saranno quindi circoscritte al rettangolo intersezione: l'intersezione è un rettangolo non più grande del più piccolo tra i due rettangoli di delimitazione. La misura precisa di questo rettangolo finale

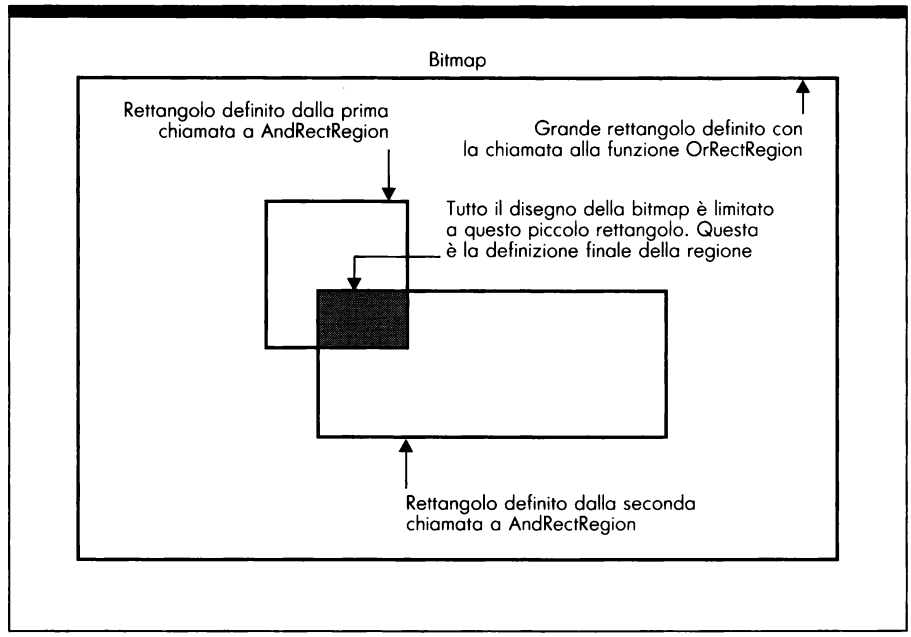


Figura 2.3: *Come opera la funzione AndRectRegion*

dipende dalla misura e dalla posizione relativa dei due rettangoli.

In conclusione, abbiamo visto che la funzione AndRectRegion permette, data una regione e un rettangolo di delimitazione, di ottenere una nuova regione che è l'intersezione geometrica della regione iniziale e del rettangolo di delimitazione, cioè l'area comune a entrambe. Abbiamo anche visto che le operazioni con le regioni sono soltanto operazioni geometriche, dal momento che le regioni sono soltanto definizioni di confini. Solo quando si associa una regione a un layer, la regione viene collegata a una bitmap. In pratica, potremmo pensare alle regioni come a fogli bianchi all'interno dei quali ritagliamo dei rettangoli attraverso i quali guardare. Una volta che il foglio è stato così preparato, lo possiamo disporre su una bitmap per limitare le azioni delle funzioni di disegno soltanto alle parti di bitmap che i rettangoli ritagliati nel foglio permettono di scorgere. Si ricordi infine che le regioni non sono altro che insiemi di rettangoli di delimitazione.

AndRegionRegion

Sintassi di chiamata della funzione

```
success = AndRegionRegion (region1, region2)
D0                         A0       A1
```

Scopo della funzione

Questa funzione effettua l'operazione logica AND bidimensionale tra due regioni. La regione risultante, assegnata a region2, è composta da tutte le aree comuni a entrambe le regioni indicate come argomenti. Si ricordi che le regioni sono soltanto insiemi di rettangoli di delimitazione. Si vedano anche le spiegazioni relative alla funzione AndRectRegion. La regione indicata come primo argomento non viene alterata dall'operazione. La funzione restituisce il valore 1 se l'operazione ha avuto successo, 0 se non vi era abbastanza memoria disponibile.

Argomenti della funzione

region1 — Indirizzo della prima struttura Region le cui sotto-regioni comuni anche alla regione indicata come secondo argomento entreranno a far parte della nuova regione creata dalla funzione.

region2 — Indirizzo della seconda struttura Region le cui sotto-regioni comuni anche alla regione indicata come primo argomento entreranno a far parte della nuova regione creata dalla funzione.

Discussione

Ci sono quattro funzioni che riguardano esclusivamente le regioni: AndRegionRegion, ClearRegion, OrRegionRegion e XorRegionRegion. Si vedano anche le spiegazioni relative a queste funzioni.

Lo scopo della funzione AndRegionRegion è molto simile a quello della funzione AndRectRegion, con la differenza però che la nuova regione che si

viene a creare è costituita dalle aree comuni a due regioni, e non a una regione e a un rettangolo di delimitazione. Due regioni formano una combinazione più versatile perché una regione non deve necessariamente essere un semplice rettangolo, e neanche essere continua; essa può assumere qualsiasi conformazione compresa in un perimetro di linee rette parallele, e può essere composta da forme disgiunte (ottenute, per esempio, effettuando l'unione, or, di regioni che non hanno parti sovrapposte). Si legga la spiegazione della funzione AndRectRegion per capire come e perché le regioni si combinano con i rettangoli di delimitazione; le idee e i concetti lì esposti si applicano allo stesso modo a una combinazione di due regioni.

AreaCircle

Sintassi di chiamata della macro

```
error = AreaCircle (rastPort, centerX, centerY, radius)
D0                 A1        D0      D1       D2
```

Scopo della macro

Questa macro-istruzione aggiunge un cerchio alle informazioni per il riempimento di aree relative a una bitmap. Il cerchio, sotto forma di vettori, viene aggiunto nel buffer di vettori mantenuto dalla struttura AreaInfo legata alla struttura RastPort che definisce la bitmap. Quando viene avviato il riempimento, il cerchio e gli altri perimetri eventualmente memorizzati vengono riempiti con colore pieno oppure con una matrice grafica, in accordo con le impostazioni di controllo presenti nella struttura RastPort. AreaCircle svolge il suo lavoro chiamando la funzione AreaEllipse; se la chiamata ha successo restituisce il valore 0 e restituisce invece -1 se non c'è più spazio nel buffer di vettori definito dalla struttura AreaInfo (inizializzata tramite la funzione InitArea).

Argomenti della macro

rastPort — Indirizzo della RastPort sotto il cui controllo si svolge l'operazione.

centerX — Coordinata x del centro del cerchio misurata nel sistema di coordinate della bitmap.

centerY Coordinata y del centro del cerchio misurata nel sistema di coordinate della bitmap.

radius Raggio del cerchio (espresso in pixel) misurato rispetto alla risoluzione orizzontale del sistema di coordinate della bitmap.

Discussione

Ci sono due macro-istruzioni che riguardano i cerchi: AreaCircle e DrawCircle. AreaCircle chiama la funzione AreaEllipse mentre DrawCircle chiama la funzione DrawEllipse.

Ci sono due maniere per inserire un cerchio in una bitmap. Primo, si può chiamare la macro DrawCircle per disegnare immediatamente il cerchio nella bitmap. Secondo, si può chiamare la macro AreaCircle per disporre le informazioni riguardanti il cerchio nella tavola, o buffer, dei vettori della struttura AreaInfo; quest'operazione non causa però l'apparizione del cerchio.

Se si procede nel secondo modo, gli argomenti indicati nella chiamata della macro permettono alla funzione AreaEllipse di costruire una rappresentazione vettoriale del cerchio e di memorizzarla nella tavola dei vettori della struttura AreaInfo. Il cerchio apparirà nella bitmap solo quando verrà chiamata la funzione AreaEnd. Nello stesso istante verranno disegnati anche gli altri cerchi, ellissi e rettangoli rappresentati dalla tavola dei vettori.

L'azione della coppia di istruzioni AreaCircle e AreaEnd è molto simile a quella del trio di funzioni AreaDraw, AreaMove e AreaEnd. Per comprendere meglio come agisce la macro AreaCircle si leggano le spiegazioni relative a queste tre funzioni.

AreaDraw

Sintassi di chiamata della funzione

```
error = AreaDraw (rastPort,     x,     y)
D0                A1            D0     D1
```

Scopo della funzione

Questa funzione aggiunge un punto alla lista dei punti-limite che definiscono un'area di riempimento. AreaDraw restituisce il valore zero se non

si verificano errori, -1 se non è rimasto spazio nel buffer di vettori inizializzato con la funzione InitArea.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort che controlla l'operazione.
x	Coordinata orizzontale del pixel nel sistema di coordinate della bitmap.
y	Coordinata verticale del pixel nel sistema di coordinate della bitmap.

Discussione

Nella libreria Graphics ci sono sei funzioni che riguardano direttamente il trattamento delle aree nel sistema Amiga: AreaDraw, AreaEnd, AreaMove, InitArea, RectFill e Flood. Si vedano anche le spiegazioni relative a queste funzioni.

AreaDraw consente di definire i vertici di un'area limitata di schermo. Quest'area sarà in seguito riempita con una certa matrice grafica e particolari combinazioni di colori. Si può usare la funzione AreaDraw per definire tutte le aree geometriche della bitmap. Questo si ottiene definendo tutti gli insiemi di vertici richiesti per fissare gli angoli di tali aree.

Una bitmap è composta da uno o più bitplane. Dopo che si è chiamata la funzione AllocRaster per assegnare abbastanza spazio RAM a ciascuno dei bitplane richiesti per definire la bitmap, si deve riempire la bitmap con le informazioni che dovranno essere visualizzate. Per svolgere questo lavoro ci sono almeno due modi. Il primo controlla direttamente i bit dei bitplane. Il secondo utilizza le funzioni di disegno per impostare adeguatamente i bit dei bitplane.

Prima procedura di disegno

Si può determinare lo stato di ciascun bit di un bitplane. Si scelga; per prima cosa, su quale bit della bitmap si desidera agire; scegliendo un bit della bitmap, in pratica si scelgono tutti i bit che hanno quella posizione nei bitplane che costituiscono la bitmap. Si impostino poi i valori di FgPen, BgPen e AOlpen attraverso chiamate a SetAPen, SetBpen e SetOpen per scegliere il registro colore desiderato. In seguito s'impostano gli opportuni bit dei bitplane.

FgPen, BgPen e AOlpen possono subire variazioni nella definizione di parti diverse di una bitmap. Per esempio, se si opera su di una bitmap composta da

un solo bitplane, si possono assegnare soltanto due colori a ciascun punto; un colore è associato al bit con valore 0 e l'altro al bit con valore 1. Se si adotta il modo di disegno JAM2 (si veda la spiegazione della funzione SetDrMd), i bit di valore 1 della bitmap avranno il colore FgPen e i bit di valore 0 avranno il colore BgPen.

Si possono usare le funzioni LoadRGB4 o SetRGB4 per impostare i valori contenuti nella tavola colormap (che stabilisce gli accoppiamenti tra i colori e i registri) e le funzioni SetAPen e SetBPen per assegnare un numero di penna primario e uno secondario alla struttura RastPort a cui è collegata la bitmap in questione. In questo modo si disegna direttamente in ciascun bit della bitmap che stiamo considerando. Questa procedura può essere estesa per disegnare in una bitmap composta di più bitplane, ma l'accesso agli stessi bit nei diversi bitplane necessari per creare il numero del registro colore che desideriamo indirizzare per quel punto, rende la questione più complicata.

Seconda procedura di disegno

Per rendere le cose più semplici, si possono usare le funzioni di disegno. Esse forniscono un maggior livello di controllo sulla creazione del disegno, come è necessario per definire gli aspetti più comuni delle presentazioni video. In particolare, le funzioni di disegno si possono usare per disegnare linee continue, linee tratteggiate in vario modo, rettangoli o poligoni, pieni o riempiti con matrici grafiche. A causa della loro capacità di maneggiare contemporaneamente sezioni estese di bitmap, queste funzioni rendono considerevolmente veloce la definizione della presentazione video.

La routine AreaDraw lavora con una particolare struttura RastPort che controlla il disegno all'interno della relativa bitmap. In memoria possono coesistere simultaneamente diverse bitmap e diverse strutture RastPort che controllano il disegno al loro interno. Il programma dev'essere organizzato in modo tale che il task agisca con una bitmap per volta. Si può per esempio definire tutte le aree poligonali che riempiranno la prima bitmap e poi ripetere l'operazione per la seconda. In questo modo si può ottimizzare l'uso della memoria, liberando temporaneamente aree di buffer, mentre si procede da una bitmap all'altra.

Il bitplane è sempre un'area continua di memoria allocata nella chip RAM; tuttavia, i bitplane che costituiscono una bitmap non devono essere necessariamente collocati in sezioni contigue di RAM. Ciascun bitplane può iniziare da una propria posizione in RAM, a seconda di ciò che viene fissato dalla funzione AllocRaster.

Si ricordi che i valori x,y di una bitmap sono compresi nella gamma tra 0,0 e 1024,1024 (si sta ora considerando il disegno nella bitmap, non sullo schermo, dove i valori x e y sono in genere limitati rispettivamente a 640 e 512 nel sistema PAL).

È importante notare che la funzione AreaDraw non disegna effettivamente su aree dello schermo; essa aggiunge semplicemente un nuovo punto a una lista di punti x,y nel buffer dei vettori relativo a quella bitmap (che sarà eventualmente usato per definire un insieme di aree riempite con la funzione

AreaEnd). Si tenga presente che dev'essere stato preventivamente predisposto abbastanza spazio in RAM, utilizzando la funzione InitArea, per conservarvi i vettori necessari.

Non è obbligatorio chiamare la funzione AreaEnd subito dopo la definizione di un'area poligonale attraverso una sequenza di chiamate alla funzione AreaDraw. Si possono infatti definire altre forme geometriche vettoriali. Una volta che tutte le aree poligonali da far apparire in una bitmap sono state definite, si può chiamare AreaEnd per produrre il desiderato riempimento di aree. Si può poi liberare l'area di memoria occupata dal buffer per aumentare la memoria libera.

Si procede poi con la bitmap seguente, predisponendo un buffer e utilizzando le funzioni AreaDraw, AreaMove e AreaEnd per definire i vettori e le aree poligonali di riempimento.

Attenzione a non confondere le funzioni AreaDraw e Draw. AreaDraw realizza un insieme di linee (vettori) per definire un'area che verrà riempita solo successivamente; Draw produce una serie di linee (spesso collegate) definite da coppie x,y ma senza lo scopo di riempire l'area da esse delimitata. La funzione Draw va impiegata per disegnare linee; la funzione AreaDraw serve per tracciare contorni di aree.

Bisogna inoltre tener presente la distinzione tra le funzioni AreaDraw e PolyDraw: PolyDraw riguarda, ancora una volta, le linee; AreaDraw agisce invece con le aree. PolyDraw consente di specificare una serie di coppie x,y per definire poligoni chiusi; tali poligoni non verranno tuttavia riempiti dalla funzione AreaEnd.

AreaEllipse

Sintassi di chiamata della funzione

```
error = AreaEllipse (rastPort, centerX, centerY, horiz_radius, vert_radius)
DØ                  A1         DØ:16   D1:16    D2:16         D3:16
```

Scopo della funzione

Questa funzione aggiunge la rappresentazione vettoriale di un'ellisse alle informazioni utilizzate per il riempimento di aree nella bitmap considerata. Quando l'ellisse viene poi disegnata, viene riempita tramite la funzione AreaEnd con il colore pieno o con una matrice grafica, in accordo con i parametri di controllo del disegno contenuti nella struttura RastPort. AreaEllipse restituisce il valore zero se ha avuto successo, oppure il valore –1 se non c'è più spazio nella tavola dei vettori rappresentata dalla struttura AreaInfo.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort sotto il cui controllo si svolge l'operazione.
centerX	Coordinata x del centro dell'ellisse, misurata con il sistema di coordinate della bitmap.
centerY	Coordinata y del centro dell'ellisse, misurata con il sistema di coordinate della bitmap.
horiz_radius	Semiasse orizzontale dell'ellisse espresso in pixel.
vert_radius	Semiasse verticale dell'ellisse espresso in pixel.

Discussione

Ci sono due funzioni che riguardano le ellissi: DrawEllipse e AreaEllipse, che rappresentano due modi completamente diversi di disegnare un'ellisse nella bitmap. Il primo consiste nell'usare la funzione DrawEllipse per disegnare l'ellisse direttamente. Il secondo consiste nel chiamare la funzione AreaEllipse per predisporre la rappresentazione vettoriale dell'ellisse nella tavola di vettori associata con la struttura AreaInfo, rappresentazione che va così ad aggiungersi alle altre nel riempimento di aree. In questo secondo caso, l'ellisse verrà inserita nella bitmap soltanto quando verrà chiamata la funzione AreaEnd. L'azione della coppia di funzioni AreaEllipse e AreaEnd è molto simile all'azione del trio di funzioni AreaDraw, AreaMove e AreaEnd. Si vedano anche le spiegazioni relative a queste tre funzioni.

AreaEnd

Sintassi di chiamata della funzione

AreaEnd (rastPort)
A1

Scopo della funzione

Questa funzione elabora una tavola di vettori x,y e produce il riempimento di aree attraverso l'apposita sequenza indicata nella tavola. Essa elabora il buffer di vettori realizzato tramite le funzioni AreaDraw e AreaMove. Dopo che il riempimento è stato completato, questa funzione reimposta il punto finale della tavola di vettori preparandolo per la successiva chiamata ad AreaMove. La funzione AreaEnd utilizza un buffer temporaneo di lavoro impostato da InitTmpRas quando si genera una maschera di riempimento di aree.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort sotto il cui controllo si svolge l'operazione.

Discussione

Ci sono sei funzioni di disegno della libreria Graphics che riguardano specificamente la creazione di aree nel sistema Amiga: AreaDraw, AreaEnd, AreaMove, InitArea, RectFill e Flood. Queste funzioni consentono di definire aree di bitmap (parti iniziali di una struttura ViewPort) che si possono impiegare per realizzare una presentazione video complessa. Si vedano anche le spiegazioni relative a queste funzioni.

La procedura di riempimento

Ipotizziamo che l'obiettivo sia la definizione di aree poligonali da riempire utilizzando per ognuna informazioni diverse. Nel momento della creazione della singola area si alterano i valori delle penne e delle matrici grafiche che le definiscono, in modo che, quando si chiama la funzione AreaEnd per avviare il riempimento, ogni area appare con i colori e con la matrice grafica stabilita. Il task deve contenere una serie di chiamate alle funzioni SetAPen, SetBPen, SetDrMd e alla macro SetAfPt, seguite immediatamente da una o più chiamate alle funzioni AreaDraw e AreaMove. Tutte le informazioni di controllo del disegno vengono conservate nella struttura RastPort; tutti i vettori x,y nella struttura AreaInfo. Le informazioni contenute in quest'ultima struttura cambiano via via che si aggiungono nuovi poligoni e vettori alla lista delle aree da riempire.

La chiamata della funzione AreaEnd dev'essere rimandata finché non sono stati completamente impostati tutti i vettori relativi a ciascuna area (utilizzando le funzioni AreaDraw e AreaMove). Ciascuna chiamata di AreaEnd riguarda tutto il complesso di definizioni inserite nella struttura AreaInfo e le

informazioni di controllo conservate nella struttura RastPort.

Si può anche usare la funzione InitTmpRas per impostare un'area di lavoro separata in memoria, al fine di mantenere le informazioni di disegno temporanee necessarie alla funzione AreaEnd. Se le informazioni per il riempimento di area sono estese, questo buffer temporaneo può rivelarsi indispensabile. Per impostarlo basta chiamare la funzione InitTmpRas indicando la grandezza richiesta per il buffer; in seguito, la struttura RastPort può usarlo per sviluppare valori intermedi da utilizzare con le definizioni del disegno.

Si noti infine che la funzione AreaEnd disegna anche linee di contorno per l'area che riempie. Per la matrice grafica di queste linee viene usata la definizione della matrice di continuità indicata nella struttura RastPort. Se non si vogliono aggiungere linee di contorno, basta rendere conforme la matrice di continuità alla matrice grafica e al colore di fondo del punto in cui le linee vengono disegnate. In tal modo i contorni vengono disegnati ugualmente, ma si identificano con lo sfondo e quindi non sono visibili.

AreaMove

Sintassi di chiamata della funzione

```
error = AreaMove (rastPort,     x,     y)
D0               A1             D0     D1
```

Scopo della funzione

Questa funzione definisce il punto di partenza per una nuova sagoma nel buffer di vettori di riempimento. Essa chiude automaticamente l'ultima area creata con la funzione AreaDraw e inizia una nuova area nella posizione x,y indicata dal proprio argomento. AreaMove restituisce il valore 0 se non si sono verificati errori, oppure il valore –1 se non è rimasto spazio nel buffer di vettori assegnato con la funzione InitArea.

Argomenti della funzione

rastPort Indirizzo della struttura RastPort sotto il cui controllo si svolge l'operazione.

x	Coordinata orizzontale del pixel nel sistema di coordinate della bitmap.
y	Coordinata verticale del pixel nel sistema di coordinate della bitmap.

Discussione

Nella libreria Graphics ci sono sei funzioni che riguardano specificamente la creazione di aree nel sistema Amiga: AreaDraw, AreaEnd, AreaMove, InitArea, RectFill e Flood. Queste funzioni consentono di definire aree di bitmap (parti iniziali di una struttura ViewPort) che si possono impiegare per realizzare una presentazione video complessa. Si vedano anche le spiegazioni relative a queste funzioni.

AreaMove definisce il punto di partenza per una nuova sagoma; i punti seguenti vengono definiti attraverso la funzione AreaDraw.

Le funzioni AreaDraw, AreaMove e AreaEnd forniscono nel loro insieme un modo per definire aree piene in una determinata bitmap. Esse forniscono un metodo di alto livello per inserire informazioni su colori, matrici di linea e di area nella bitmap. Se si dovesse procedere operando su ogni singolo bit il lavoro risulterebbe molto più lungo. La funzione AreaMove dà la possibilità d'iniziare una nuova definizione di area per un successivo riempimento. Per la precisione, AreaMove consente di definire il punto iniziale (una coppia di valori x,y) di una nuova area da riempire in seguito con un'opportuna matrice grafica. Questa matrice dev'essere impostata tramite una chiamata alla macro SetAfPt nella struttura RastPort associata alla bitmap sulla quale si sta disegnando.

La struttura RastPort definisce anche una matrice di linea, detta matrice di continuità. La matrice di linea viene impostata tramite una chiamata alla macro SetDrPt. Infine la struttura RastPort possiede nei parametri FgPen, BgPen e AOlPen i numeri di penna primario, secondario e di contorno impostati dalle ultime chiamate rispettivamente a SetAPen, SetBPen e SetOPen.

Vi sono altri due parametri nella struttura RastPort: cp_x e cp_y. Rappresentano la posizione della penna nel corso dei suoi movimenti all'interno della bitmap. Ogni volta che si effettua una chiamata alle funzioni AreaMove e AreaDraw, i valori di questi parametri di posizione vengono alterati.

Se la bitmap è relativamente piccola, bisogna stare attenti che i valori x,y specificati nella funzione AreaMove non superino i limiti precedentemente impostati con le chiamate alle funzioni AllocRaster o InitBitMap. Se s'imposta una coppia di valori x,y esterna all'intervallo consentito, può accadere che una chiamata alle funzioni AreaDraw o AreaMove mandi in crash il sistema.

Bisogna anche tenere d'occhio il parametro di mascheramento per la scrittura presente nella struttura RastPort. Questo parametro determina quali bitplane sono abilitati per la scrittura. In genere tutti i bitplane di una bitmap consentono operazioni di scrittura. In tal caso il valore del parametro di abilitazione è impostato con una serie completa di 1.

Se si vuole evitare che la funzione AreaEnd interessi alcuni bitplane di una

bitmap, bisogna usare il parametro di mascheramento per la scrittura presente nella struttura RastPort; basta impostare tale parametro a un valore binario per impedire la scrittura nei bitplane che non devono essere coinvolti nell'operazione.

Si badi bene a non confondere le funzioni AreaMove e Move. AreaMove definisce sulla bitmap il punto d'inizio di una nuova area da riempire; Move sposta la penna per iniziare a disegnare una linea sulla bitmap o per inserire caratteri di testo.

AttemptLockLayerRom

Sintassi di chiamata della funzione

```
success = AttemptLockLayerRom (layer)
DØ                             A5
```

Scopo della funzione

Questa funzione tenta di bloccare la struttura Layer puntata dall'argomento layer. Se la chiamata ad AttemptLockLayerRom ha successo, nessun altro task potrà alterare quella struttura Layer fino a quando non verrà usata una funzione di sblocco (UnLockLayerRom, UnLockLayer o UnLockLayers). AttemptLockLayerRom non mette mai in attesa il task chiamante, ma restituisce il valore TRUE (vero) se ha avuto successo, oppure il valore FALSE (falso) in caso contrario.

Argomenti della funzione

layer	Indirizzo della struttura Layer da bloccare.

Discussione

La funzione AttemptLockLayerRom è molto simile alla funzione LockLayerRom, con un'eccezione: AttemptLockLayerRom restituisce un valore tramite il quale il programma può determinare se il layer è stato bloccato. Se viene restituito TRUE, significa che il layer specificato è stato opportunamente bloccato. Se viene restituito FALSE significa che il layer è già bloccato e la

funzione non può quindi bloccarlo a sua volta. In tal caso, tuttavia, il task chiamante legge il valore di ritorno (FALSE) e può proseguire l'esecuzione senza entrare in attesa. Questo non è possibile con la funzione LockLayerRom, la quale, se il layer risulta bloccato, manda in attesa il task fino a quando il layer non viene sbloccato.

BltBitMap

Sintassi di chiamata della funzione

```
planes = BltBitMap (srcBitMap, srcX, srcY, destBitMap, destX,
DØ                  AØ         DØ   D1   A1          D2
                       destY, sizeX, sizeY, minTerm, mask, tempA)
                       D3     D4     D5     D6       D7    A2
```

Scopo della funzione

Questa funzione svolge un'operazione di copia servendosi del Blitter. L'operazione consiste nello spostare un rettangolo da un punto all'altro di una bitmap o da una bitmap all'altra. La funzione BltBitMap restituisce il numero dei bitplane coinvolti nell'operazione di copia. Non esegue alcun controllo e non restituisce nessuna condizione di errore (per esempio nel caso che il rettangolo specificato non sia interamente contenuto nella bitmap). Il numero dei bitplane restituito dalla funzione BltBitMap sarà inferiore a quello atteso se la memoria chip che il task ha allocato e indicato nel parametro tempA non risulta sufficiente.

Argomenti della funzione

srcBitMap — Indirizzo della struttura BitMap che definisce la bitmap di partenza.

srcX,srcY — Coordinate x e y, espresse in pixel nel sistema di coordinate della bitmap; si riferiscono all'angolo superiore sinistro del rettangolo di partenza che si trova nella bitmap indicata nell'argomento srcBitMap.

destBitMap Indirizzo della struttura BitMap che definisce la bitmap destinazione. Può anche essere lo stesso indirizzo indicato dal task nel parametro srcBitMap.

destX,destY Coordinate x e y, espresse in pixel nel sistema di coordinate della bitmap; si riferiscono all'angolo superiore sinistro del rettangolo destinazione che si trova nella bitmap indicata nell'argomento destBitMap.

sizeX,sizeY Misura in pixel del rettangolo che dev'essere copiato.

minTerm Byte senza segno (unsigned byte) i cui bit indicano la funzione logica da applicare ai rettangoli di partenza e di destinazione per produrre un nuovo rettangolo finale.

mask Maschera di scrittura da applicare per l'operazione con il Blitter. Indica su quali bitplane la funzione deve agire.

tempA Indirizzo dell'area buffer temporanea della memoria chip dove può essere memorizzata una riga di dati sorgenti (fino a 1024 pixel); questo buffer viene usato quando i rettangoli di partenza e di destinazione si sovrappongono. Si noti che il task può non indicare alcun buffer: se durante l'elaborazione se ne presenta la necessità, la funzione provvede ad allocarlo autonomamente.

Discussione

Nella libreria Graphics ci sono dieci funzioni che interagiscono con il Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit.

La funzione BltBitMap impiega il Blitter per muovere una porzione rettangolare di bitmap verso un'altra area della stessa misura che può trovarsi nella stessa bitmap o in un'altra. Durante l'operazione di copia, i dati possono anche essere alterati. Il Blitter è il mezzo ideale per raggiungere questo obiettivo perché può trasferire fino a quattro megabyte al secondo, oltre a essere in grado di disegnare linee alla rapidità di un milione di pixel al secondo.

La funzione BltBitMap lavora con una bitmap origine e una bitmap destinazione. La posizione del rettangolo dev'essere specificata in entrambe le bitmap. Se i trasferimenti di dati vengono effettuati nel periodo che passa tra il completamento di un quadro video e l'inizio del successivo, si può produrre un effetto simile alle animazioni ottenibili con i bob.

Vengono copiati dall'origine alla destinazione solo i bit che hanno numeri di bitplane identici e una maschera di scrittura diversa da zero (si veda la definizione della struttura RastPort nell'introduzione a questo capitolo). Inoltre possono essere copiati solo quei bitplane i cui numeri di bitplane sono inferiori al numero massimo di bitplane ammessi.

Le variabili srcX, srcY, destX e destY sono limitate a valori compresi tra 0 e 32767-size. Ci si deve perciò assicurare che questi due insiemi di valori vengano scelti in modo che le operazioni di copia non superino i confini della bitmap destinazione.

L'intervallo di variazione consentito all'argomento sizeX va da 1 a 976; quello dell'argomento sizeY va da 1 a 1023. Entrambi gli argomenti devono essere scelti in modo che le operazioni di copia non oltrepassino i confini della bitmap destinazione.

L'argomento minTerm controlla il modo in cui agisce la logica di trasferimento dati durante l'azione del Blitter. Si può scrivere un'equazione logica per definire ciascun tipo di operazione di trasferimento logico tra una bitmap origine. A titolo di esempio si noti che il valore $0C0 richiede la semplice copia, il valore $030 richiede che venga invertita l'area sorgente prima di procedere alla copia, il valore $050 richiede d'ignorare l'origine e d'invertire la destinazione. Si veda anche il *Manuale dell'Hardware dell'Amiga* (pubblicato dalla IHT Gruppo Editoriale), per maggiori informazioni sui valori che si possono indicare in questo argomento.

I bit impostati nella variabile di mascheramento indicano quali bitplane partecipano all'operazione. Normalmente la variabile di mascheramento è impostata a 0xFF (11111111) per indicare che sono interessati tutti i bitplane.

L'argomento tempA è impiegato per consentire copie quando origine e destinazione si sovrappongono. Se la copia si sovrappone esattamente a sinistra o a destra su un confine di indirizzi (il che significa che gli indirizzi sorgente e destinazione si ritrovano entrambi in una word di confine) e l'argomento tempA è diverso da zero, tempA deve contenere l'indirizzo di un buffer nella chip RAM sufficiente per contenere una linea della bitmap origine durante le operazioni del Blitter. Questa memoria dev'essere stata allocata in precedenza impiegando una delle funzioni della libreria Exec previste per questo scopo.

BltBitMapRastPort

Sintassi di chiamata della funzione

```
BltBitMapRastPort (sourceBitMap, sourceX, sourceY, destRastPort,
                   A0            D0       D1       A1
                   destX, destY, sizeX, sizeY, minTerm)
                   D2     D3     D4     D5     D6
```

Scopo della funzione

Questa funzione, tramite il Blitter, svolge un'operazione di copia dalla bitmap origine verso la bitmap destinazione. La bitmap destinazione viene definita indirettamente attraverso l'indirizzo della relativa struttura RastPort. I dettagli dell'operazione compiuta dal Blitter sono completamente controllati dall'argomento minTerm che specifica come devono essere combinati i pixel origine e destinazione.

Argomenti della funzione

sourceBitMap Indirizzo della struttura BitMap che definisce la bitmap origine.

sourceX Coordinata x del punto iniziale (in alto a sinistra) nella bitmap origine.

sourceY Coordinata y del punto iniziale nella bitmap origine.

destRastPort Indirizzo della struttura RastPort che contiene l'appropriato puntatore alla struttura BitMap destinazione.

destX Coordinata x del punto iniziale (in alto a sinistra) nella bitmap destinazione.

destY Coordinata y del punto iniziale nella bitmap destinazione.

sizeX Numero di pixel (in direzione x) su cui effettuare l'operazione con il Blitter.

sizeY Numero di pixel (in direzione y) su cui effettuare l'operazione con il Blitter.

minTerm Indica al Blitter l'operazione che deve effettuare.

Discussione

Ci sono due funzioni che riguardano le operazioni effettuabili con il Blitter da una bitmap origine a una bitmap destinazione: BltBitMapRastPort e BltMaskBitMapRastPort. Entrambe le funzioni impiegano una combinazione logica per impostare l'operazione del Blitter; in particolare, BltMaskBitMapRa-

stPort usa una maschera. Per comprendere meglio come e perché vengono svolte queste operazioni, si consultino le spiegazioni relative alle altre funzioni che utilizzano il Blitter (BltBitMap, BltClear, BltPattern e BltTemplate).

BltClear

Sintassi di chiamata della funzione

BltClear (memBlock, bytecount, flags)
A1 D0 D1

Scopo della funzione

Questa funzione impiega il Blitter per cancellare un blocco di memoria nella chip RAM. Nel modo righe/byte-per-riga, la variabile riga dev'essere inferiore o uguale a 1024 e la variabile byte-per-riga dev'essere inferiore o uguale a 128. Indicando in bytecount il modo standard, la funzione può anche impiegare il Blitter più volte per azzerare tutta la memoria indicata.

Argomenti della funzione

memBlock Indirizzo del blocco di memoria da azzerare. L'argomento memBlock dev'essere un indirizzo pari, perché le operazioni di azzeramento inizino allineate alle word.

bytecount Se il secondo bit (il bit 1) dell'argomento flag non è impostato a 1, rappresenta un numero pari di byte da cancellare (modo standard bytecount); altrimenti i 16 bit inferiori dell'argomento bytecount indicano il numero dei byte per riga, e i 16 bit superiori il numero delle righe (modo riga/byte-per-riga).

flags Questo è un argomento da un byte; il bit 0 va impostato a 1 per forzare la funzione BltClear ad attendere finché le operazioni del Blitter non sono terminate.

Discussione

Nella libreria Graphics ci sono dieci funzioni che riguardano specificamente le operazioni effettuabili con il Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Queste funzioni consentono di sfruttare le velocissime operazioni DMA del Blitter per svolgere le seguenti operazioni:

- azzerare la memoria impostando i byte a zero (BltClear).
- Svolgere operazioni di riempimento di area (BltPattern).
- Trasferire blocchi di memoria (BltBitMap, BltTemplate e ClipBlit).
- Ottenere l'uso esclusivo del Blitter (OwnBlitter).
- Restituire il Blitter agli altri task (DisownBlitter).
- Accodare una richiesta per l'uso del Blitter (QBlit).
- Sincronizzare una richiesta inviata dal task al Blitter con la posizione del pennello elettronico durante la scansione video dello schermo (QBSBlit).
- Entrare in attesa che il Blitter finisca prima di procedere con altre operazioni (WaitBlit).

BltClear è uno strumento per cancellare blocchi di RAM ad alta velocità. Si ricordi che BltClear agisce soltanto sulla memoria chip, costituita dal primo megabyte di RAM e utilizzata dagli speciali chip dedicati dell'Amiga: Agnus (il chip dedicato all'animazione), Denise (il chip dedicato alla grafica) e Paula (il chip dedicato alle periferiche e al suono). Tali chip, insieme con altri, possono leggere e scrivere soltanto nella chip RAM. Quindi, tutte le bitmap devono essere confinate in quest'area di memoria.

La funzione BltClear richiede tre argomenti. Il primo, memBlock, è l'indirizzo che individua l'inizio del blocco di memoria in cui deve aver luogo l'operazione di cancellazione. Nella maggior parte dei casi, questa variabile dev'essere un numero pari.

Il secondo argomento è bytecount, nel quale si deve indicare il numero di byte da cancellare. Può trattarsi di un numero pari oppure dispari, a seconda degli altri argomenti della funzione.

Il terzo argomento è flags, che contiene diversi valori in grado d'influenzare il modo d'esecuzione della funzione. Se il bit 0 viene impostato a 1, la funzione BltClear non restituisce il controllo fino a quando l'operazione del Blitter non termina. Ciò significa che il task entra in attesa fino a quando l'operazione non si conclude. Si noti che questo è un comportamento simile a quello della funzione WaitBlit, che sospende le operazioni del task fino a quando una serie

di richieste accodate per il Blitter non sono state soddisfatte.

Se il bit 1 dell'argomento flags viene impostato a 0, la funzione BltClear ne deduce che bytecount è il numero (pari) di byte da cancellare e agisce di conseguenza.

Se il bit 1 viene invece impostato a 1, la funzione BltClear opera nel modo riga/byte-per-riga; i 16 bit inferiori della variabile bytecount indicano il numero dei byte per riga e i 16 bit superiori il numero delle righe da cancellare. Questo modo viene usato molto spesso per "pulire" una bitmap, in particolare per cancellare un blocco di memoria assegnato a un particolare bitplane. Si può utilizzare il puntatore restituito dalla funzione AllocRaster (quando si è allocato il bitplane) per ricavarne un puntatore al blocco di memoria in questione. La funzione BltClear va poi chiamata con questo puntatore.

Si noti che se si usa il modo riga/byte-per-riga si deve sottostare alle limitazioni di 1024 righe e 128 byte per riga. Se il blocco totale che si vuole cancellare è più esteso, si può chiamare più volte la funzione BltClear.

BltMaskBitMapRastPort

Sintassi di chiamata della funzione

```
BltMaskBitMapRastPort (sourceBitMap, sourceX, sourceY, destRastPort,
                       A0            D0      D1      A1
                       destX, destY, sizeX, sizeY, minTerm, bltMask)
                       D2     D3     D4    D5     D6       A2
```

Scopo della funzione

Questa funzione svolge un'operazione di Blitter da una bitmap origine verso una bitmap destinazione, indicata attraverso l'indirizzo della relativa struttura RastPort. I dettagli dell'operazione di Blitter sono controllati in parte dal valore logico minTerm, che specifica come devono essere combinati i pixel origine e destinazione, e in parte da una maschera che definisce i pixel interessati all'operazione seguendo le regole logiche indicate da minTerm. In altre parole, la maschera determina quali pixel vengono alterati dal Blitter nel corso dell'operazione.

Argomenti della funzione

sourceBitMap Indirizzo alla struttura BitMap che rappresenta la bitmap origine.

sourceX Coordinata x del punto iniziale (in alto a sinistra) nella bitmap origine.

sourceY Coordinata y del punto iniziale nella bitmap origine.

destRastPort Indirizzo della struttura RastPort che contiene il puntatore alla struttura BitMap destinazione.

destX Coordinata x del punto iniziale (in alto a sinistra) nella bitmap destinazione.

destY Coordinata y del punto iniziale nella bitmap destinazione.

sizeX Numero di pixel in orizzontale su cui effettuare l'operazione di Blitter; la misura viene calcolata nel sistema di coordinate della bitmap origine.

sizeY Numero di pixel in verticale su cui effettuare l'operazione di Blitter.

minTerm Istruzioni logiche da impiegare per l'operazione di Blitter; si può usare, ad esempio, (ABC|ABNC|ANBC) se si desidera copiare l'origine ed effettuare su di essa l'operazione di blitter attraverso la maschera, oppure (ANBC) se si desidera prima invertire l'origine ed effettuare in seguito su di essa l'operazione di blitter attraverso la maschera.

bltMask Puntatore in RAM a una maschera costituita da un bitplane; il bitplane della maschera deve avere le stesse dimensioni dei bitplane che costituiscono la bitmap origine.

Discussione

Ci sono due funzioni che riguardano le operazioni di Blitter: BltBitMapRastPort e BltMaskBitMapRastPort. Entrambe svolgono un'operazione di Blitter partendo da una bitmap origine e agendo verso una bitmap destinazione con l'impiego di una combinazione logica.

BltMaskBitMapRastPort impiega inoltre una maschera per controllare tale operazione. Per comprendere come e perché vengono svolte le operazioni di Blitter, si leggano le spiegazioni di queste due funzioni insieme a quelle delle altre funzioni relative al Blitter (BltBitMap, BltClear, BltPattern e BltTemplate).

BltPattern

Sintassi di chiamata della funzione

```
BltPattern (rastPort, maskBitMap, x1, y1, maxx, maxy, bytecount)
            A1        A0          D0  D1  D2   D3    D4
```

Scopo della funzione

Questa funzione svolge un'operazione di Blitter usando il modo di disegno, la matrice grafica di riempimento, la linea di contorno e la maschera di scrittura della struttura RastPort. L'area rettangolare è definita dalle coordinate x1,y1 e maxx,maxy. La bitmap immagine deve iniziare a un indirizzo pari.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort sotto il cui controllo si svolge l'operazione.
maskBitMap	Indirizzo alla bitmap bidimensionale che funge da maschera.
x1,y1	Coordinate dell'angolo superiore sinistro di una regione rettangolare della bitmap.
maxx,maxy	Coordinate dell'angolo inferiore destro di una regione rettangolare della bitmap.
bytecount	Numero dei byte per riga usando il modo byte-per-riga.

Discussione

Nella libreria Graphics ci sono dieci funzioni che riguardano specificamente la gestione delle operazioni di Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Queste funzioni consentono di sfruttare le velocissime operazioni DMA del Blitter. Si vedano anche le spiegazioni relative a queste funzioni.

BltPattern è la sola funzione della libreria Graphics che riguarda direttamente l'uso del Blitter per la creazione dei disegni. In particolare BltPattern impiega una matrice di disegno chiamata *maschera*. Questa funzione si utilizza quando si devono variare i bit di un blocco di memoria, e si vuole che soltanto determinati bit rimangano impostati a 1 dopo che BltPattern ha svolto il suo lavoro. In effetti, l'idea è quella di porre una matrice o maschera su un blocco di memoria e poi azzerare i bit che nella maschera sono impostati a zero.

S'immagini, per esempio, di avere una sezione di memoria a quattro word (ossia otto byte) e di porvi sopra una maschera. La maschera possiede bit a 1 soltanto in determinate posizioni. Per esempio:

```
0000000000000000
000000 1111 000000
000000 1111 000000
0000000000000000
```

Come si può vedere, la maggior parte dei bit di questa maschera è impostata a 0; soltanto otto bit nel centro sono impostati a 1.

Si consideri un'area di memoria con i bit completamente impostati a 1:

```
1111111111111111
1111111111111111
1111111111111111
1111111111111111
```

Ora useremo la bitmap di maschera come una matrice sovrapposta alla seconda bitmap. Il compito viene eseguito dalla funzione BltPattern. Il risultato è il seguente:

```
0000000000000000
000000 1111 000000
000000 1111 000000
0000000000000000
```

Quando s'impiega la funzione BltPattern, si deve per prima cosa stabilire quanto è grande il blocco di memoria, ossia quanta memoria la matrice debba ricoprire. In genere, si avrà a che fare con un bitplane. I limiti vanno impostati con le variabili x1, y1, maxx e maxy. Le coordinate saranno espresse nel sistema di coordinate della bitmap. Si tenga presente che i valori massimi

sono pari a 1024,1024. Bytecount indica il numero di byte per riga della maschera.

Subito dopo si stabilisce per la bitmap di maschera un blocco di memoria che abbia la stessa misura dell'area di memoria da mascherare. Il secondo argomento (maskBitMap) serve per puntare a tale maschera. I bit delle word nella bitmap di maschera vanno impostati in accordo con la maschera specifica che si intende predisporre. Va poi precisata la variabile di puntamento RastPort. Essa sarà un puntatore alla struttura RastPort che controlla il disegno in questa particolare bitmap. Si noti che la funzione BltPattern disegna nella bitmap seguendo le regole precedentemente impostate nella definizione della struttura RastPort.

Prima di chiamare la funzione BltPattern, ci si assicuri che i parametri di disegno nella RastPort associata corrispondano alle proprie intenzioni. Si predispongano opportunamente il modo di disegno, la matrice grafica, i parametri, FgPen e BgPen. Il modo JAM1 userà FgPen per i bit a 1, mentre il modo JAM2 userà FgPen per i bit a 1 e BgPen per quelli a 0. Si vedano anche le spiegazioni relative alla funzione SetDrMd.

BltTemplate

Sintassi di chiamata della funzione

BltTemplate (source, srcX, srcModulo, destRastPort, destX, destY, sizeX, sizeY)
A0 D0 D1 A1 D2 D3 D4 D5

Scopo della funzione

Questa funzione usa il Blitter per estrarre un'area rettangolare da un array origine e porla in un'area destinazione di una bitmap. Durante il trasferimento vengono utilizzati i parametri di disegno indicati da FgPen, BgPen e dal modo di disegno.

Argomenti della funzione

source Indirizzo dell'array di dati; una sotto-sezione di questo array contiene l'area rettangolare di bitmap che dev'essere trasferita.

srcX Coordinata x dell'angolo superiore sinistro del rettangolo originale nel sistema di coordinate dell'array.

srcModulo Modulo sorgente; questo valore è impiegato per trovare la successiva riga dei bit di dati del rettangolo originale nell'array.

destRastPort Indirizzo della struttura RastPort che contiene un puntatore alle informazioni della bitmap destinazione.

destX, destY Coordinate x e y del rettangolo finale nel sistema di coordinate della bitmap destinazione.

sizeX, sizeY Misure x e y del rettangolo da spostare; questi valori sono misurati nel sistema di coordinate della bitmap sorgente.

Discussione

Nella libreria Graphics ci sono dieci funzioni che riguardano specificamente la gestione delle operazioni di Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit.

La funzione BltTemplate viene usata per trasferire un'area rettangolare da un array verso una bitmap. BltTemplate differisce in parte da BltBitMap perché la copia viene eseguita senza alterazioni, cioè senza effettuare alcuna operazione logica durante il trasferimento dei dati.

BltTemplate differisce inoltre da BltBitMap per il modo in cui vengono indicati i dati origine e destinazione. Il rettangolo origine impiegato dalla funzione BltTemplate viene puntato direttamente; fa cioè parte dell'array di bit presente in RAM.

Un'altra differenza tra le due funzioni è che BltTemplate punta a una struttura RastPort per specificare la bitmap destinazione. Perciò la bitmap destinazione viene determinata indirettamente attraverso il puntatore alla struttura BitMap contenuto nella struttura RastPort. La struttura BitMap contiene a sua volta un puntatore a una specifica bitmap e la relativa dimensione.

La variabile srcModulo dice alla funzione BltTemplate dove trovare ogni riga dei dati di origine nell'array. L'argomento srcModulo rappresenta il numero di byte di dati da aggiungere all'inizio di una riga di dati del rettangolo origine per puntare alla riga successiva. Ciò dice alla funzione BltTemplate com'è organizzato l'array dati completo e consente di estrarre correttamente ciascuna linea del rettangolo origine. Il valore di srcModulo dev'essere un numero pari.

Si presume che origine e destinazione non si sovrappongano, neanche nel caso che facciano parte della stessa bitmap. Se parte del trasferimento dati

ricade al di fuori dei confini della bitmap, esso viene limitato alla bitmap destinazione.

La variabile srcX viene misurata nell'array origine. L'angolo superiore sinistro dell'array ha le coordinate 0,0. Per effettuare copie ripetute di diverse sotto-sezioni dell'array si possono variare gli argomenti srcX, sizeX e sizeY.

Le coordinate destX e destY sono misurate nel sistema di coordinate della bitmap destinazione, con l'angolo superiore sinistro della bitmap posto sempre nella posizione 0,0.

Si può usare la funzione BltTemplate per trasferire caratteri da un'area di memoria alla loro posizione finale di presentazione, all'interno di una definizione di bitmap più estesa che essere parte di uno schermo video.

BNDRYOFF

Sintassi di chiamata della macro

BNDRYOFF (rastPort)
A1

Scopo della macro

Questa macro disattiva le linee di contorno generate dalle funzioni AreaEnd e RectFill. Azzera inoltre il flag AREAOUTLINE nella struttura RastPort.

Argomenti della macro

rastPort — Indirizzo della struttura RastPort che controlla lo svolgimento dell'operazione.

Discussione

Qualsiasi funzione che usi AOlPen produce normalmente una linea di contorno per gli oggetti che disegna; si ricordino per esempio le funzioni AreaEnd e RectFill. Si noti che l'impostazione della pennà AOlPen viene di solito determinata chiamando la funzione SetOPen. Se si vuole evitare che venga disegnato il contorno, basta chiamare la macro BNDRYOFF prima di

effettuare la chiamata alle funzioni in questione. BNDRYOFF provvede a azzerare il parametro AREAOUTLINE nella struttura RastPort, indicando che i contorni non devono essere disegnati.

Dopo che l'operazione di disegno è stata completata, si dovrebbe riportare ancora una volta la condizione di disegno dei contorni alla situazione di partenza. Però non esiste nessuna funzione o macro per raggiungere tale risultato; si deve quindi impostare il flag AREAOUTLINE nella struttura RastPort con una istruzione del tipo:

```
MyRastPort.Flags | = AREAOUTLINE;
```

CEND

Sintassi di chiamata della macro

```
CEND (uCopList)
      A1
```

Scopo della macro

Questa macro conclude una lista crescente d'istruzioni Copper definite dal programmatore. Essa chiama la routine CWait di basso livello del Copper. Quando CEND restituisce il controllo, il task dispone di una lista d'istruzioni Copper completamente definita.

Argomenti della macro

uCopList Indirizzo della struttura UCopList.

Discussione

Ci sono quattro macro che riguardano specificamente le istruzioni Copper definite dai task: CINIT, CMOVE, CWAIT e CEND. Tutte e quattro usano l'argomento di puntamento uCopList e lavorano quindi con la struttura UCopList. Si vedano anche le spiegazioni delle macro CINIT, CMOVE e CWAIT.

La macro CEND conclude una lista di istruzioni Copper preparate in precedenza in un buffer dalla macro CINIT. Ogni volta che si vuole concludere

la definizione di un insieme di istruzioni Copper definite dal programmatore, si dovrebbe chiamare la macro CEND.

CINIT

Sintassi di chiamata della macro

```
CINIT (uCopList, numInst)
       A1        DØ
```

Scopo della macro

Questa macro imposta la struttura UCopList per accettare un insieme di istruzioni Copper definite dal programmatore. Essa chiama una routine di basso livello del Copper: UCopperListInit. Quando CINIT restituisce il controllo si può iniziare a definire le istruzioni Copper utilizzando le macro CMOVE e CWAIT; si conclude poi la lista di istruzioni Copper con una chiamata alla macro CEND.

Argomenti della macro

uCopList	Puntatore a una struttura UCopList.
numInst	Numero massimo di istruzioni Copper che secondo le previsioni il buffer dovrà contenere.

Discussione

Ci sono quattro macro che riguardano specificamente le istruzioni Copper definite dal programmatore: CINIT, CMOVE, CWAIT e CEND. Tutte e quattro usano l'argomento di puntamento uCopList e lavorano quindi con la struttura UCopList. Si vedano anche le spiegazioni relative alle macro CINIT, CMOVE e CWAIT.

La macro CINIT predispone un buffer per contenere un insieme crescente di istruzioni Copper definite dal programmatore. Ciò viene ottenuto indirettamente attraverso la struttura UCopList, la quale contiene un puntatore all'area buffer che verrà utilizzata per contenere tali istruzioni. Prima di chiamare la

macro CINIT si deve valutare quante sono le istruzioni che dovranno trovar posto nel buffer in questione. Una volta che il buffer sia stato assegnato, le macro CMOVE, CWAIT e CEND vi dispongono automaticamente le loro istruzioni.

Se l'argomento di puntamento uCopList vale 0, la macro assegna memoria per la struttura UCopList e un buffer per conservare le istruzioni Copper indicate nell'argomento numInst. Se uCopList è diverso da 0, la macro CINIT reimposta la lista di istruzioni Copper per accettare nuove istruzioni; in tal caso l'argomento numInst viene ignorato.

ClearRectRegion

Sintassi di chiamata della funzione

```
success = ClearRectRegion (region, rectangle)
D0                         A0      A1
```

Scopo della funzione

Questa funzione cancella la parte di un rettangolo di delimitazione che si trova all'interno di una regione, lasciando il risultato nella regione stessa. La parte di regione che si trova all'interno del rettangolo viene eliminata. ClearRectRegion restituisce il valore TRUE se l'operazione ha avuto successo, altrimenti FALSE.

Argomenti della funzione

region	Indirizzo della struttura Region che rappresenta la regione che dev'essere combinata con il rettangolo di delimitazione.
rectangle	Indirizzo della struttura Rectangle che definisce il rettangolo di delimitazione che dev'essere sottratto dalla regione.

Discussione

ClearRegion è una delle quattro funzioni relative alle regioni. ClearRegion permette di eseguire un'operazione impossibile con le funzioni AndRegionRegion, OrRegionRegion e XorRegionRegion: la sottrazione di una rettangolo da una regione. L'effetto che si ottiene è la creazione di un buco nella regione. Qualsiasi successiva operazione di disegno non avrà effetto nella parte di bitmap che si trova all'interno del rettangolo.

Si noti che questa operazione di cancellazione è diversa dall'esecuzione di operazioni di AND, OR o XOR. Si può tuttavia giungere allo stesso risultato impiegando le funzioni regione-regione e rettangolo-regione, definendo adeguatamente i rettangoli e le regioni e impiegando le funzioni nell'ordine corretto. ClearRectRegion mette solo a disposizione un modo più semplice.

Per capire come e perché le regioni e i rettangoli di delimitazione vengono combinati si dovrebbero consultare le spiegazioni delle funzioni AndRectRegion, OrRectRegion e XorRectRegion.

ClearRegion

Sintassi di chiamata della funzione

ClearRegion (region)
A0

Scopo della funzione

Questa funzione elimina tutti i rettangoli di delimitazione che costituiscono una regione, annullando quindi la regione.

Argomenti della funzione

region	Indirizzo della struttura Region.

Discussione

Ci sono sette funzioni della libreria Graphics che riguardano specificamente le regioni e i rettangoli di delimitazione che le definiscono: AndRectRegion, ClearRectRegion, ClearRegion, DisposeRegion, NewRegion, OrRectRegion e XorRectRegion. Queste funzioni consentono di manipolare le regioni per modellarne la forma.

La funzione ClearRegion elimina completamente una regione, cioè elimina tutti i rettangoli di delimitazione a essa associati. La funzione richiede come argomento l'indirizzo della struttura Region che definisce la regione, indirizzo di cui il task entra in possesso quando chiama la funzione NewRegion.

Quando s'impiegano le routine di gestione dei rettangoli di delimitazione e delle regioni al fine di disegnare solo in alcune parti di un layer, si deve operare nel modo seguente:

1. Si usano alcune semplici istruzioni di programma per definire i vertici di tutti i rettangoli di delimitazione che s'intendono impiegare nella definizione della regione. Ogni rettangolo è definito da una struttura Rectangle.

2. Si usa la funzione NewRegion per creare una regione nulla.

3. Si usano le funzioni OrRectRegion, AndRectRegion e XorRectRegion per sfruttare i rettangoli di delimitazione nella definizione della regione. Se per esempio si volessero semplicemente sommare i rettangoli di delimitazione, basterebbe chiamare ripetutamente la funzione OrRectRegion.

4. Si salva in una variabile temporanea l'indirizzo della lista DamageList del layer considerato, e si memorizza nel parametro DamageList l'indirizzo della nostra regione.

5. Si chiama la funzione BeginUpdate. Ora si possono definire i parametri di disegno tramite le funzioni SetAPen, SetBPen e SetDrMd. Successivamente si inizia a disegnare tramite le funzioni di disegno: i disegni appariranno nel layer solo nelle parti racchiuse nella regione. I rettangoli di delimitazione che costituiscono la regione funzionano come una maschera attraverso la quale si disegna.

6. Per terminare il processo di disegno nella regione, si chiama la funzione EndUpdate. Dopo la chiamata alla funzione, si memorizza di nuovo l'indirizzo della DamageList nell'omonimo parametro della struttura Layer.

7. Si usa la funzione ClearRegion per cancellare la definizione di regione che abbiamo impiegato. Da questo momento in poi si può ridefinire la regione e ripetere le operazioni illustrate. Oppure è possibile impiegare

la funzione DisposeRegion per liberare la memoria occupata dalla regione.

ClipBlit

Sintassi di chiamata della funzione

```
ClipBlit (srcRastPort, srcX, srcY, destRastPort, destX, destY,
          AØ           DØ   D1   A1           D2     D3
          sizeX, sizeY, minTerm)
          D4     D5     D6
```

Scopo della funzione

Questa funzione svolge un'operazione di Blitter tra una bitmap origine e una bitmap destinazione.

Argomenti della funzione

srcRastPort — Indirizzo della struttura RastPort di controllo che punta alla bitmap origine.

srcX,srcY — Coordinate x e y dell'angolo superiore sinistro del rettangolo, misurate nel sistema di coordinate della bitmap origine.

destRastPort — Indirizzo della struttura RastPort di controllo che punta alla bitmap destinazione.

destX, destY — Coordinate x e y dell'angolo superiore sinistro del rettangolo finale, misurate nel sistema di coordinate della bitmap destinazione.

sizeX, sizeY — Misura in pixel del rettangolo da spostare.

minTerm — Byte che indica la funzione logica da applicare ai rettangoli origine e destinazione per produrre il nuovo rettangolo destinazione.

Discussione

Nella libreria Graphics ci sono dieci funzioni che riguardano specificamente la gestione delle operazioni di Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit.

La funzione ClipBlit è molto simile alla funzione BltBitMap, per quanto riguarda la copia d'informazioni da un punto all'altro. Tuttavia la funzione ClipBlit differisce da BltBitMap perché lavora con la struttura RastPort. Ciascuna delle due strutture RastPort che intervengono (origine e destinazione) contiene un puntatore a una struttura Layer. Ogni struttura RastPort contiene inoltre un puntatore a una struttura BitMap. Ciascuna struttura BitMap contiene infine puntatori a ognuno dei bitplane che costituiscono la bitmap. La funzione ClipBlit fornisce perciò un modo indiretto per raggiungere i bit di una bitmap.

Dal momento che lavora con le strutture RastPort, la funzione ClipBlit ha accesso al layer (o ai layer) controllati dalle strutture RastPort origine e destinazione. Essa controlla questi layer per vedere se ci sono zone di sovrapposizione. Se ci sono aree sovrapposte, ClipBlit suddivide le operazioni in un insieme di sotto-operazioni. Ciascuna di queste sotto-operazioni trasferisce dati dalla bitmap origine a un'area della bitmap destinazione. Si veda anche la spiegazione della funzione BltBitMap.

CMOVE

Sintassi di chiamata della macro

```
CMOVE (uCopList, regnum, regvalue)
       A1        D0      D1
```

Scopo della macro

Questa macro aggiunge un'istruzione Copper per spostare il valore regvalue nel registro hardware regnum. Essa chiama due funzioni di basso livello della libreria Graphics, che agiscono sul Copper: CMove per aggiungere l'istruzione MOVE del Copper, e CBump per far avanzare fino all'istruzione seguente il puntatore alle istruzioni Copper.

Argomenti della macro

uCopList	Indirizzo della struttura UCopList.
regnum	Numero del registro hardware.
regvalue	Valore a 16 bit che dev'essere scritto in tale registro.

Discussione

Ci sono quattro macro che riguardano specificamente le istruzioni Copper definite dal programmatore: CINIT, CMOVE, CWAIT e CEND. Tutte e quattro le macro usano l'argomento uCopList e lavorano quindi con la struttura UCopList. Si vedano anche le spiegazioni delle macro CINIT, CWAIT e CEND.

CMOVE e CWAIT costituiscono la controparte diretta delle chiamate alle routine MOVE e WAIT in linguaggio Assembly. Se si vuole controllare l'hardware a questo livello, si può accedere direttamente a MOVE e WAIT attraverso il linguaggio Assembly. Infatti CMOVE e CWAIT non fanno niente di più che chiamare indirettamente le istruzioni del Copper MOVE e WAIT. Con CMOVE e CWAIT si possono alterare direttamente le caratteristiche dell'hardware in qualsiasi momento della creazione di una view.

Quando si specificano istruzioni Copper con CMOVE e CWAIT, si realizza in memoria una struttura UCopList: tale struttura contiene soltanto le chiamate CMOVE e CWAIT predisposte per un particolare quadro video. Quando si chiama la funzione MrgCop, queste istruzioni vengono unite alle istruzioni Copper generate dalle funzioni MakeVPort e LoadView per formare la lista completa delle istruzioni Copper che definiscono il quadro successivo. Inoltre, se esistono effetti di animazione nel quadro in arrivo, anche le relative istruzioni Copper vengono unite nel flusso quando si esegue la funzione MrgCop.

Se il quadro seguente è uguale a quello visibile, significa che l'insieme delle istruzioni Copper non è cambiato. D'altra parte, se si sono ridefiniti sprite, viewport, oppure la struttura UCopList per questa nuova view, l'insieme completo delle istruzioni Copper risulterà variato per il nuovo quadro.

Se nell'insieme globale delle istruzioni Copper si è verificato qualche cambiamento, la nuova lista sarà sottoposta al Copper durante l'intervallo di vertical-blanking.

Il primo argomento della macro CMOVE è uCopList, nel quale occorre indicare l'indirizzo della struttura UCopList che individua tutte le istruzioni Copper indicate dal programmatore per la definizione del successivo quadro video. Tale struttura si riferisce a una lista di istruzioni MOVE e WAIT per specificare dettagliatamente i cambiamenti nella definizione del quadro.

Queste istruzioni Copper generate attraverso CMOVE prendono la forma di semplici istruzioni MOVE per spostare un valore in un registro del Copper.

Il valore da spostare viene specificato come terzo argomento della chiamata a CMOVE. Il numero di registro destinato a ricevere il valore viene

invece precisato come secondo argomento. È importante notare che la macro CMOVE non sposta realmente un valore in un registro; essa aggiunge semplicemente un'istruzione per effettuare un cambiamento nel pool d'istruzioni della struttura UCopList. Il valore indicato non viene spostato finché non viene eseguita la funzione MrgCop e, in seguito, la funzione LoadView.

Esistono diversi importanti registri hardware che si possono controllare attraverso CMOVE, inclusi i registri di colore impiegati per impostare la tavolozza (palette) dei colori per il quadro video. Sempre con CMOVE si può controllare un altro insieme di registri che include quelli relativi al modo video (alta o bassa risoluzione, single-playfield o dual-playfield, modo Hold And Modify e così via) per il quadro successivo.

Il Copper può controllare i seguenti gruppi di registri:

1. Qualsiasi registro dall'indirizzo 0x80 esadecimale in su.

2. Qualsiasi registro il cui indirizzo sia compreso tra l'esadecimale 0x40 e 0x80 quando il bit Danger del Copper è impostato a 1. Sono i secondi 32 registri del sistema.

Si noti che i registri al di sotto dell'esadecimale 0x40 (ovvero i primi 32 registri del sistema) non possono essere controllati dal Copper.

Se si vuole scrivere una propria lista d'istruzioni Copper è necessario possedere una dettagliata conoscenza sull'hardware dell'Amiga. Bisogna conoscere il nome e lo scopo di ogni registro e il numero a esso associato.

CopySBitMap

Sintassi di chiamata della funzione

CopySBitMap (layer)
A0

Scopo della funzione

Questa funzione è l'inverso della funzione SyncSBitMap. CopySBitMap copia il contenuto di parte della superbitmap assegnata a un layer nella bitmap del layer. La funzione CopySBitMap lavora soltanto con layer direttamente impostati come layer di tipo superbitmap.

Argomenti della funzione

layer Indirizzo della struttura Layer, che a sua volta contiene un puntatore alla struttura BitMap per una superbitmap; la struttura Layer dovrebbe essere preventivamente bloccata dal task chiamante.

Discussione

Ci sono due funzioni della libreria Graphics che riguardano specificamente le superbitmap e i layer a esse associati: CopySBitMap e SyncSBitMap. Queste funzioni lavorano con la struttura BitMap che definisce una superbitmap e con la struttura Layer che definisce una bitmap di layer.

Le funzioni CopySBitMap e SyncSBitMap consentono di sincronizzare i contenuti di una superbitmap con una bitmap di layer. *Sincronizzare* significa in questo contesto "combinare", allo scopo di effettuare il refresh di schermo. Una *bitmap di layer* è la porzione di un layer che si trova sullo schermo video.

Una superbitmap è una singola bitmap, spesso molto grande, dove le porzioni oscurate e le porzioni fuori schermo della bitmap di layer sono mantenute in permanenza per il refresh dello schermo; si tratta di un'area di copia di riserva per una bitmap di layer. Essa è chiamata superbitmap per due ragioni. Prima di tutto è spesso più larga della bitmap di schermo prevista per un layer. In secondo luogo essa, da sola, può soddisfare le necessità del layer di superbitmap senza ricevere aiuti da altre bitmap in sezioni diverse di memoria RAM.

Quando si ricorre alle funzioni di disegno della libreria Graphics (per esempio AreaDraw, AreaMove e AreaEnd) per disegnare in un layer di superbitmap, solo parte della figura viene visualizzata nella bitmap di layer sullo schermo. Comunque, anche le parti celate del disegno (non visibili sullo schermo) saranno interessate da tali funzioni. Le informazioni di bitmap per queste porzioni di layer vengono conservate nella superbitmap. Sono perciò i bit del bitplane di superbitmap che vengono alterati dalle funzioni di disegno per creare la porzione fuori schermo di questo layer.

Una volta che le funzioni di disegno hanno terminato il loro lavoro, risultano definite tanto le porzioni visibili quanto quelle fuori schermo. Quando il task o l'utente porta il layer in posizione visibile sullo schermo, oppure ne visualizza alcune parti, le sezioni precedentemente oscurate saranno mostrate con le più recenti informazioni di disegno previste per esse.

La superbitmap viene impiegata insieme alla bitmap di layer per eseguire cambiamenti anche quando il layer viene fatto scorrere o viene ridimensionato. Mentre lo scorrimento procede, porzioni della superbitmap vengono copiate nella bitmap di layer e porzioni di quest'ultima vengono simultaneamente copiate nella superbitmap.

Un'operazione di copia simile ha luogo quando vengono modificate le

misure del layer. Se il layer viene ingrandito, alcune porzioni della superbitmap vengono copiate nella bitmap visibile a schermo. Se il layer viene rimpicciolito, alcune porzioni della bitmap vengono copiate nella superbitmap.

La funzione CopySBitMap viene usata per copiare gli appropriati bit dai bitplane della superbitmap verso la bitmap di layer. Si presume quindi che siano state usate le funzioni di disegno per ridefinire le porzioni nascoste del layer. Come si è visto, il disegno viene effettuato nelle porzioni fuori schermo della superbitmap.

Per mostrare queste aree oscurate, è necessario copiare le sezioni appropriate della superbitmap nella bitmap di layer. Ciò è precisamente quello che fa la funzione CopySBitMap. Si noti che quest'operazione ottiene l'effetto opposto a quello della funzione SyncSBitMap, che copia una bitmap di layer in una superbitmap. Si veda anche la spiegazione della funzione SyncSBitMap.

CWAIT

Sintassi di chiamata della macro

```
CWAIT (uCopList, vertBpos, horizBpos)
       A1        DØ        D1
```

Scopo della macro

Questa macro aggiunge un'istruzione alla lista d'istruzioni Copper definite dal programmatore e già associate al task. CWAIT attende il raggiungimento della posizione verticale v e orizzontale h del pennello elettronico di scansione del video per la generazione dell'immagine. CWAIT si serve di due funzioni di basso livello della libreria Graphics che agiscono sul Copper: prima CWait per aggiungere l'istruzione WAIT del Copper, e poi CBump per far avanzare fino all'istruzione successiva il puntatore alle istruzioni Copper.

Argomenti della macro

uCopList Indirizzo della struttura UCopList.

vertBpos Posizione verticale del pennello elettronico di scansione del video relativa alla parte alta della viewport, misurata in pixel.

horizBpos Posizione orizzontale del pennello elettronico di scansione del video relativa alla parte sinistra della viewport, misurata in pixel.

Discussione

Ci sono quattro macro che riguardano specificamente le istruzioni Copper definite dal programmatore: CINIT, CMOVE, CWAIT e CEND. Tutte e quattro usano l'argomento uCopList e lavorano quindi con la struttura UCopList.

Si può utilizzare la macro CWAIT insieme con la macro CMOVE per acquisire un controllo dettagliato dello schermo video. La macro CWAIT serve per attendere una precisa posizione orizzontale e verticale del pennello elettronico di scansione del video in un dato quadro. Per esempio, si può scrivere un'istruzione che attenda fino a quando il pennello non raggiunge il pixel 640 nella linea 100 in alta risoluzione senza interlace. Quando si effettua questa chiamata alla macro CWAIT, un'istruzione Wait per il Copper viene aggiunta alle informazioni controllate dalla struttura UCopList presente in RAM.

L'istruzione CWAIT è, in genere, immediatamente seguita dall'istruzione CMOVE per porre un valore specifico in uno speciale registro hardware del sistema Amiga. Per esempio si può disporre un valore nel registro che controlla la risoluzione orizzontale dello schermo video e passare in bassa risoluzione dopo che è stata raggiunta la posizione 640,100. Questo è quanto dovrebbe essere fatto per specificare una nuova viewport in bassa risoluzione come parte del nuovo quadro che si sta definendo.

È tuttavia importante capire che conviene esaurire tutte le funzioni di alto livello della libreria Graphics prima di usare CMOVE e CWAIT. È molto più semplice programmare utilizzando le funzioni di alto livello per la maggior parte delle definizioni di schermo. Dopotutto, sono state progettate per svolgere la maggior quantità di lavoro con il minimo sforzo di programmazione. CMOVE e CWAIT vanno usate soltanto quando le funzioni di alto livello non forniscono un controllo sufficientemente dettagliato delle definizioni e degli attributi di schermo necessari per definire le viewport e le loro view.

Si ricordi che l'ordine delle istruzioni Copper è molto importante: la lista delle istruzioni deve seguire una successione che corrisponda alla progressione del pennello elettronico di scansione del video nel creare il quadro. Il movimento sullo schermo va dall'alto in basso e da sinistra a destra. Il pennello elettronico attraversa lo schermo dalla posizione 0,0 (angolo superiore sinistro) alla posizione 320,256 (angolo inferiore destro, in PAL) che costituisce la fine dell'area di presentazione video. Per questo, un'istruzione CMOVE o CWAIT che svolga un'azione nel punto 100,200 deve venire dopo un'istruzione che agisca nel punto 100,100; entrambi gli ordini d'incremento, tanto quello delle coordinate x quanto quello delle coordinate y, devono essere rispettati.

La funzione MrgCop si aspetta che le istruzioni Copper provenienti dalle sue tre sorgenti siano appropriatamente disposte in tale ordine, così come fa la struttura UCopList. La funzione MakeVPort, invece, mette automaticamente in

ordine i suoi contributi alla lista delle istruzioni Copper delle tre sorgenti unite. Altrettanto fanno le routine di animazione che generano le istruzioni Copper di controllo di sprite per il quadro video in arrivo.

Come la macro CMOVE, anche la macro CWAIT prevede l'argomento uCopList. Esso dice al sistema a quale struttura UCopList dev'essere aggiunta l'istruzione Wait del Copper. La macro CWAIT riguarda tanto la posizione verticale del pennello elettronico quanto quella orizzontale. La posizione orizzontale viene misurata nei termini prodotti dall'ultima impostazione del parametro di modo video, 320 o 640.

È tuttavia importante ricordare che la posizione verticale del pennello elettronico di scansione del video è sempre misurata in termini di schermo senza interlace, e raggiunge perciò sempre un massimo di 312 linee nei sistemi PAL. Si noti che siccome il Copper può operare nella regione di overscan dello schermo, si possono scrivere chiamate CMOVE e CWAIT per rilevare e cambiare i registri di sistema anche mentre il pennello elettronico si trova nella posizione di overscan del video, che nel sistema PAL include le prime 44 linee (le linee da 0 a 43) e le ultime 12 linee (le linee da 300 a 311).

DisownBlitter

Sintassi di chiamata della funzione

DisownBlitter ()

Scopo della funzione

Questa funzione lascia libero il Blitter, perché gli altri task possano utilizzarlo.

Argomenti della funzione

Questa funzione non prevede argomenti.

Discussione

Nella libreria Graphics ci sono dieci funzioni che riguardano specificamente la gestione delle operazioni di Blitter: BltClear, BltPattern, BltBitMap,

BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Queste funzioni consentono di controllare le velocissime operazioni DMA del Blitter.

La possibilità di accesso e di possesso esclusivo del sistema hardware è un aspetto della programmazione che non è necessario considerare per i sistemi non multitasking, mentre è di primaria importanza in un sistema multitasking come l'Amiga. Un sistema operativo multitasking permette ai programmatori di realizzare architetture parallele molto sofisticate, ma impone anche un maggiore grado di attenzione nella programmazione, e una conoscenza assai approfondita del sistema. Un aspetto di questa preparazione riguarda il controllo e la padronanza del Blitter, il più veloce dispositivo di spostamento dati disponibile nell'Amiga. Questo è il momento in cui entrano in gioco le funzioni OwnBlitter e DisownBlitter.

La funzione OwnBlitter consente di acquisire il completo controllo del Blitter per le necessità di movimento dati di uno specifico task. In tal modo, si possono accodare richieste al Blitter e attendere che esse vengano soddisfatte. La funzione DisownBlitter fa proprio l'opposto: fa cessare l'accesso esclusivo al Blitter, cosicché possano utilizzarlo anche gli altri task.

Come si può vedere, queste due funzioni sono tipiche di un sistema multitasking, dove i task spesso devono impadronirsi temporaneamente di determinate risorse hardware mentre elaborano informazioni essenziali. Una volta che l'informazione è stata elaborata, tali risorse hardware possono essere restituite al sistema, e altri task possono prenderne il controllo. Si vedano anche le spiegazioni delle funzioni OwnBlitter e WaitBlit.

Quando un task deve servirsi del Blitter in maniera esclusiva, chiama per prima cosa la funzione WaitBlit, la quale lo manda in attesa fino a quando il Blitter non ha terminato il suo compito precedente. Quando riottiene il controllo chiama la funzione OwnBlitter. A questo punto il task possiede il controllo esclusivo del Blitter e può utilizzarlo.

Una volta terminate le operazioni necessarie, il task può eseguire la funzione DisownBlitter per lasciare il Blitter a disposizione degli altri. Questa è la logica generale per l'uso del Blitter.

È interessante notare che, contrariamente al Blitter, il controllo del Copper non viene mai assunto dai task in modo esclusivo. Il Copper, infatti, è principalmente legato alla presentazione di schermo, ed è quindi sempre necessario al sistema e agli altri task.

DisposeRegion

Sintassi di chiamata della funzione

DisposeRegion (region)
A0

Scopo della funzione

Questa funzione libera la memoria allocata per tutte le strutture Rectangle che costituiscono la regione; in seguito, libera la memoria allocata per la struttura Region indicata come argomento.

Argomenti della funzione

region Indirizzo della struttura Region che definisce la regione.

Discussione

Ci sono sei funzioni della libreria Graphics che riguardano specificamente le regioni e i rettangoli di delimitazione: AndRectRegion, ClearRegion, DisposeRegion, NewRegion, OrRectRegion e XorRectRegion. Queste funzioni consentono di costruire regioni di qualsiasi forma geometrica per poi associarle ai layer. Si vedano anche le spiegazioni delle altre funzioni sopraelencate.

Una regione è definita come un insieme di rettangoli di delimitazione. Una volta che tutti i rettangoli di delimitazione sono stati definiti, e sono stati riuniti in una regione, è possibile associare la regione a una bitmap e disegnare sulla bitmap con la sicurezza che i disegni verranno eseguiti solo nelle aree della regione.

In tal modo è possibile alterare parte di una bitmap di layer in modo molto selettivo. Questo metodo di disegno si aggiunge alle altre tecniche spiegate in questo capitolo. Il maggiore vantaggio di limitare il disegno a una regione è la possibilità di modificare solo alcune parti di una bitmap di layer, rendendo perciò più rapide le operazioni di disegno.

Non si confondano questi tipi di regioni con le regioni di disegno create da funzioni come AreaDraw, AreaMove, AreaEnd e RectFill. Bisogna inoltre fare attenzione a non confondere i rettangoli di delimitazione con i rettangoli riempiti dalla funzione RectFill.

Sei strutture chiave giocano una parte importante nell'uso e nella definizione delle regioni; si tratta delle strutture Layer, Layer_Info, Region, Rectangle, ClipRect e DamageList. Per le spiegazioni relative alle strutture Layer, Layer_Info, ClipRect e DamageList, si veda il capitolo 5.

Quando assegnamo una nuova regione con la funzione NewRegion, viene automaticamente assegnata memoria per la struttura Region. La funzione DisposeRegion serve a liberare la memoria precedentemente assegnata con la funzione NewRegion. Ciò rende la memoria disponibile per altri usi. DisposeRegion si aspetta come argomento l'indirizzo della struttura di tipo Region che il task ottiene quando chiama la funzione NewRegion.

Draw

Sintassi di chiamata della funzione

```
Draw (rastPort,     x,     y)
      A1            DØ     D1
```

Scopo della funzione

Questa funzione disegna una linea, dalla posizione in cui si trova la penna al punto di coordinate x,y.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort che svolge il lavoro di controllo e indica dove va disegnata la linea.
x	Coordinata orizzontale del secondo punto della linea.
y	Coordinata verticale del secondo punto della linea.

Discussione

Ci sono due funzioni, nella libreria Graphics, che riguardano direttamente il disegno delle linee: Draw e PolyDraw. La funzione Draw disegna una linea che parte dalla posizione della penna e arriva al punto indicato come argomento. PolyDraw utilizza invece un array di punti con coordinate x,y per disegnare un insieme di linee che possono anche definire un poligono.

Le funzioni Draw e PolyDraw richiedono entrambe che venga indicato l'indirizzo della struttura RastPort nella cui bitmap devono produrre il disegno.

La funzione Draw lavora con una struttura RastPort, che controlla il disegno all'interno della relativa bitmap. Durante il disegno delle linee, si possono alterare i valori dei parametri di disegno mediante una serie di chiamate alle funzioni SetDrMd, SetAPen e SetBPen e alla macro SetOPen, seguite da una chiamata alla funzione Draw e alla funzione Move per disegnare la linea. Ogni linea interesserà il contenuto della bitmap controllata dalla struttura RastPort. Tutte queste informazioni per il controllo del disegno vengono conservate nella definizione della struttura RastPort. La struttura

RastPort è una struttura dinamica che cambia ogni volta che viene alterato uno dei suoi parametri. Si può modificarla ogni volta che si effettua il disegno di una linea, oppure si possono impiegare diverse strutture RastPort per la stessa bitmap. Ciascuna di esse avrà valori diversi per i parametri di disegno, e per usarne un'altra basterà cambiare il puntatore alla struttura.

Non ci si deve lasciar confondere dalla somiglianza tra le funzioni Draw e AreaDraw. AreaDraw prepara la rappresentazione vettoriale di un insieme di linee per definire un'area che sarà riempita con un colore o con una matrice grafica; Draw produce subito una linea che nulla ha a che vedere con il riempimento di aree.

Si tenga inoltre presente la distinzione tra le funzioni Draw e PolyDraw. PolyDraw richiede che le coppie di valori x,y vengano definite in una tavola, una volta per tutte. Draw richiede, di volta in volta, la definizione di un singolo punto di coordinate x,y; viene poi disegnata un'unica linea tra la posizione della penna in quell'istante e il punto, e la penna viene spostata nella nuova posizione.

DrawCircle

Sintassi di chiamata della macro

DrawCircle (rastPort, centerX, centerY, radius)
A1 DØ D1 D2

Scopo della macro

Questa macro disegna un cerchio di raggio e centro dati, nella bitmap controllata dalla struttura RastPort specificata. Viene tracciata soltanto la circonferenza, e non è previsto alcun riempimento. DrawCircle svolge il suo compito chiamando la funzione DrawEllipse. Questa macro non restituisce alcun valore.

Argomenti della macro

rastPort — Indirizzo della struttura RastPort che definisce i parametri di disegno della bitmap sulla quale si desidera disegnare.

centerX	Coordinata x del centro del cerchio, calcolata nel sistema di coordinate della bitmap.
centerY	Coordinata y del centro del cerchio.
radius	Raggio del cerchio (in numero di pixel).

Discussione

Ci sono due macro che riguardano i cerchi: AreaCircle e DrawCircle. AreaCircle è una macro che chiama la funzione AreaEllipse e DrawCircle è una macro che chiama la funzione DrawEllipse. Chiamando la macro AreaCircle si proroga il disegno di un cerchio pieno ponendone i parametri tra le informazioni controllate dalla tavola di vettori definita dalla struttura AreaInfo. Il cerchio pieno apparirà solo quando si esegue la funzione AreaEnd.

Nel secondo modo si può disegnare immediatamente il cerchio impiegando la macro DrawCircle. Si noti che la macro DrawCircle produce soltanto il contorno del cerchio. Esso viene disegnato con il colore FgPen impostato con la più recente chiamata a SetAPen. Se si vuole riempire il cerchio con un colore pieno o una matrice grafica, si può poi chiamare la funzione Flood.

DrawEllipse

Sintassi di chiamata della funzione

```
DrawEllipse (rastPort, centerX, centerY, horiz_radius, vert_radius)
             A1        D0:16    D1:16    D2:16         D3:16
```

Scopo della funzione

Questa funzione disegna un'ellisse nella bitmap controllata dalla struttura RastPort indicata come argomento. Produce soltanto il contorno dell'ellisse, che non viene quindi riempita. Questa funzione non restituisce alcun valore.

Argomenti della funzione

rastPort — Indirizzo della struttura RastPort nella cui bitmap si desidera disegnare l'ellisse.

centerX — Coordinata x del centro dell'ellisse, calcolata nel sistema di coordinate della bitmap.

centerY — Coordinata y del centro dell'ellisse.

horiz_radius — Semiasse orizzontale dell'ellisse (in numero di pixel), misurato nella risoluzione orizzontale del sistema di coordinate della bitmap.

vert_radius — Semiasse verticale dell'ellisse (in numero di pixel), misurato nella risoluzione verticale del sistema di coordinate della bitmap.

Discussione

Ci sono due funzioni che riguardano le ellissi: AreaEllipse e DrawEllipse. Esse forniscono due modi per inserire un'ellisse in una bitmap. Nel primo modo si può chiamare la funzione AreaEllipse per prorogare disegno e riempimento, ponendone la rappresentazione vettoriale nella tavola dei vettori della struttura AreaInfo. Nel secondo modo si può disegnare immediatamente l'ellisse impiegando la funzione DrawEllipse. Si noti che la funzione DrawEllipse produce soltanto il contorno. Tale contorno viene disegnato nel colore FgPen impostato con la più recente chiamata della funzione SetAPen. Se si vuole riempire l'ellisse con un colore pieno o con una matrice grafica, si può poi chiamare la funzione Flood.

Flood

Sintassi di chiamata della funzione

```
Flood (rastPort, floodmode,    x,    y)
       A1        D2            DØ    D1
```

Scopo della funzione

Questa funzione riempie la bitmap in modo molto simile a un'operazione di riempimento di aree eseguita con la funzione AreaEnd. Flood cerca la posizione x,y all'interno della bitmap, e poi riempie tutti i pixel adiacenti in accordo con quanto indicato nell'argomento floodmode. Flood utilizza i parametri di disegno che si applicano alle abituali operazioni di riempimento di aree.

Argomenti della funzione

rastPort — Indirizzo della struttura RastPort nella cui bitmap agirà la funzione.

floodmode — Argomento che controlla il modo di effettuare l'operazione di riempimento; quando viene impostato a 0, la funzione Flood riempie tutti i pixel che non hanno lo stesso colore previsto dal parametro di disegno AOlPen; quando invece l'argomento floodmode viene impostato a 1, la funzione riempie i pixel che hanno lo stesso colore del pixel che si trova alle coordinate x,y.

x — Coordinata orizzontale del pixel, calcolata nel sistema di coordinate della bitmap.

y — Coordinata verticale del pixel, calcolata nel sistema di coordinate della bitmap.

Discussione

Nella libreria Graphics ci sono sei funzioni di disegno che riguardano direttamente la creazione di aree nel sistema Amiga: AreaDraw, AreaEnd, AreaMove, InitArea, RectFill e Flood. Si vedano anche le spiegazioni delle altre funzioni sopraelencate.

Ci sono diversi modi per riempire un'area con informazioni di disegno. Il primo consiste nell'uso della funzione SetRast per impostare con un certo colore un'intera bitmap; ciò però non consente alcuna possibilità di riempire con matrici grafiche, cioè creando uno sfondo variegato. Nel secondo modo si può usare la funzione AreaEnd per riempire un insieme di aree, definite con le funzioni AreaDraw e AreaMove. Si ricordi che il modo di disegno, i parametri FgPen, BgPen, AOlPen, le matrici grafiche e di continuità sono tutti elementi che vanno impostati preliminarmente, dal momento che le funzioni ne tengono

conto. Un terzo modo di procedere consiste nel ricorso alla funzione RectFill che permette di riempire aree rettangolari con determinate matrici grafiche; le caratteristiche di disegno possono essere controllate come nel modo precedente. La funzione Flood costituisce l'ultimo modo per riempire un'area.

Flood costituisce una funzione a parte, rispetto alle altre funzioni di riempimento di aree, dato che possiede due modi operativi. Il primo, chiamato modo outline (o modo 0), utilizza due sorgenti d'informazione per svolgere le sue operazioni. Per prima cosa rileva l'impostazione di AOlPen. Rileva poi i colori di tutti i punti che si dipartono dal punto x,y, in tutte le direzioni, procedendo verso i bordi dell'area. Se il colore di tali punti *non* corrisponde al colore AOlPen, i punti vengono riempiti utilizzando le informazioni fornite dalla struttura RastPort.

La seconda modalità, chiamata modo colore o modo 1, opera diversamente. I pixel vengono riempiti con un diverso criterio di scelta. In particolare, i pixel il cui colore corrisponde al colore del pixel definito nella funzione Flood e situato alle coordinate x,y, verranno riempiti utilizzando le informazioni presenti nella definizione della struttura RastPort. I pixel che non corrispondono al colore di tale punto non saranno modificati. La ricerca dei punti che soddisfano questa condizione continua finché non viene raggiunto il primo pixel sul bordo dell'area prescelta.

Il modo 1 prosegue fino al bordo mentre il modo 0 spesso sospende la ricerca e le operazioni di riempimento prima d'incontrare il bordo. Nel modo 1 le operazioni di riempimento interessano ogni pixel che soddisfi il criterio assegnato, raggiungendo quindi i limiti dell'area, il che significa che l'area viene riempita da un dato punto di coordinate x,y in avanti, in tutte le direzioni, finché non si raggiunge ogni bordo. La funzione incorpora un algoritmo per rilevare la presenza dei bordi. La funzione Flood è generalmente più lenta delle altre funzioni di riempimento di aree.

Per poter utilizzare la funzione Flood è necessario che alla struttura RastPort sia associata una struttura TmpRas già inizializzata e di dimensioni almeno pari a quelle dell'area da modificare.

FreeColorMap

Sintassi di chiamata della funzione

FreeColorMap (colorMap)
AØ

Scopo della funzione

Questa funzione libera la memoria preventivamente allocata per una struttura ColorMap dalla funzione GetColorMap.

Argomenti della funzione

colorMap Indirizzo della struttura ColorMap originariamente allocata con la funzione GetColorMap.

Discussione

Ci sono due funzioni di disegno della libreria Graphics che riguardano specificamente la gestione della mappa colore: FreeColorMap e GetColorMap. GetColorMap alloca memoria per una struttura ColorMap. FreeColorMap libera la memoria così occupata. La memoria viene restituita al pool di memoria libera del sistema.

FreeColorMap richiede che sia indicato come argomento l'indirizzo della struttura ColorMap originariamente allocata da GetColorMap. Il più importante parametro della struttura ColorMap è il puntatore ColorTable, che individua in memoria la tavola dei colori; si tratta di un array di valori RGB da abbinare ai registri di colore. Tale tavola consente di determinare i colori da impiegare in una viewport.

FreeCopList

Sintassi di chiamata della funzione

FreeCopList (copList)
A0

Scopo della funzione

Questa funzione libera la memoria associata alla struttura CopList, la lista di Copper intermedia. Ciò include tutta la memoria per le istruzioni intermedie del Copper richieste per definire un quadro video. Questo blocco di memoria viene restituito al pool di memoria libera del sistema.

Argomenti della funzione

copList Indirizzo della struttura CopList.

Discussione

Ci sono tre funzioni che liberano la memoria allocata per conservare le istruzioni Copper necessarie per definire il quadro video successivo: FreeCopList, FreeCprList e FreeVPortCopLists. Ciascuna di queste funzioni riguarda una diversa categoria di istruzioni Copper.

Esistono diverse categorie di istruzioni Copper. Le istruzioni intermedie, e quelle hardware. Le istruzioni intermedie sono a loro volta ottenute raggruppando quelle di viewport e quelle definite dal programmatore. La funzione FreeCopList libera la memoria allocata per le istruzioni intermedie. La funzione FreeVPortCopLists libera la memoria allocata per le istruzioni di viewport. La memoria per le istruzioni definite dal programmatore dev'essere allocata e liberata con le funzioni di gestione della memoria contenute nella libreria Exec. Prima che queste categorie di istruzioni vengano sottoposte al Copper, occorre riunirle in un'unica lista, chiamata lista delle istruzioni hardware del Copper. La funzione FreeCprList libera la memoria allocata per queste istruzioni hardware.

La struttura CopList riguarda le istruzioni intermedie del Copper; essa contiene un puntatore al successivo blocco di istruzioni intermedie del Copper. Tutte le istruzioni che fanno parte dell'insieme intermedio sono quindi collegate in una lista. La struttura CopList contiene un contatore del numero di istruzioni intermedie del Copper contenute in ciascun blocco.

Per comprendere la differenza tra istruzioni intermedie e istruzioni hardware è necessario capire in che modo il sistema grafico giunge all'insieme finale delle istruzioni hardware del Copper. Quest'operazione si svolge in due fasi. La prima fase rileva tutte le definizioni di viewport e di view (includendo anche le istruzioni del Copper definite dal programmatore) e giunge all'insieme delle istruzioni intermedie. Tale insieme comprende quindi le istruzioni del Copper di video, colore, sprite e quelle definite dal programmatore. Sebbene le istruzioni definite dal programmatore costituiscano una categoria separata, esse vengono ugualmente aggiunte alle istruzioni intermedie.

Questo insieme di istruzioni del Copper viene chiamato *intermedio* perché non è ancora nella forma finale (in particolar modo, non è nell'ordine definitivo) necessaria per guidare l'hardware del coprocessore Copper nella definizione del quadro video. La funzione MrgCop si occupa di disporre le varie componenti della lista nell'ordine appropriato. Questa operazione genera la lista finale delle istruzioni che dovrà poi essere sottoposta al Copper per la definizione del successivo quadro video.

Il sistema alloca automaticamente uno o più blocchi di memoria per tali istruzioni intermedie. Si vedano anche le spiegazioni relative alle funzioni FreeCprList e FreeVPortCopLists.

FreeCprList

Sintassi di chiamata della funzione

FreeCprList (cprList)
AØ

Scopo della funzione

Questa funzione libera tutta la memoria associata alla struttura CprList che definisce la lista di istruzioni hardware del Copper. Tale blocco di memoria viene restituito al pool di memoria libera del sistema.

Argomenti della funzione

cprList	Indirizzo della struttura CprList

Discussione

Ci sono tre funzioni che liberano la memoria assegnata per conservare le istruzioni del Copper relative al successivo quadro video: FreeCopList, FreeCprList e FreeVPortCopLists. Ciascuna di queste funzioni riguarda una diversa categoria di istruzioni del Copper.

Tutte e tre queste funzioni rilasciano la memoria allocata per la particolare categoria d'istruzioni del Copper a cui si riferiscono. Ci sono quattro categorie di istruzioni del Copper: intermedie, hardware, di viewport e definite dal programmatore.

Le istruzioni Copper contenute nella struttura CprList sono le istruzioni finali *hardware* del Copper. Tali istruzioni hanno origine direttamente dal risultato della chiamata alla funzione MrgCop. Esse contengono tutte le istruzioni WAIT, MOVE e SKIP richieste per definire un quadro video. Queste istruzioni vengono ordinate secondo le coordinate crescenti y,x. La struttura CprList riguarda le istruzioni di controllo hardware del Copper; essa contiene un puntatore a una lista contenente tutte le istruzioni hardware per il Copper necessarie per definire un quadro video. La variabile di partenza della struttura CprList punta al primo blocco di istruzioni della serie.

Quando l'insieme di istruzioni non serve più, si dovrebbe liberare tramite la funzione FreeCprList la memoria a esso riservata.

FreeRaster

Sintassi di chiamata della funzione

FreeRaster (memBlock, width, height)
A0 D0 D1

Scopo della funzione

Questa funzione libera il blocco di memoria precedentemente allocato per un bitplane tramite la funzione AllocRaster, e lo restituisce al pool di memoria libera del sistema. Vanno usati gli stessi valori di larghezza e altezza impiegati per la chiamata alla funzione AllocRaster.

Argomenti della funzione

memBlock	Indirizzo del blocco di memoria occupato dal bitplane che si desidera eliminare.
width	Larghezza in bit del bitplane.
height	Altezza in bit del bitplane.

Discussione

Ci sono quattro funzioni della libreria Graphics che riguardano specificamente i bitplane: AllocRaster, FreeRaster, InitTmpRas e ScrollRaster. Una bitmap è un'area di memoria costituita da un certo numero di bitplane, che dipende dal numero di colori che si vogliono visualizzare.

Nella Tavola 2.2 è indicata la quantità di memoria necessaria per ogni tipo di modo video.

Tre sono le cose da notare nella Tavola 2.2:

- con il modo video dual-playfield non è possibile impiegare più di otto colori. Dovendo usare più colori si dovrà rinunciare a questo modo video.
- Con il modo video ad alta risoluzione non si possono usare più di 16 colori contemporaneamente.
- Se si vogliono sullo schermo più di 32 colori, si *deve* usare il modo Hold And Modify oppure il modo Extra Half-Brite.

Ora faremo alcuni esempi per avere un'idea di quanta memoria è necessaria con i vari modi grafici e come effettuare il calcolo; in questa trattazione diamo per scontato che lo standard video in uso sia il PAL. Si consideri un video costituito da 320 x 256 pixel. Si tratta del modo video a bassa risoluzione senza interlace. Questo è il tipo di presentazione meno "costoso" in termini di memoria. Ciascuna linea di schermo è composta da 320 pixel, per un totale di 40 byte. Con un solo bitplane si ottiene una bitmap a due colori, che quindi occupa in memoria 10240 byte. Se invece si desidera una bitmap a quattro colori, occorrono due bitplane per un totale di 20480 byte; per avere 32 colori (cinque bitplane per ogni bitmap) dobbiamo disporre di 51200 byte d'informazioni in memoria.

Ciò definisce le esigenze di memoria per i più comuni tipi di schermo. Si possono dedurre le richieste di memoria per schermi più complessi basandosi sui tipi fondamentali presentati dalla Tavola 2.2. Per il modo video dual-playfield con otto colori per playfield, sono necessari 30720 byte per ciascun playfield, per un totale di 61440 byte.

Ora si consideri un video costituito da 640 per 256 pixel. Si tratta del modo video ad alta risoluzione senza interlace. Questo tipo di presentazione possiede esattamente il doppio dei pixel in ciascuna linea di schermo rispetto al modo in bassa risoluzione. Ciò porta a 20480 byte la quantità di memoria necessaria

Tavola 2.2: *Numero di colori e di bitplane per i vari modi video*

Numero di colori	Numero di bitplane	Modi video possibili
2	1	Tutti tranne Hold And Modify e Extra HB
4	2	Tutti tranne Hold And Modify e Extra HB
8	3	Tutti tranne Hold And Modify e Extra HB
16	4	Tutti tranne dual-playfield, Hold And Modify e Extra HB
32	5	Tutti tranne alta risoluzione, dual-playfield, Hold And Modify e Extra HB
64 o 4096	6	Tutti tranne alta risoluzione e dual-playfield

per ogni bitplane. In questo caso, una presentazione a 16 colori, cioè una bitmap da quattro bitplane (il massimo consentito nel modo in alta risoluzione), richiede 81920 byte d'informazioni video. Per un video dual-playfield con otto colori sono necessari 61440 byte per playfield, per un totale di 122880 byte.

Si consideri ora un video costituito da 640 x 512 pixel. Si tratta di uno schermo ad alta risoluzione in interlace, la più alta risoluzione video di cui possiamo disporre. Con questa risoluzione, ciascun bitplane richiede 40960 byte d'informazioni. Per una presentazione a 16 colori sono quindi necessari ben 163840 byte d'informazioni.

Si consideri ora un video single-playfield in modo Hold And Modify da 320 x 256 pixel. Si tratta di uno schermo a bassa risoluzione senza interlace. Il vantaggio del modo Hold And Modify consiste nella possibilità di mostrare 4096 colori in un singolo quadro video. Per raggiungere questo obiettivo si devono specificare sei bitplane, e quindi con la risoluzione indicata sono sufficienti 61440 byte per l'intera bitmap.

La tecnica del double-buffer, usata per rendere più semplici gli aggiornamenti nello sviluppo di una presentazione video a cambiamento rapido richiede ancor più memoria. Si devono predisporre due aree di memoria, ciascuna contenente una bitmap. La prima viene usata per guidare l'hardware video nella produzione del quadro visibile. Nello stesso momento, la seconda bitmap viene aggiornata per produrre le informazioni necessarie per la definizione del quadro successivo. È ovvio che la tecnica double-buffer raddoppia le esigenze di memoria per ogni schermo.

Da tutto ciò si deduce quanto sia importante nella programmazione la funzione FreeRaster. Appena le informazioni di bitplane e/o bitmap non sono più necessarie, la memoria che veniva da esse utilizzata va restituita al pool di memoria libera del sistema con la funzione FreeRaster.

FreeVPortCopLists

Sintassi di chiamata della funzione

FreeVPortCopLists (viewPort)
A0

Scopo della funzione

Questa funzione libera la memoria assegnata a tutte le liste intermedie di Copper e alle rispettive intestazioni connesse a una particolare viewport.

FreeVPortCopLists cerca le liste Copper per il video, il colore, gli sprite e la lista del programmatore, e poi chiama la funzione FreeMem per liberare la memoria occupata da tali liste. Inoltre imposta a 0 i puntatori DspIns, ClrIns, SprIns, UCopIns presenti nella struttura ViewPort.

Argomenti della funzione

viewPort Indirizzo di una struttura ViewPort.

Discussione

Ci sono tre funzioni che liberano la memoria allocata per conservare le istruzioni di Copper: FreeCopList, FreeCprList e FreeVPortCopLists.

La funzione FreeVPortCopLists lavora con la struttura ViewPort e con l'insieme completo delle strutture intermedie CopList, necessarie per definire la presentazione di una particolare ViewPort. Questa funzione riguarda la memoria allocata per tutte le liste Copper relative a una data viewport. In particolar modo, FreeVPortCopLists può essere usata per liberare la RAM occupata dalle seguenti liste di istruzioni video per il Copper: liste per l'alterazione dei registri di colore, liste di routine per l'animazione, liste definite dal programmatore e conservate nella struttura UCopList.

FreeVPortCopLists lavora con una sola viewport per volta. Se si desidera liberare la memoria assegnata a tutte le viewport di una view bisogna chiamare la funzione più volte.

La struttura ViewPort contiene quattro puntatori:

DspIns	Indirizzo di una struttura CopList per le istruzioni video del Copper.
SprIns	Indirizzo di una struttura CopList per le istruzioni sprite del Copper.
ClrIns	Indirizzo di una struttura CopList per le istruzioni Copper relative al registro colore degli sprite.
UCopList	Indirizzo di una struttura UCopList per le istruzioni Copper definite dal programmatore.

Ciascuno di questi puntatori viene impostato a 0 dalla funzione FreeVPortCopLists.

GetColorMap

Sintassi di chiamata della funzione

```
colorMap = GetColorMap (entries)
D0                      D0
```

Scopo della funzione

Questa funzione alloca memoria per una struttura ColorMap, la inizializza e ne restituisce l'indirizzo. La struttura ColorMap risultante viene collegata alla struttura ViewPort per memorizzare i valori dei colori che costituiscono la tavolozza (palette). Se non risulta possibile allocare la memoria necessaria, la funzione restituisce il valore zero.

Argomenti della funzione

entries	Numero degli elementi-colore che devono essere presenti nella tavola puntata dalla struttura ColorMap. Ogni elemento occupa una word nella quale i 12 bit meno significativi devono indicare la composizione RGB del colore, che verrà memorizzata nel corrispondente registro di colore quando la palette diventerà attiva. Il numero di elementi contenuti nella palette deve corrispondere al numero di colori impiegati nella bitmap che contiene la viewport.

Discussione

Ci sono due funzioni di disegno della libreria Graphics che riguardano specificamente la tavola dei colori e la struttura ColorMap: FreeColorMap e GetColorMap. GetColorMap alloca la memoria necessaria per una struttura ColorMap. Il task usa questa struttura per definire una palette e assegnarla successivamente a una delle viewport che sta preparando.

FreeColorMap libera la memoria occupata dalla struttura ColorMap indicata come argomento. La funzione GetColorMap non richiede di specificare argomenti puntatori a strutture, e restituisce l'indirizzo della struttura ColorMap che ha creato e inizializzato. La funzione FreeColorMap impiega

come argomento puntatore l'indirizzo della struttura ColorMap che il task ha ottenuto chiamando GetColorMap. Si veda anche la spiegazione relativa alla funzione LoadRGB4.

GetRGB4

Sintassi di chiamata della funzione

```
colorvalue = GetRGB4 (colorMap, entry)
DØ                    AØ        DØ
```

Scopo della funzione

Questa funzione permette di leggere il valore RGB contenuto nell'elemento della tavolozza indicato dal parametro entry. GetRGB4 restituisce il valore -1 se l'argomento entry non indica un elemento valido nella tavola definita dalla struttura ColorMap indicata come primo argomento; altrimenti restituisce il valore RGB contenuto nella word indicata dall'argomento entry. In questo valore hanno significato solo i 12 bit meno significativi, nei quali sono definiti i tre colori fondamentali della sintesi additiva. I bit da 0 a 3 vengono assegnati al blu; i bit da 4 a 7 al verde; i bit da 8 a 11 al rosso. Dal momento che ogni colore fondamentale può assumere valori compresi fra 0 e 15 (16 valori esprimibili su quattro bit), con ogni elemento della tavolozza possiamo rappresentare 4096 combinazioni di sfumature diverse, cioè 4096 colori.

Argomenti della funzione

colorMap — Indirizzo della struttura ColorMap che contiene l'array della tavola dei colori. In ogni elemento dell'array dev'essere memorizzato un valore RGB di colore espresso su 12 bit, in modo che quando la palette viene associata a una viewport e diventa attiva, questi valori RGB possano essere inseriti nei registri colore dell'hardware.

entry — Numero dell'elemento nell'array della tavola dei colori.

Discussione

Nella libreria Graphics ci sono tre funzioni che riguardano specificamente i colori assegnati alle viewport e da esse utilizzati: GetRGB4, LoadRGB4 e SetRGB4.

LoadRGB4 aggancia a una viewport una tavolozza definita da una struttura ColorMap. GetRGB4 consente di leggere il valore RGB di un particolare colore definito nella tavola dei colori della struttura ColorMap. Una volta che si ottiene il valore RGB corrispondente a un particolare registro di colore, si può usarlo o alterarlo. In particolare, lo si può rimemorizzare nella tavolozza tramite la funzione SetRGB4, la quale consente d'impostare un determinato colore nella tavola dei colori associata a una struttura ColorMap.

La funzione GetRGB4 richiede come argomento l'indirizzo della struttura ColorMap sulla cui tavola dei colori deve agire.

InitArea

Sintassi di chiamata della funzione

```
InitArea (areaInfo, buffer, maxvectors)
          A0        A1      D0
```

Scopo della funzione

Questa funzione inizializza la matrice che raccoglierà i vettori durante la preparazione dell'area di riempimento. Si tratta di un insieme di punti terminali (vertici) che definiscono un'area. L'area di riempimento potrà arrivare ad avere un massimo di "maxvectors" coppie x,y. Per valutare le dimensioni della matrice, e quindi del buffer che la conterrà, il task deve tener conto che per ogni vettore occorrono cinque byte, e determinare di conseguenza il valore, espresso in byte, da inserire nell'argomento maxvectors.

Durante la definizione dell'area di riempimento (effettuata tramite le funzioni AreaMove, AreaDraw, AreaEllipse e AreaEnd) occorre che nella matrice dei vettori ci sia abbastanza spazio da contenere il massimo numero di vettori (vertici dell'area) che potrebbero essere richiesti.

Se non c'è abbastanza spazio in memoria, InitArea restituisce il valore -1. Se invece non incontra problemi di memoria, la struttura AreaInfo viene inizializzata in modo da essere pronta a ricevere i vettori prodotti dalle funzioni AreaMove, AreaDraw e AreaEllipse. Le routine grafiche interne che collaborano con la funzione suddividono la matrice dei vettori in due parti per salvare sia le coordinate sia i flag.

Argomenti della funzione

areaInfo Indirizzo della struttura AreaInfo.

buffer Indirizzo del blocco di memoria che il task utilizza per la matrice dei vettori x,y; questo buffer deve iniziare allineato alle word.

maxvectors Massimo numero di coppie x,y (vettori) che il buffer è predisposto ad accogliere. Nella determinazione di questo argomento, si tenga presente che ogni vettore richiede cinque byte per essere memorizzato.

Discussione

Nella libreria Graphics ci sono sette funzioni di disegno che riguardano specificamente il riempimento delle aree: AreaDraw, AreaEnd, AreaMove, AreaEllipse, InitArea, RectFill e Flood. Queste funzioni permettono di definire aree di bitmap che possono essere utilizzate per realizzare una presentazione video complessa partendo da un gruppo di aree. Si vedano anche le spiegazioni delle altre funzioni or ora citate.

Le funzioni AreaDraw, AreaMove, AreaEllipse e AreaEnd riguardano i vettori che definiscono le aree di riempimento. I vettori sono le linee che congiungono i punti definiti dai valori x,y. Se si vuole realizzare un insieme complesso di aree da riempire, ci sarà bisogno di un numero elevato di punti x,y per una definizione completa. È a questo punto che entra in gioco la funzione InitArea.

La funzione InitArea ha principalmente due scopi. Primo, predispone il buffer in RAM per conservare tutti i valori x,y necessari per definire una determinata area di riempimento. Secondo, inizializza opportunamente la struttura AreaInfo che servirà per gestire la matrice dei vettori. La struttura RastPort possiede un puntatore adibito a individuare in memoria la relativa struttura AreaInfo.

La struttura AreaInfo agisce come mezzo di registrazione. Il parametro Count in essa contenuto indica il numero dei vertici (valori x,y) conservati nella tavola dei vettori allocata nel buffer. Il parametro VctrTbl individua in memoria la tavola nella quale sono conservati i vettori x,y. Il secondo parametro puntatore, VctrPtr, individua in memoria il vettore che sta per essere elaborato dalla struttura AreaInfo.

Si noti che il buffer specificato nella chiamata alla funzione InitArea dev'essere abbastanza grande da contenere la più estesa tra le operazioni di riempimento di aree che saranno effettuate tramite la struttura AreaInfo. Si ricordi che si deve iniziare la sequenza di riempimento definendo l'appropriato modo di disegno, la matrice grafica di riempimento per le aree e quella di continuità per le linee, nonché i parametri FgPen, BgPen e AOlPen della

struttura RastPort. Queste necessità limitano la grandezza del buffer che sarà di volta in volta necessario allo svolgimento del disegno; ciascuna chiamata alla funzione AreaEnd utilizza ben precise impostazioni di disegno. Quindi, si possono disegnare e riempire soltanto le aree che hanno le stesse caratteristiche. Si noti, inoltre, che le chiamate a AreaMove, AreaDraw e AreaEnd utilizzano un solo vettore della tavola, mentre la funzione AreaEllipse e la macro che la utilizza, AreaCircle, ne usano due.

Conviene raccogliere in sottogruppi tutte le aree con analoghe caratteristiche di disegno. Questo va bene anche nel caso che le aree siano disperse nella bitmap che si sta definendo. La misura del buffer, impostata precedentemente con la funzione InitArea dovrebbe essere basata sul numero massimo dei valori x,y richiesti per definire tutte le aree per il sottogruppo più esteso. Una volta che si conosca questo numero, si chiama InitArea, proseguendo poi con le chiamate ad AreaDraw, AreaMove, AreaEllipse e concludendo con AreaEnd.

Quando la chiamata ad AreaEnd ha avuto termine, le aree previste sono state effettivamente disegnate, e si può procedere nello stesso modo per il sottogruppo successivo. L'operazione va svolta per tutti i sottogruppi che interessano la prima bitmap. In seguito si può procedere con la successiva bitmap. Si continua in tal modo fino a che non sono state definite tutte le aree da riempire.

Riducendo in sottogruppi i riempimenti di aree, non è necessario che il buffer per i vettori sia particolarmente ampio. Se lo spazio RAM a disposizione è ristretto, si può limitare la misura del buffer prodotto da InitArea perché tratti soltanto una particolare sezione del disegno, anche se altre sezioni possiedono le stesse caratteristiche.

InitBitMap

Sintassi di chiamata della funzione

```
InitBitMap (bitMap, depth, width, height)
            AØ      DØ     D1     D2
```

Scopo della funzione

Questa funzione inizializza con i valori che descrivono una particolare bitmap la struttura BitMap indicata come primo argomento. Questi valori sono il numero di bitplane e le loro dimensioni in larghezza e in altezza. InitBitMap dev'essere chiamata prima d'iniziare a utilizzare la bitmap. Nella struttura BitMap gli elementi del vettore Planes, adibiti a individuare in memoria i vari bitplane che costituiscono la bitmap, non vengono inizializzati da questa

funzione. Lo stesso vale per i bitplane veri e propri, che il task deve provvedere ad allocare uno per uno per proprio conto, tramite la funzione AllocRaster.

Argomenti della funzione

bitMap	Indirizzo della struttura BitMap che il task deve aver già allocato.
depth	Numero dei bitplane che costituiranno la bitmap.
width	Larghezza in bit della bitmap.
height	Altezza in bit della bitmap.

Discussione

Ci sono tre funzioni della libreria Graphics dedicate in special modo alle bitmap: CopySBitMap, InitBitmap e SyncSBitMap. Si vedano anche le spiegazioni delle funzioni CopySBitMap e SyncSBitMap.

La procedura per lo sviluppo di una view sul video dell'Amiga consiste in parte nella determinazione delle bitmap da usare per definire la view. Per creare una bitmap si devono eseguire le seguenti operazioni. Si alloca una struttura BitMap e la si inizializza tramite la funzione InitBitMap. Poi si chiama ripetutamente la funzione AllocRaster per allocare nella chip RAM tutti i bitplane richiesti; infine l'indirizzo di ogni bitplane dev'essere memorizzato nel vettore Planes della struttura BitMap.

Impostare la struttura BitMap è compito della funzione InitBitMap. Quando si esegue una chiamata a questa funzione, uno degli argomenti che vanno specificati è la profondità della bitmap, cioè il numero di bitplane che la compongono, il quale stabilisce quanti colori saranno contemporaneamente visualizzabili sulla bitmap. Quando si chiama InitBitMap si devono anche fornire gli argomenti width e height. Queste variabili definiscono la misura della bitmap in memoria. Larghezza e altezza possono estendersi fino a 1024 pixel. In genere, queste dimensioni sono le stesse che si indicano nella chiamata alla funzione AllocRaster per allocare i bitplane.

Si tenga presente che la bitmap conservata in RAM può arrivare come estensione fino a 1024 x 1024 pixel; quindi può capitare che sul video se ne possa visualizzare soltanto una parte. Quest'operazione viene svolta specificando opportunamente i parametri di offset RxOffset e RyOffset della struttura RasInfo al fine di selezionare una porzione della bitmap per la viewport. Lo scopo della struttura RasInfo è quello di selezionare una sotto-sezione di una bitmap estesa per definire la bitmap di viewport.

Si ricordi che la struttura RasInfo fornisce al sistema informazioni relative alla posizione della struttura BitMap e alla disposizione dell'area video. Queste

variabili di collocazione vengono misurate rispetto all'angolo superiore sinistro della bitmap in memoria, che possiede le coordinate 0,0; l'angolo inferiore destro possiede coordinate 1024,1024.

InitRastPort

Sintassi di chiamata della funzione

InitRastPort (rastPort)
A1

Scopo della funzione

Questa funzione inizializza con valori standard una struttura RastPort, la struttura principale per il controllo del disegno in una bitmap. La struttura RastPort descrive il modo di agire delle funzioni di disegno sui bit della bitmap a essa associata. Ogni funzione di disegno che svolga operazioni su una bitmap in RAM deve fare riferimento a questa struttura.

Quando InitRastPort restituisce il controllo, tutte le voci della struttura RastPort risultano azzerate, a eccezione dei parametri Mask, FgPen, AOlPen e LinePtrn che assumono il valore –1, del parametro DrawMode che viene impostato a JAM2 e della fonte-carattere che viene impostata al valore di default, cioè Topaz-60 o Topaz-80, a seconda di quanto impostato tramite Preferences.

Argomenti della funzione

rastPort Indirizzo della struttura RastPort da inizializzare.

Discussione

InitRastPort è la sola funzione grafica che riguarda direttamente la struttura RastPort. Essa consente d'inizializzare con valori standard alcuni parametri della struttura RastPort in memoria. Questa funzione dev'essere chiamata quando è necessario riportare una struttura RastPort alle sue impostazioni iniziali, tornando ai valori di default. Si veda anche l'introduzione del capitolo.

È possibile creare diverse strutture RastPort che possono essere poi usate per definire varie strutture ViewPort. Le strutture ViewPort possono essere collegate per definire una larga gamma di view distinte, a loro volta legate a una gamma estesa di schermi video.

La chiamata alla funzione InitRastPort stabilisce diversi valori di default nella struttura RastPort, ma non dice dove si trova la bitmap in memoria. Per eseguire operazioni grafiche che tengano conto dei valori memorizzati nella struttura RastPort, il task deve agganciarle una struttura BitMap opportunamente definita.

Si ricordi anche che la bitmap può arrivare a una grandezza di 1024 x 1024 pixel. La sua misura minima è di 1 x 1 pixel. Molto spesso, si definiscono bitmap di dimensioni standard, cioè pari alla risoluzione di schermo. Nello standard PAL le risoluzioni di schermo sono 320 x 256, 640 x 256, 320 x 512 o 640 x 512. Se si usa una di queste risoluzioni, la misura della viewport coincide esattamente con quella della bitmap. In tal caso non è possibile far scorrere la viewport all'interno della bitmap.

Tenendo presente queste considerazioni relative alla misura, vediamo quali sono i parametri chiave impiegati nelle varie strutture per fissare le relazioni tra le informazioni sulla bitmap e le informazioni sulla viewport e sulla view effettivamente visualizzate. Tali parametri sono Rows e BytesPerRow della struttura BitMap, RxOffest e RyOffset della struttura RasInfo, e DWidth, DHeight, DxOffset, DyOffset della struttura ViewPort.

Il parametro Rows è la dimensione verticale (misurata in pixel) della bitmap. Può variare tra un minimo di 1 e un massimo di 1024, ed è quello che viene usato nelle chiamate alle funzioni AllocRaster e InitBitMap. Se Rows contiene un valore maggiore di 512 è ovvio che non tutta la bitmap può essere visualizzata su un normale schermo, tanto meno se l'interlace non è attivato. Ciò nonostante, si tenga presente che si può continuare a disegnare anche sulle parti della bitmap che non sono in vista. Quindi, si può definire in memoria una bitmap molto estesa e presentarne solo una parte nel quadro inviato all'hardware video.

Il parametro BytesPerRow è la dimensione orizzontale (in byte) della bitmap. Questo parametro può contenere un valore compreso tra 1 e 128. Se BytesPerRow supera il valore 80, solo una parte delle informazioni contenute nella bitmap può essere presentata in un normale schermo.

RxOffset è il parametro che indica l'offset in direzione x, all'interno della bitmap; individua la coordinata x iniziale per la definizione della parte di bitmap che viene estratta per essere visualizzata nella viewport. RxOffset viene misurato dall'angolo superiore sinistro della bitmap conservata in memoria. Le coordinate dell'angolo superiore sinistro sono 0,0.

RyOffset è il parametro analogo per quanto riguarda l'offset verticale dall'angolo superiore sinistro della bitmap. Insieme con RxOffset, definisce nella bitmap le coordinate dell'angolo superiore sinistro della parte rettangolare di bitmap che dev'essere visualizzata nella viewport.

Il parametro DWidth indica la larghezza (in pixel) della parte di bitmap che dev'essere visualizzata nella viewport. Per esempio, se si vuole usare una viewport in bassa risoluzione, si può impostare DWidth a 320.

Il parametro DHeight indica l'altezza (in pixel) della parte di bitmap che

dev'essere visualizzata nella viewport. Per esempio, se si vuole usare una viewport all'interno di una view dotata di interlace, si può impostare DHeight a 512.

La variabile DxOffset imposta sullo schermo video la posizione iniziale, relativa alla coordinata x, della bitmap che costituisce la viewport, eventualmente prelevata da una bitmap più estesa. Qui le distanze vengono riferite allo schermo fisico dell'Amiga, non alla bitmap presente in RAM. Ciò pone il punto 0,0 nell'angolo superiore sinistro dello schermo video (inteso come rettangolo utile, cioè escluse le zone di overscan). Valori negativi (fino a -16) spostano la viewport a sinistra mentre valori positivi la spostano a destra.

La variabile DyOffset imposta sullo schermo video la posizione iniziale, relativa alla coordinata y, della bitmap che costituisce la viewport, eventualmente prelevata da una bitmap più estesa. Come per la variabile DxOffset, le distanze vengono riferite allo schermo fisico dell'Amiga, non alla bitmap presente in RAM. Valori negativi spostano la viewport verso l'alto mentre valori positivi la spostano verso il basso. Tutte queste relazioni sono illustrate nella Figura 2.4.

InitTmpRas

Sintassi di chiamata della funzione

```
InitTmpRas (tmpRas, buffer, size)
            A0      A1      D0
```

Scopo della funzione

Questa funzione predispone l'area di chip RAM indicata dal task per le funzioni di riempimento di aree e quelle destinate ai testi. Tali funzioni usano questa memoria per sezioni intermedie di bitmap in preparazione del loro inserimento nella bitmap finale.

L'argomento tmpRas dev'essere il puntatore alla struttura TmpRas che la funzione inizializza per gestire il buffer temporaneo. Dopo l'esecuzione della funzione, per rendere il buffer operativo il task deve agganciarlo alla struttura RastPort per cui l'ha predisposto. La struttura TmpRas è molto elementare; contiene semplicemente un puntatore in RAM a una bitmap temporanea, e la misura di questa bitmap.

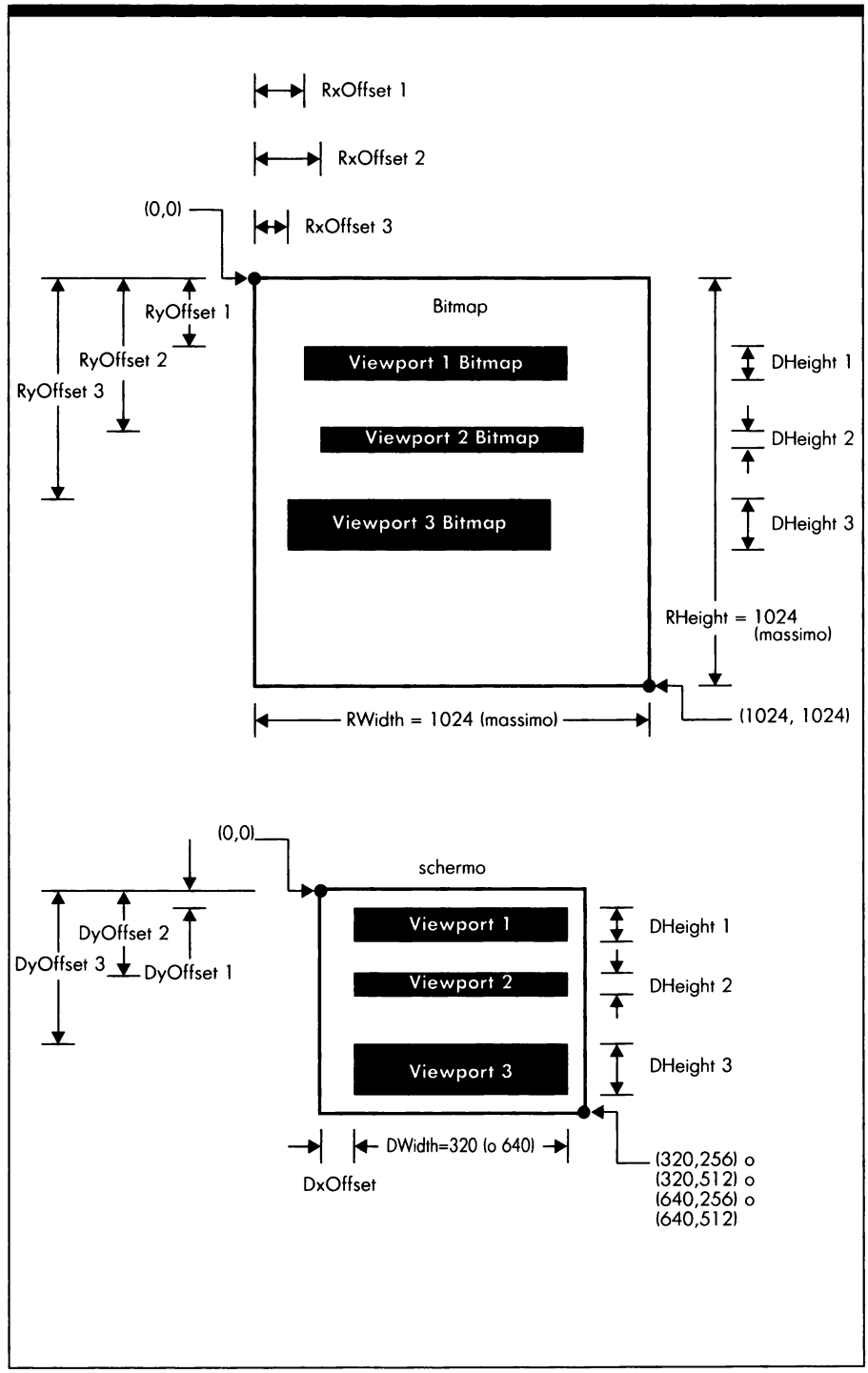


Figura 2.4: *Bitmap, viewport, offset nella view e dimensioni*

Argomenti della funzione

tmpRas Indirizzo della struttura TmpRas che dev'essere inizializzata e che successivamente il task dovrà legare a una struttura RastPort.

buffer Indirizzo del buffer temporaneo che il task deve aver predisposto nella chip RAM.

size Dimensione in byte del buffer.

Discussione

Nella libreria Graphics ci sono quattro funzioni che riguardano specificamente le bitmap: AllocRaster, FreeRaster, InitTmpRas e ScrollRaster. Si vedano anche le spiegazioni di queste funzioni.

Nelle funzioni di riempimento aree, oltre al buffer dei vettori il sistema di disegno richiede di predisporre anche buffer provvisori per sotto-sezioni temporanee della bitmap in elaborazione. Questo buffer viene utilizzato dalle funzioni AreaEnd e Flood, le quali funzionano nel modo seguente: quando vengono chiamate, effettuano i disegni nel buffer temporaneo individuato dalla struttura RastPort e generalmente creato dalla funzione InitTmpRas; poi, quando il buffer risulta pienamente definito, ne trasferiscono il contenuto nella bitmap destinazione sulla quale il task desidera effettuare il disegno.

Si può osservare che si tratta di un modo indiretto per riempire la bitmap con le informazioni grafiche. Una volta che la funzione AreaEnd è stata eseguita, il buffer viene automaticamente rilasciato per la successiva serie di chiamate alle funzioni AreaDraw, AreaMove e AreaEnd. A causa di questa procedura interna di sistema, si deve predisporre un buffer temporaneo di dimensioni opportune.

Si noti che questi buffer devono coesistere nella chip RAM. Questo significa che, per operazioni grafiche complesse, che interessino diverse bitmap, viewport e view, si dovrà pianificare un'opportuna strategia di lavoro per non dover chiedere al sistema troppa chip RAM.

Per allocare il buffer si può utilizzare la funzione AllocRaster, dal momento che il buffer viene trattato proprio come un bitplane. Per esempio, se si sa che in una bitmap verrà riempita un'area di 100 x 100 pixel, occorre allocare un bitplane, cioè un buffer, da 1400 byte: si chiama la funzione AllocRaster indicando il valore 100 nell'argomento width, che viene arrotondato a 112 (multiplo di 16) dalla funzione stessa, e indicando il valore 100 nell'argomento height. Procedendo in questo modo, l'indirizzo restituito da AllocRaster dev'essere poi usato come secondo argomento nella chiamata alla funzione InitTmpRas. Ovviamente, quando il buffer non sarà più necessario, il task dovrà provvedere autonomamente a liberare la memoria che occupa.

Si noti infine che lo stesso buffer temporaneo che il task associa alla

struttura RastPort può essere utilizzato per numerose operazioni di riempimento. Anziché allocarlo, disallocarlo e riallocarlo ogni volta, il task può continuare a usare sempre lo stesso, ma deve assicurarsi che sia delle dimensioni opportune per qualsiasi tipo di riempimento di aree possa essere richiesto con una particolare bitmap. Un tipico frammento di codice potrebbe essere questo:

```
struct TmpRas tmpRas;
struct RastPort *rastPort;
...
InitTmpRas (&tmpRas, AllocRaster (width, height), RASSIZE (width, height));
rastPort -> TmpRas = &tmpRas;
```

InitView

Sintassi di chiamata della funzione

```
InitView (view)
          A1
```

Scopo della funzione

Questa funzione inizializza la struttura View con un insieme di valori di default. Inizialmente i parametri della struttura View vengono impostati tutti a 0. Poi vengono assegnati a DxOffset e DyOffset i valori necessari per disporre lo schermo video di default circa 2,5 centimetri a destra e altrettanto in basso rispetto all'angolo superiore sinistro dell'Amiga (DxOffset viene a contenere il valore 0x81, mentre DyOffset il valore 0x2C).

Argomenti della funzione

view	Indirizzo della struttura View da inizializzare.

Discussione

Ci sono cinque funzioni della libreria Graphics che riguardano specificamente le viewport dello schermo video: InitView, InitVPort, LoadView, MakeVPort e ScrollVPort. Queste funzioni lavorano con due tipi di strutture:

ViewPort e View. InitView e LoadView utilizzano l'argomento view e quindi lavorano con la struttura View. InitVPort e ScrollVPort utilizzano l'argomento viewPort e lavorano perciò con la struttura ViewPort. MakeVPort impiega entrambi gli argomenti (view e viewPort) e lavora quindi con entrambe le strutture. Si veda anche l'introduzione del capitolo.

Questa funzione serve per disporre lo schermo circa 2,5 centimetri a destra e altrettanto in basso rispetto all'angolo superiore sinistro del monitor dell'Amiga, per assicurare che la view appaia nella porzione visibile dello schermo (disegnando nelle aree di overscan, infatti, non si ha la certezza che le informazioni video appaiano per intero su tutti i tipi di monitor).

I modi video di viewport

Ciascuna viewport può scegliere tra i sei diversi modi video disponibili. Questi modi vengono controllati attraverso sei bit (DUALPF, PFBA, HIRES, LACE EXTRA_HALFBRITE e HAM) del parametro Modes della struttura ViewPort. Segue una sintetica descrizione di ciascuno di questi bit.

Il bit DUALPF controlla il numero dei playfield che costituiscono lo schermo. Consente quindi di scegliere fra lo schermo single-playfield e quello dual-playfield. Se si sceglie lo schermo dual-playfield (flag DUALPF impostato a 1), lo schermo risulterà diviso in due playfield controllabili separatamente. Tale scelta modifica il modo in cui sono raggruppati i bitplane e il modo con cui vengono determinati i colori.

Il bit PFBA è il bit di priorità dello schermo dual-playfield. Questo bit va impiegato insieme al bit DUALPF per descrivere la configurazione dello schermo dual-playfield. Quando PFBA è impostato a 1 significa che il secondo playfield possiede una priorità video superiore a quella del primo, e quindi apparirà sullo schermo video davanti al primo playfield.

Il bit HIRES informa il sistema che la viewport descritta da questa struttura ViewPort verrà mostrata in modo alta-risoluzione con 640 pixel orizzontali. Se HIRES è a 0, verrà invece impiegato il modo video a bassa risoluzione con 320 pixel in ciascuna linea di schermo.

Il bit LACE informa il sistema che la viewport specificata dalla struttura ViewPort verrà visualizzata con l'interlace attivo. Essa possiederà quindi 512 linee per quadro video (sistema PAL). È importante notare che non è possibile alterare il flag LACE tra una viewport e l'altra all'interno di una data view; il flag LACE viene ignorato nella definizione della struttura ViewPort. La struttura View invece lo possiede. Quando la struttura View combina una o più viewport in un singolo schermo, l'attivazione dell'interlace varrà per tutte le viewport. I rimanenti flag di modo video (DUALPF, PFBA, HIRES, EXTRA_HALFBRITE e HAM) sono invece propri di ciascuna struttura ViewPort, e quindi possono cambiare passando da una viewport all'altra.

Il bit EXTRA_HALFBRITE indica al sistema di utilizzare per questa viewport (che dev'essere formata da 6 bitplane), il modo Extra Half-Brite a 64 colori. Il secondo gruppo di 32 colori non è altro che una copia del primo ma a luminosità dimezzata (su ognuno dei valori R, G, B viene eseguito lo shift di un bit a destra).

Quando il flag HAM è impostato a 1, il sistema sa che deve usare per lo schermo video il modo Hold And Modify. Il principale vantaggio di questo modo video consiste nel fatto di poter mostrare contemporaneamente 4096 colori. Per la definizione dei vari modi video si veda l'appendice B.

InitVPort

Sintassi di chiamata della funzione

InitVPort (viewPort)
AØ

Scopo della funzione

Questa funzione imposta la struttura ViewPort ai valori di default.

Argomenti della funzione

viewPort	Indirizzo della struttura ViewPort da inizializzare.

Discussione

Ci sono cinque funzioni nella libreria Graphics che riguardano specificamente le viewport: InitView, InitVPort, LoadView, MakeVPort e ScrollVPort. Si vedano anche le spiegazioni relative a queste funzioni.

Tutte le strutture View sono composte da una o più strutture ViewPort. Prima che queste ultime possano essere aggiunte al sistema, devono essere opportunamente inizializzate. Questo è lo scopo della funzione InitVPort. Dopo che questa funzione è stata eseguita, il task può inizializzare i parametri della struttura ViewPort secondo le proprie esigenze. La funzione ne inizializza alcuni con valori di default, ma il task ha comunque la facoltà di variare anche quei parametri, se lo ritiene opportuno. In particolare, la funzione imposta a 0 il parametro Modes, in modo che per default la viewport sia in bassa risoluzione.

Le restrizioni che riguardano le viewport

Vi sono diverse restrizioni riguardanti le viewport e le strutture ViewPort a esse associate. Primo, le viewport possono essere soltanto sezioni orizzontali dello schermo: non è possibile definire due viewport che appaiano sullo schermo una di fianco all'altra. Se si vogliono definire finestre sovrapposte e affiancate, bisogna usare una sola viewport disposta all'interno di una view e creare le finestre all'interno di tale viewport utilizzando i layer. Questo è sostanzialmente il modo in cui le funzioni della libreria Intuition creano e gestiscono le finestre.

Secondo, le viewport non possono sovrapporsi in alcun modo sullo schermo video. Ciò comprende non solo la sovrapposizione orizzontale, ma anche quella verticale. Tale restrizione è sottintesa dalle altre.

Terzo, le viewport non possono essere esattamente adiacenti tra loro: devono risultare separate almeno da una linea di schermo. Ciò è richiesto per lasciare al sistema il tempo di cambiare le impostazioni video (modi, colori...) mentre il pennello elettronico si appresta a disegnare la successiva viewport. In alcuni casi è necessaria più di una linea di separazione, per esempio quando nel passaggio da una viewport alla successiva viene effettuato un cambio di palette.

La definizione della viewport

Per definire una viewport, dobbiamo stabilirne l'altezza, la larghezza, la profondità (numero di bitplane) e il modo video, e memorizzare queste informazioni negli opportuni parametri della struttura ViewPort che la definisce. L'altezza della viewport è il numero di linee orizzontali di schermo che la viewport è destinata a occupare nel momento in cui viene visualizzata, e dev'essere memorizzata nel parametro DHeight della struttura ViewPort. Spesso questo valore viene fissato a 256 o 512 (sistema PAL), per definire una view nella quale è presente una sola viewport che occupa l'intero spazio disponibile rispettivamente in modo senza interlace e con interlace. Le dimensioni della viewport possono essere anche maggiori, ma con alcuni monitor può accadere che non tutta la parte che cade nella zona di overscan sia totalmente visibile.

La larghezza della viewport è il numero dei pixel orizzontali che tale viewport è destinata a occupare nel momento in cui viene visualizzata, e dev'essere memorizzata nel parametro DWidth della struttura ViewPort. Spesso questo valore viene fissato a 320 o 640, sebbene possa essere un qualunque numero intermedio. Non potrà essere tuttavia maggiore di 640, a meno di utilizzare la zona di overscan.

La profondità della viewport indica quanti bitplane vengono utilizzati per definire la bitmap della viewport. Si tratta del parametro Depth, presente nella struttura BitMap impiegata per definire la bitmap della viewport. Tale numero può variare tra 1 e 6, a seconda del numero di colori che devono essere presenti nella viewport. Per esempio, per ottenere 32 colori, quando si chiama la funzione InitBitMap si deve specificare una profondità di 5; sono infatti

necessari cinque bitplane per ogni bitmap.

Il modo video impiegato per la viewport in questione viene controllato da sei flag della struttura ViewPort: DUALPF, PFBA, HIRES, LACE, EXTRA_HALFBRITE e HAM, che informano l'hardware su come dev'essere visualizzata la viewport quando viene inserita in una definizione di view. Le modalità video di viewport sono trattate nella spiegazione della funzione InitView.

Si noti che il flag LACE viene ignorato per tutte le viewport; questo flag viene preso in considerazione solo nella struttura View che guida la presentazione di tutte le viewport in una data view. Per questa ragione, all'interno di una view in interlace tutte le viewport in essa definite appaiono in interlace. Non è possibile nessun cambiamento sull'interlace tra una viewport e l'altra se appartengono alla stessa view di schermo.

LoadRGB4

Sintassi di chiamata della funzione

LoadRGB4 (viewPort, colorTable, count)
A0 A1 D0

Scopo della funzione

Questa funzione assegna un insieme di valori di registri di colore RGB a una determinata struttura ViewPort. Tali colori (rosso, verde e blu) devono essere contenuti nella tavola dei colori (un array costituito da word) che il task indica come secondo argomento.

Argomenti della funzione

viewPort — Indirizzo della struttura ViewPort in cui si vuole alterare la definizione dei registri colore.

colorTable — Indirizzo della tavola dei colori (un array costituito da word) contenente le definizioni dei registri colore.

count — Numero delle word da caricare dalla tavola dei colori. Si inizia con il colore 0 (colore di fondo) e si procede verso il colore di numero progressivo più alto

specificato nell'array della tavola dei colori. Tramite questo parametro si può selezionare un sottoinsieme da una tavolozza più grande.

Discussione

Ci sono quattro funzioni della libreria Graphics che lavorano in particolar modo con i colori assegnati alle viewport: GetRGB4, LoadRGB4, SetRGB4 e SetRGB4CM. LoadRGB4 carica un insieme di valori RGB nella definizione di una viewport, prelevandoli dall'array di colori indicato dal task.

GetColorMap alloca la memoria necessaria per una struttura ColorMap e la imposta; questa funzione richiede come argomento il numero degli elementi presenti nella tavola dei colori. Si tratta di un numero compreso tra 1 e 32, che dipende da quanti registri colore verranno definiti o ridefiniti attraverso la successiva chiamata alla funzione LoadRGB4.

LoadRGB4 carica i colori definiti nell'array nella tavola dei colori associata alla viewport indicata. Alcuni esempi dei colori che possono essere collocati nei 32 registri sono riportati nella trattazione della funzione SetRGB4.

Si ricordi che l'Amiga possiede 32 registri hardware per il controllo del colore. È interessante notare che ciascuno di questi 32 registri ha la capacità di definire un colore da una scelta di 4096. Ciò è possibile perché tali registri includono quattro bit (i bit da 11 a 8) per il rosso, quattro bit (i bit da 7 a 4) per il verde e quattro bit (i bit da 3 a 0) per il blu. Questo significa che sono a disposizione 16 variazioni per ciascuno dei colori primari (rosso, blu, verde) per un totale di 4096 colori, qualora si combinino tutte le sfumature di ciascun colore.

Per quanto riguarda l'uso della funzione LoadRGB4, è necessario cominciare con il definire in RAM un array di colori. Da questo array la funzione preleva soltanto il numero di colori indicato come terzo argomento della chiamata alla funzione LoadRGB4. Ciascun elemento dev'essere composto da due byte. Il significato di questi 16 bit è il seguente:

0000 rrrr gggg bbbb (0000 rosso verde blu)

Qui, 0000 rappresenta i quattro bit più significativi di un registro colore; tali bit non vengono utilizzati e sono sempre impostati a zero; rrrr rappresenta i quattro bit assegnati alle variazioni d'intensità del rosso; gggg rappresenta i quattro bit assegnati alle variazioni d'intensità del verde; bbbb rappresenta i bit assegnati alle variazioni d'intensità del blu. Per ogni colore fondamentale, i valori che possiamo stabilire devono variare da 0000 (assenza di colore) a 1111 (colore alla massima intensità).

È importante rendersi conto che possono coesistere in memoria, nello stesso momento, parecchi di questi array di colori. Per qualsiasi viewport si possono creare tanti array quanti ne occorrono. Ciascuno di essi viene predisposto con una chiamata alla funzione GetColorMap che assegna la memoria, facendo seguire una chiamata a LoadRGB4 per ottenere il

riempimento della tavola dei colori. Quando si vogliono alterare i colori impiegati in una viewport, si chiama LoadRGB4 utilizzando un diverso argomento per puntare a un altro (già definito) array di colori.

Per esempio, è possibile disporre di una view composta dalla stessa viewport disegnata due volte nello stesso quadro video. Esisterà almeno una linea di scorrimento del raggio di scansione video tra queste due viewport. La prima impiega un puntatore a un array della tavola dei colori. La seconda è identica alla prima, ma utilizza un puntatore a un secondo array della tavola dei colori. In tal modo è possibile presentare sullo stesso schermo due copie delle stesse informazioni, ma con colori diversi.

LoadView

Sintassi di chiamata della funzione

LoadView (view)
A1

Scopo della funzione

Questa funzione impiega una lista di istruzioni Copper per creare la view di schermo video corrente. Questa lista è stata precedentemente prodotta dalle funzioni InitVPort, MakeVPort e MrgCop. Quando LoadView restituisce il controllo, la nuova view appare sullo schermo video. Considerato che è possibile mostrare sullo schermo solo una view alla volta, la nuova view apparsa è l'unica attiva.

Argomenti della funzione

view	Indirizzo della struttura View che definisce la view da visualizzare.

Discussione

Nella libreria Graphics vi sono cinque funzioni che lavorano in particolar modo con le viewport e con le view: InitView, InitVPort, LoadView, MakeVPort e ScrollVPort. Si vedano anche le spiegazioni di queste funzioni.

La funzione LoadView lavora direttamente con la lista di istruzioni hardware del coprocessore Copper. Essa genera l'insieme delle istruzioni Copper necessarie per definire un quadro video, basato soltanto sulle istruzioni della struttura View. La struttura View è stata sviluppata da altre chiamate di allestimento, prima di giungere a questo punto dell'elaborazione. L'insieme finale di istruzioni Copper può includere istruzioni aggiuntive definite da due altre sorgenti: le funzioni di animazione della libreria Graphics e le istruzioni Copper definite dal programmatore nella struttura UCopList.

Le istruzioni Copper provenienti da queste tre sorgenti devono essere unite prima che venga chiamata la funzione LoadView. Tale operazione di unione costituisce il lavoro della funzione MrgCop. Sostanzialmente la funzione LoadView impiega l'insieme completo delle istruzioni Copper fornite dalla funzione MrgCop, comprese quelle che essa stessa ha creato per la view, per presentare sul video il risultato finale.

Ci sono due macro-istruzioni di sistema che possono controllare gli eventi video che precedono e seguono la chiamata alla funzione LoadView: ON_DISPLAY e OFF_DISPLAY. Esse lavorano insieme con la funzione LoadView per determinare esattamente cosa accade quando viene eseguita la funzione LoadView.

Queste macro possono essere chiamate appena prima e appena dopo che è stata effettuata la funzione LoadView. Se si chiama ON_DISPLAY appena prima di LoadView, lo schermo video verrà aggiornato immediatamente dopo l'esecuzione della funzione LoadView. Se si chiama OFF_DISPLAY appena prima di LoadView, lo schermo video non verrà aggiornato immediatamente dopo l'esecuzione della funzione LoadView e il nuovo quadro video non verrà mostrato. Infatti questa nuova view non verrà presentata fino all'esecuzione della macro ON_DISPLAY. ON_DISPLAY costituisce la situazione di default del sistema; il video viene automaticamente aggiornato dopo una chiamata a LoadView, a meno che non si chiami preventivamente la macro OFF_DISPLAY.

L'impiego di queste macro consente di sopprimere temporaneamente la view. Il maggior vantaggio di questa operazione consiste nella soppressione della possibile confusione dello schermo video (resti di dati inutili) dovuta alle fasi intermedie che intervengono nella definizione grafica di una view. Per evitare questo effetto di "scarto" conviene eseguire il disegno con OFF_DISPLAY attivo; poi, una volta che le funzioni di disegno avranno completato la loro esecuzione, si chiama la macro ON_DISPLAY. Ciò riporta lo schermo video alle normali condizioni operative e mostra la view creata dalla più recente chiamata alla funzione LoadView.

LockLayerRom

Sintassi di chiamata della funzione

LockLayerRom (layer)
A5

Scopo della funzione

Questa funzione assegna un lock una struttura Layer impiegando codice ROM nella libreria Graphics. Quando la funzione LockLayerRom restituisce il controllo, il layer risulta bloccato per essere destinato all'uso esclusivo del task che ha effettuato la chiamata. Nessun altro task può modificare questo layer fino a che il task che lo detiene non esegue una chiamata alla funzione UnlockLayerRom. La chiamata alla funzione LockLayerRom non distrugge il contenuto di nessun registro.

Argomenti della funzione

layer	Indirizzo della struttura Layer da bloccare.

Discussione

Ci sono tre funzioni video della libreria Graphics che riguardano in particolar modo la struttura Layer: AttemptLockLayerRom LockLayerRom e UnlockLayerRom. Queste funzioni hanno lo scopo di bloccare e sbloccare i layer di schermo per impedire o consentire ai task di effettuare operazioni di disegno al loro interno. Con queste funzioni si possono evitare alterazioni indesiderate delle informazioni grafiche di una bitmap di layer da parte di altri task.

Tutte queste funzioni possiedono un solo argomento: un puntatore a una struttura Layer. Il valore di questa variabile è noto dalla precedente chiamata alle funzioni CreateUpfrontLayer o CreateBehindLayer (si veda il capitolo 5).

Il sistema Amiga può agire con quattro tipi di layer: il layer a refresh semplice, il layer a refresh avanzato, il layer di superbitmap e il layer backdrop. Per quanto riguarda i layer di superbitmap si vedano le spiegazioni delle funzioni CopySBitMap e SyncSBitMap; per approfondire le conoscenze sulla struttura Layer si veda l'introduzione al capitolo 5.

Si supponga di disporre di un task grafico nell'ambito del quale si vogliono

creare bitmap di layer nelle quali verrà effettuato il disegno. Si supponga che il task grafico non sia l'unico task nel sistema ma che sia invece parte di un più esteso programma grafico in corso di creazione. Tale programma produrrà e definirà diversi layer e sarà composto da diversi task che interagiranno tra loro. Ciascuno di questi task può modificare il contenuto della bitmap di qualsiasi layer, quando prende il controllo del sistema.

Si supponga che un task si occupi del disegno in un layer specifico. Per impedire che gli altri task del programma possano alterare il layer che il task sta definendo, bisogna fare in modo che solo questo task possa cambiare la bitmap di layer fino a quando non si informi il sistema di procedere altrimenti. È necessario agire in questo modo perché non si conosce la sequenza esatta delle azioni degli altri task, che potrebbero comportare una sequenza di disegno nel layer in questione. Lo si può impedire mandando in esecuzione la funzione LockLayerRom. Solo quando il primo task esegue la funzione UnlockLayerRom, gli altri task potranno modificare la definizione di bitmap di questo layer.

Le chiamate alla funzione LockLayerRom possono essere nidificate. Si immagini di avere tre task grafici che agiscono all'interno di un programma grafico. Il primo task crea un layer chiamando la funzione CreateUpfrontLayer della libreria Layers. L'operazione assegna automaticamente una bitmap al layer, e vi associa inoltre una struttura RastPort per il controllo del disegno. Si possono usare alcune funzioni di disegno (per esempio AreaDraw, AreaMove e AreaEnd) per inserire informazioni di disegno in questa bitmap di layer.

Una volta che il disegno è finito, si decide di cambiare task per passare al secondo dei tre task grafici esistenti nel programma. Tuttavia, prima di effettuare lo scambio, si decide che agli altri task non debba essere concesso di disegnare all'interno del primo layer. Prima di abbandonare il primo task si esegue perciò la funzione LockLayerRom per bloccare il layer in questione.

Quando il secondo task grafico del programma diventa attivo e prende il controllo della macchina, esso crea un secondo layer e disegna al suo interno. Poi, prima di trasferire il controllo al terzo task, esegue a sua volta una chiamata alla funzione LockLayerRom per il secondo layer. A questo punto soltanto il secondo task potrà modificare il secondo layer.

Si può continuare a procedere in questo modo. In alternativa si può ritornare all'interno di ciascuno dei tre task ed eseguire la chiamata alla funzione UnlockLayerRom.

In tal modo si possono controllare con precisione i cambiamenti nelle bitmap di layer, bloccando e sbloccando i layer come si ritiene necessario.

MakeVPort

Sintassi di chiamata della funzione

MakeVPort (view, viewPort)
A0 A1

Scopo della funzione

Questa funzione impiega le informazioni contenute nelle strutture View e ViewPort per generare una lista intermedia Copper per una viewport.

Se il puntatore alla struttura ColorMap nella struttura ViewPort è nullo, MakeVPort utilizza una palette di default. Se il flag DUALPF del parametro Modes nella struttura View risulta impostato, ci dev'essere una seconda struttura RasInfo puntata dalla prima.

Argomenti della funzione

view	Indirizzo della struttura View.
viewPort	Indirizzo di una struttura ViewPort che contiene un puntatore a una struttura RasInfo.

Discussione

Ci sono cinque funzioni nella libreria Graphics dell'Amiga che riguardano specificamente le viewport dello schermo video e le view: InitView, InitVPort, LoadView, MakeVPort e ScrollVPort. Si vedano anche le spiegazioni di queste funzioni.

Le informazioni video richieste per definire un quadro vengono realizzate seguendo alcuni passi ben definiti. Ciò include i passi per allocare memoria per bitmap e strutture, e per definire le strutture ViewPort e View. Tutti questi passi portano alla completa definizione di una struttura View e della view di schermo a essa associata, che costituisce un quadro di una sequenza video.

Si ricordi che le funzioni video devono, alla fine, trattare direttamente con i chip hardware correlati al video. Il sistema ha bisogno di un insieme di istruzioni per guidare l'hardware; in particolare necessita di un insieme di istruzioni per guidare il coprocessore Copper. Il Copper informa gli altri chip

correlati al video di quali devono essere i colori e le altre caratteristiche del quadro video. Questo è il momento in cui entra in gioco la funzione MakeVPort.

La funzione MakeVPort controlla la struttura View in memoria e genera un insieme di istruzioni Copper che servono per produrre la view sullo schermo video dell'Amiga. Queste istruzioni Copper vengono poi unite alle istruzioni Copper di animazione sprite (se ci sono) e alle istruzioni Copper definite dal programmatore (quelle presenti nella struttura UCopList, se ce ne sono) per giungere all'insieme finale di istruzioni Copper per il quadro video. Se lo schermo non cambia tra un quadro e l'altro, questo insieme d'istruzioni Copper può essere utilizzato ripetutamente (ogni cinquantesimo di secondo) per guidare l'hardware video.

Se invece il task vuole alterare la presentazione dello schermo tra un quadro e l'altro, è necessario cambiare le istruzioni Copper con le quali agisce la funzione MakeVPort. Mentre il pennello elettronico è occupato a percorrere lo schermo dal basso all'alto (intervallo di vertical-blanking), le istruzioni Copper vengono aggiornate per definire un nuovo quadro. In tal modo si può modificare la presentazione video in un tempo ridottissimo.

Si ricordi sempre che è la funzione LoadView che carica effettivamente la view totale nell'hardware, cioè che la visualizza. Se si dispone di ulteriori istruzioni Copper, queste ultime dovranno essere unite alle istruzioni Copper della struttura View appena prima di effettuare la chiamata alla funzione LoadView. La funzione MrgCop va usata per unire tutte e tre le possibili sorgenti di istruzioni Copper.

Questi sono i principi generali che mettono il task in relazione con i cambiamenti dello schermo video. Si vedano anche le spiegazioni delle macro CINIT, CMOVE, CWAIT e CEND.

Move

Sintassi di chiamata della funzione

```
Move (rastPort,     x,     y)
      A1            D0     D1
```

Scopo della funzione

Questa funzione muove la penna grafica alle coordinate indicate. Le coordinate sono relative all'angolo superiore sinistro del sistema di coordinate della bitmap (cioè il punto 0,0).

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
x	Coordinata orizzontale nel sistema di coordinate della bitmap.
y	Coordinata verticale nel sistema di coordinate della bitmap.

Discussione

Nella libreria Graphics ci sono cinque funzioni di disegno che riguardano specificamente l'uso delle penne: Move, SetAPen, SetBPen, SetOPen e SetRast. Queste funzioni consentono di muovere e di manipolare le penne di disegno per definire i colori e le altre caratteristiche della bitmap.

La funzione Move sposta la penna verso una nuova posizione nella bitmap. Il movimento ha luogo all'interno del sistema di coordinate della bitmap. Si tratta della bitmap associata alla struttura RastPort di controllo che il task ha indicato come argomento. Spesso, insieme alla funzione Draw, per definire linee all'interno di una bitmap viene impiegata la funzione Move. Esiste inoltre la funzione PolyDraw, che fornisce un modo per inserire insiemi di linee in una bitmap. Queste funzioni, insieme alle funzioni AreaEnd e RectFill che disegnano linee di contorno su aree che provvedono a riempire, forniscono tutte le possibilità di disegno di linee all'interno di una bitmap.

La funzione Move differisce dalle altre per un aspetto molto importante: non fa apparire niente sulla bitmap; essa sposta la penna "sollevandola" sulla superficie di disegno costituita dalla bitmap, per posarla poi in un'altra posizione della bitmap che si sta definendo. In tal modo si può concludere il tracciamento di una linea effettuato con la funzione Draw, trasportare la penna in un'altra posizione x,y e quindi riprendere a disegnare con le funzioni Draw, AreaDraw, AreaMove, PolyDraw, RectFill o Text.

Proprio come per le altre funzioni di disegno, qualsiasi movimento venga effettuato, deve rimanere all'interno dei confini della bitmap in uso. Se si specifica una posizione x,y al di fuori del sistema di coordinate della bitmap, si può causare il crash del sistema. I valori massimi delle coordinate sono 1024,1024, se è stata predisposta una bitmap di queste dimensioni.

Sebbene la funzione Move non imposti né azzeri i bit della bitmap, questi bit saranno interessati da una successiva chiamata a qualsiasi altra funzione di disegno.

MrgCop

Sintassi di chiamata della funzione

MrgCop (view)
A1

Scopo della funzione

Questa funzione unisce le istruzioni intermedie video Copper, di colore, di sprite e definite dal programmatore, in un unico insieme ordinato di istruzioni. In pratica, MrgCop prepara per il coprocessore Copper un programma finalizzato alla produzione del quadro video. Questa funzione viene utilizzata automaticamente dalle routine di animazione grafica, che in pratica aggiungono informazioni su uno sfondo statico. Tale procedura automatica altera alcune delle istruzioni sprite o definite dal programmatore, ma non quelle che costituiscono la presentazione di base. Quando tutti i tipi di istruzioni di coprocessore sono stati fusi insieme, il task dispone di una lista completa di istruzioni per la generazione del quadro, pronte per essere sottoposte al coprocessore Copper.

Argomenti della funzione

view	Indirizzo della struttura View nella quale le istruzioni Copper devono essere miscelate.

Discussione

Ci sono otto macro e funzioni video che appartengono alla libreria Graphics e riguardano in modo particolare il coprocessore Copper: CINIT, CMOVE, CWAIT, CEND, FreeCopList, FreeCprList, FreeVPortCopLists e MrgCop. Si vedano anche le spiegazioni di queste macro e funzioni.

L'obiettivo di un lavoro grafico è produrre una serie di view da presentare al sistema hardware affinché le visualizzi sul video dell'Amiga. Per ottenere queste view, una serie di bitmap dev'essere inizializzata, le viewport devono essere definite come parti delle view e poi combinate insieme nelle view e, infine, si devono eseguire i disegni nelle bitmap. A questo punto il programma può trasferire queste informazioni all'hardware e presentare una serie di view.

Per esempio, in un programma guidato da menu, ciascun nuovo menu selezionato può richiamare un diverso schermo video. In background il programma rileva la selezione effettuata e carica un'altra view impiegando la funzione LoadView.

Quando ciò accade, molti registri di controllo del sistema hardware devono essere reimpostati ai loro nuovi valori. In particolare, se la palette della nuova view è diversa da quella del precedente schermo, devono essere modificati i registri colore. Perciò, la lista di istruzioni Copper, costituita da istruzioni video per il Copper, dev'essere ridefinita per produrre lo schermo video associato con la nuova selezione di menu.

Prima di effettuare la chiamata alla funzione LoadView, bisogna quindi definire la provenienza delle istruzioni Copper necessarie alla view. Si tratterà generalmente di tre fonti: le istruzioni Copper generate internamente e associate con la particolare view appena definita, le istruzioni Copper provenienti dalle funzioni di animazione che definiscono l'inserimento degli sprite nei playfield e nelle view, l'insieme delle istruzioni Copper definite dal programmatore con le chiamate alle macro CMOVE, CWAIT e CEND.

La funzione MrgCop svolge questo lavoro radunando insieme tutte queste istruzioni Copper e ordinandole per produrre una lista valida per l'intero quadro video, in modo che risulti definita la presentazione della view. È necessario che tutte le liste intermedie presenti nelle viewport associate alla view, siano già correttamente ordinate in ordine y,x crescente.

NewRegion

Sintassi di chiamata della funzione

```
region = NewRegion ()
DØ
```

Scopo della funzione

Questa funzione crea una struttura Region che definisce una regione nulla. La struttura impostata è vuota; ciò significa che non vi sono rettangoli assegnati alla regione. NewRegion restituisce l'indirizzo della nuova struttura Region.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Nella libreria Graphics ci sono sei funzioni che riguardano specificamente le regioni e i rettangoli di delimitazione che le costituiscono: AndRectRegion, ClearRegion, DisposeRegion, NewRegion, OrRectRegion e XorRectRegion. Tali funzioni consentono di costruire una regione da un insieme di rettangoli di delimitazione e modificare poi le caratteristiche della regione. Si vedano anche le spiegazioni di queste funzioni.

Per comprendere a che cosa servono le regioni si devono capire innanzi tutto due strumenti di cui si serve la libreria Graphics: i rettangoli di delimitazione (clipping-rectangle) e le liste di danneggiamento (damage-list). Questi due elementi giocano un ruolo vitale nella definizione e nell'uso delle regioni. Essi entrano in gioco in modo rilevante anche nella definizione e nell'uso dei layer.

Un *rettangolo di delimitazione* è un rettangolo che delimita l'area in cui le funzioni di disegno (per esempio AreaDraw, AreaMove e AreaEnd) sono abilitate a disegnare. La composizione di un insieme di tali rettangoli di delimitazione definisce l'area totale della bitmap di layer alla quale le funzioni di disegno possono accedere. I rettangoli di delimitazione prendono il nome dal tipo di azione che svolgono: limitano le operazioni di disegno all'interno dei loro contorni.

Per esempio, se una funzione di disegno (come Draw, Move o PolyDraw) richiede l'aggiornamento del layer, soltanto le parti di layer che si trovano all'interno dei rettangoli di delimitazione riceveranno una nuova definizione dei bit di bitmap. In altre parole, tutte le operazioni di disegno del layer rimarranno limitate semplicemente all'interno delle aree definite dall'insieme di rettangoli di delimitazione.

Per rendere più comprensibile questo concetto, si consideri uno schermo di tipo Intuition, multi-finestra e quindi multi-layer. Si supponga di avervi aperto due finestre. Chiamiamo queste finestre 1 e 2. Si immagini che la finestra 2 sia stata aperta dopo la finestra 1. Al momento, la finestra 2 è quella attiva.

Inoltre si supponga che queste due finestre non si sovrapponessero quando sono state aperte; esse rimangono quindi separate dallo schermo di sfondo che si trova attorno ai loro bordi. Ora supponiamo di trascinare la finestra 2 fino a sovrapporla alla finestra 1. Nel far ciò, la finestra 1 viene parzialmente coperta: una sua porzione rettangolare viene nascosta dalla finestra 2.

Ora si sposti la finestra 2 di nuovo lontana dalla 1. Automaticamente, le routine della libreria Intuition ripristinano l'angolo della finestra 1 precedentemente nascosto dalla finestra 2. Nel corso dell'operazione di spostamento della finestra, la finestra 1 viene parzialmente reintegrata: viene ricostruito solo l'angolo nascosto, non l'intera finestra.

La reintegrazione dell'angolo nascosto viene effettuata ricorrendo ai rettangoli di delimitazione e alle liste di danneggiamento, definiti rispettivamente dalle strutture ClipRect e DamageList.

In tal caso è interessato solo un rettangolo di delimitazione. Esso rappresenta il rettangolo di delimitazione rimasto celato nell'angolo della finestra 1. Si noti che non si tratta di un rettangolo visibile. In realtà esiste solo

in astratto, dal momento che non sarà mai visibile nessuno dei suoi bordi, né ora, né in qualsiasi successivo momento della presentazione video: si limita a circondare l'area nascosta sullo schermo. Tale area viene detta *danneggiata*; essa viene ricostruita quando le finestre vengono nuovamente separate.

Ogni volta che le finestre Intuition vengono spostate o disposte su piani diversi (quando vengono portate davanti o dietro alle altre finestre) o viene alterata la loro forma, la lista di danneggiamento del layer e le definizioni dell'insieme dei rettangoli di delimitazione associati cambiano per rappresentare le porzioni nascoste.

Quando le finestre vengono nuovamente spostate, oppure disposte su di un altro piano, le definizioni dei confini dell'insieme dei rettangoli di delimitazione e la lista di danneggiamento associata vengono impiegati per definire le aree da aggiornare. Nel caso ora descritto, soltanto le aree danneggiate verranno ricostruite. Grazie a questa caratteristica, l'aggiornamento di schermo è molto rapido, sia per gli schermi Intuition sia gli schermi creati dai programmi grafici.

La lista di danneggiamento e i rettangoli di delimitazione possono presentarsi almeno in due diversi casi. Il primo caso si verifica quando l'utente di un programma grafico sposta elementi grafici su uno schermo video multi-layer. Il veloce schermo Intuition, con le finestre a sovrapposizione multipla, costituisce l'esempio più comune di questo tipo di comportamento dinamico della lista di danneggiamento e dei rettangoli di delimitazione.

Nel secondo caso, la lista di danneggiamento e i rettangoli di delimitazione possono essere progettati appositamente per definire e controllare il disegno in una specifica sezione (o nell'intero schermo) di una bitmap di layer grafico. Questa è la fase in cui entrano in gioco regioni e rettangoli di delimitazione. Le *regioni* sono insiemi di rettangoli di delimitazione che, quando vengono uniti insieme, diventano parte di una lista di danneggiamento personalizzata che viene impiegata per controllare il disegno in una bitmap di layer. Le funzioni di regione della libreria Graphics consentono di progettare e costruire una struttura personalizzata DamageList che contenga puntatori a un insieme di strutture Rectangle per gestire la definizione di una regione. Con questa lista si può aggiornare selettivamente parte di una specifica bitmap di layer senza coinvolgere nessun'altra bitmap di layer relativa allo schermo video in uso.

L'idea è quella d'impiegare NewRegion per assegnare memoria a una nuova struttura Region. Questa struttura non contiene informazioni sulla grandezza finale della regione: in questa fase la regione non contiene rettangoli di delimitazione ed è praticamente vuota. Soltanto quando si usano le funzioni AndRectRegion, OrRectRegion e XorRectRegion vengono definiti i rettangoli di delimitazione associati a questa regione. Quando ciò avviene, si cominciano a definire le caratteristiche del disegno in tale regione; essa si amplia e si riduce mentre si usano le tre citate funzioni per combinare rettangoli di delimitazione e definire la sua misura finale.

Si veda anche il capitolo 5.

OFF_DISPLAY

Sintassi di chiamata della macro

OFF_DISPLAY

Scopo della macro

Questa macro azzera il bit BPLEN presente nel registro di controllo del DMA. Ogni view caricata in seguito con la funzione LoadView non verrà mostrata finché non viene eseguita la macro ON_DISPLAY.

Argomenti della macro

Questa macro non ha argomenti.

Discussione

Ci sono due macro che riguardano il controllo DMA del video: ON_DISPLAY e OFF_DISPLAY. Si veda anche la spiegazione della macro ON_DISPLAY. OFF_DISPLAY azzera il bit BPLEN nel registro DMACON. Vi sono diverse situazioni in cui può essere utile la macro OFF_DISPLAY. Per esempio quando si sta disegnando in una bitmap e non si vuole che l'utente veda le fasi intermedie di una sequenza di disegno; OFF_DISPLAY impedisce la presentazione della view sulla quale viene eseguito il lavoro. Un altro caso è la necessità di fornire cicli addizionali di memoria alla CPU 68000 o al Blitter. OFF_DISPLAY interrompe il DMA video, accelerando conseguentemente altre operazioni DMA.

Il sistema DMA dell'Amiga, comunque, fornisce quattro canali hardware per il suono, un canale per le operazioni di lettura e scrittura su disco e uno per la bassa risoluzione a 16 colori (oppure per il modo video a quattro colori in alta risoluzione), in grado di operare tutti contemporaneamente e senza rallentare le operazioni della CPU 68000. Per questo OFF_DISPLAY in genere non viene utilizzato per accelerare le altre operazioni DMA.

OFF_VBLANK

Sintassi di chiamata della macro

OFF_VBLANK

Scopo della macro

Questa macro azzera il bit di abilitazione dell'interrupt di vertical-blanking (il bit numero 5 chiamato VERTB) nel registro di controllo degli interrupt INTENA.

Argomenti della macro

Questa macro non ha argomenti.

Discussione

Ci sono due macro che riguardano il controllo degli interrupt nel sistema grafico: ON_VBLANK e OFF_VBLANK. Si veda anche la spiegazione della macro ON_VBLANK, che costituisce il complemento della macro OFF_VBLANK.

La disabilitazione dell'interrupt di vertical blanking fa sì che l'aggiornamento delle informazioni presenti sul quadro venga sospeso fino a quando non viene eseguita la macro ON_VBLANK.

ON_DISPLAY

Sintassi di chiamata della macro

ON_DISPLAY

Scopo della macro

Questa macro imposta il bit BPLEN presente nel registro di controllo del DMA, riattivando quindi l'aggiornamento del video effettuato da questo canale DMA. Da questo momento in poi verrà perciò mostrata qualsiasi view caricata con la funzione LoadView.

Argomenti della macro

Questa macro non ha argomenti.

Discussione

Ci sono due macro che riguardano il controllo DMA del video: ON_DISPLAY e OFF_DISPLAY. Si veda anche la spiegazione della macro OFF_DISPLAY.

ON_DISPLAY imposta a 1 il bit 8 (chiamato bit BPLEN) nel registro di controllo DMA (il registro denominato DMACON). Dopo aver caricato una view con la funzione LoadView, si può utilizzare la macro ON_DISPLAY per consentire al sistema DMA di presentarla sullo schermo.

ON_DISPLAY è attiva per default; quindi, se il DMA video non viene disattivato con una chiamata alla macro OFF_DISPLAY, ogni view caricata viene automaticamente presentata sullo schermo.

ON_VBLANK

Sintassi di chiamata della macro

ON_VBLANK

Scopo della macro

Questa macro imposta il bit di abilitazione dell'interrupt di vertical blanking (il bit numero 5 chiamato VERTB) nel registro di controllo degli interrupt INTENA.

Argomenti della macro

Questa macro non ha argomenti.

Discussione

Ci sono due macro che riguardano il controllo degli interrupt nel sistema grafico: ON_VBLANK e OFF_VBLANK.

Al sistema Amiga viene spesso richiesto di eseguire alcune operazioni durante l'intervallo di vertical blanking; deve aggiornare vari registri di puntamento e riscrivere, se necessario, la lista Copper per il successivo quadro video. Il sistema possiede due registri di controllo degli interrupt: INTENA (interrupt enable, registro di abilitazione degli interrupt) e INTREQ (interrupt request, registro delle richieste di interrupt). Le impostazioni in questi due registri determinano il modo in cui sono gestiti gli eventi di interrupt.

La macro ON_VBLANK imposta il bit 5 del registro di abilitazione degli interrupt. Quando tale bit è impostato si verifica un interrupt ogni volta che il pennello elettronico finisce un quadro e si appresta a iniziare il successivo. ON_VBLANK è attiva per default; quindi se non si procede alla disattivazione dell'interrupt di vertical blanking con la macro OFF_VBLANK, ogni view (vecchia o nuova) che si carica viene automaticamente mostrata.

OrRectRegion

Sintassi di chiamata della funzione

```
OrRectRegion (region, rectangle)
             A0      A1
```

Scopo della funzione

Questa funzione svolge un'operazione di OR bidimensionale tra un rettangolo di delimitazione e una regione, creando così una nuova regione. Tutte le parti di rettangolo che non sono già presenti nella regione vengono aggiunte alla regione stessa.

Argomenti della funzione

region Indirizzo di una struttura Region. Può anche trattarsi di una regione nulla.

rectangle Indirizzo di una struttura Rectangle che definisce il rettangolo di delimitazione.

Discussione

Nella libreria Graphics ci sono sei funzioni che riguardano specificamente le regioni e i rettangoli di delimitazione che le compongono: AndRectRegion, ClearRegion, DisposeRegion, NewRegion, OrRectRegion e XorRectRegion. Tali funzioni consentono di costruire una propria regione di disegno da agganciare successivamente a un layer al fine di delimitare le aree nelle quali le funzioni di disegno possono accedere. Si vedano anche le altre funzioni ora citate.

La funzione OrRectRegion crea una nuova regione effettuando un'operazione OR fra la regione e il rettangolo di delimitazione indicati dal task come argomenti. L'operazione viene effettuata soltanto a livello geometrico. Trattandosi di un'operazione OR, il risultato è una regione non nulla anche quando la regione originaria e il rettangolo di delimitazione non si sovrappongono, cioè anche quando non hanno parti in comune.

Per comprendere meglio il concetto di rettangolo di delimitazione, si supponga di voler disegnare un quadrato pieno in un layer. Ci sono almeno due modi per ottenere questo obiettivo. Primo: si decide la posizione che deve avere il quadrato nel layer e s'impiegano le funzioni AreaDraw, AreaMove e AreaEnd (o la funzione RectFill) per definirlo e disegnarlo. Si usano le funzioni di disegno (SetAPen, SetBPen...) perché la struttura RastPort si occupi di controllare adeguatamente i colori con cui effettuare il disegno del quadrato e dei bordi. Quando infine vengono eseguite le funzioni AreaEnd o RectFill, il quadrato viene disegnato nella bitmap.

In alternativa, si può usare la funzione OrRectRegion per limitare il disegno a una regione quadrata del layer. Per fare ciò si definisce, per prima cosa, una struttura Rectangle impiegando quattro semplici istruzioni che definiscono i vertici del quadrato che s'intende disegnare.

Si chiama poi la funzione NewRegion per creare una nuova regione di misura zero. In seguito si impiega la funzione OrRectRegion per includere il rettangolo di delimitazione nella nuova regione. Quest'operazione fa in modo che la regione assuma le dimensioni del rettangolo di delimitazione.

A questo punto disponiamo di una definizione di regione e dobbiamo agganciarla al layer inserendo l'indirizzo della struttura Region nel parametro DamageList della struttura Layer (il vecchio indirizzo contenuto in questo parametro lo salviamo temporaneamente, per poi ripristinarlo quando abbiamo terminato di disegnare nel layer).

Eseguiamo ora una chiamata alla funzione BeginUpdate della libreria

Layers, per aggiornare il layer. Dal momento che nella lista delle aree danneggiate abbiamo inserito provvisoriamente l'indirizzo della nostra regione, la funzione predispone il layer perché le funzioni di disegno accedano soltanto a essa. Questo significa che tutte le operazioni di disegno effettuate prima di chiamare la funzione EndUpdate (sempre della libreria Layers) accederanno soltanto all'interno del nostro quadrato (per le spiegazioni delle funzioni BeginUpdate e EndUpdate si veda il capitolo 5). Il sistema continua a tener conto, anche in questo caso, del fatto che il layer potrebbe essere parzialmente coperto, preoccupandosi di effettuare un clip automatico sulla nostra regione qualora sconfini in parti del layer coperte da altri layer.

Ora, come in una normale procedura di disegno, si usano le funzioni di disegno (SetAPen, SetBPen...) per preparare la struttura RastPort del layer, così che possa controllare i colori, i bordi e così via per il disegno del quadrato. Si usa poi la funzione RectFill per riempire il layer con il colore che si desidera. Normalmente l'intera bitmap del layer riceverebbe questo colore (salvo le parti coperte da altri layer), ma avendo definito una regione di clip, verrà riempita solo l'area della bitmap da essa circoscritta.

Si può estendere questo processo di disegno indiretto chiamando più volte la funzione OrRectRegion. Per esempio, prima di collegare la regione al layer e chiamare la funzione BeginUpate, si può definire un altro rettangolo di delimitazione. Si può poi eseguire ancora la funzione OrRectRegion fra la regione e il secondo rettangolo di delimitazione per creare una nuova regione.

Vediamo alcuni casi. Come prima ipotesi supponiamo che il secondo rettangolo di delimitazione si trovi nella stessa posizione del primo: in questo caso la definizione di regione non cambia, perché l'operazione di OR è avvenuta fra una regione e un rettangolo di delimitazione delle stesse dimensioni e sovrapposti.

Supponiamo ora che il secondo rettangolo di delimitazione sia disgiunto dal primo (cioè che i due rettangoli non abbiano parti in comune). In questo caso, chiamando la funzione OrRectRegion si otterrà una regione composta da due rettangoli. Se uniamo la regione alla struttura Layer e chiamiamo la funzione BeginUpdate, le funzioni di disegno che verranno chiamate in seguito opereranno soltanto in questi due rettangoli.

Il terzo caso (quello illustrato dalla Figura 2.5) è quello di due rettangoli che si intersecano. In particolare, supponiamo che il secondo rettangolo di delimitazione si sovrapponga parzialmente al primo. Tutte le successive operazioni di disegno saranno circoscritte alla figura geometrica unione dei due rettangoli di delimitazione.

In conclusione, abbiamo visto che la funzione OrRectRegion permette, data una regione e un rettangolo di delimitazione, di ottenere una nuova regione che è l'unione della regione iniziale e del rettangolo di delimitazione. Abbiamo anche visto che le operazioni con le regioni sono in realtà semplici operazioni geometriche, dal momento che le regioni sono soltanto definizioni di confini. Solo quando si associa una regione a un layer, infatti, la regione individua parti di una bitmap. In pratica, potremmo pensare alle regioni come a fogli bianchi all'interno dei quali siano stati ritagliati dei rettangoli. Disponendo poi il "foglio" su una bitmap, si limita l'azione delle funzioni di disegno alle parti di bitmap che i rettangoli ritagliati nel foglio portano alla

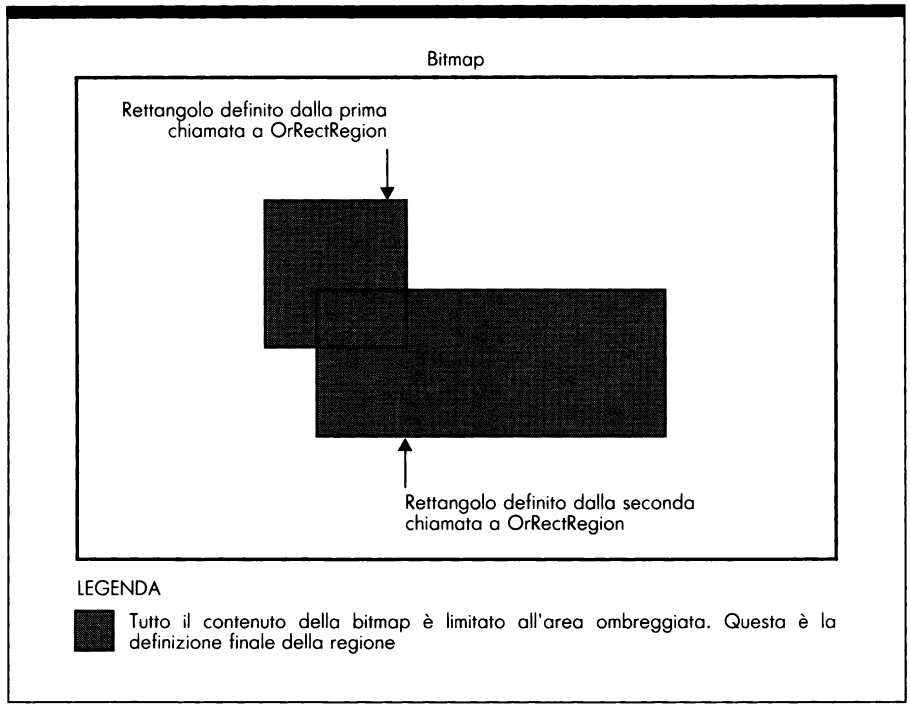


Figura 2.5: *Come opera la funzione OrRectRegion*

luce. Ripetiamo ancora che le regioni non sono altro che insiemi di rettangoli di delimitazione.

Si noti che l'ordine in cui i rettangoli di delimitazione vengono aggiunti alla definizione di regione non ha importanza; la regione da essi definita è comunque sempre la stessa.

La funzione OrRectRegion viene spesso impiegata per creare una regione ex novo. Si procede creando una regione nulla con la funzione NewRegion, e poi si effettua l'operazione OR fra questa regione nulla e un rettangolo di delimitazione della grandezza desiderata. Il risultato è ovviamente una regione non nulla delle dimensioni del rettangolo di delimitazione.

OrRegionRegion

Sintassi di chiamata della funzione

```
success = OrRegionRegion (region1, region2)
D0                        A0       A1
```

Scopo della funzione

Questa funzione svolge un'operazione di OR bidimensionale tra una regione e un'altra regione: unisce due regioni per formarne una terza. Il risultato va a sostituire la seconda regione indicata come parametro, mentre la prima non viene toccata. Questo in genere produce una regione più ampia, a meno che le due regioni non siano identiche come misura, forma e posizione. In tal caso la regione risultante sarà identica alle regioni che concorrono alla sua formazione.

Argomenti della funzione

region1	Indirizzo della struttura Region della prima regione.
region2	Indirizzo della struttura Region della seconda regione.

Discussione

OrRegionRegion è una delle quattro funzioni che riguardano esclusivamente le regioni. Si ricordi che le regioni sono semplici insiemi di rettangoli di delimitazione. Le regioni vengono definite utilizzando le funzioni AndRectRegion, OrRectRegion e XorRectRegion. Lo scopo e il funzionamento della funzione OrRegionRegion è molto simile a quello della funzione OrRectRegion. La sola differenza è che la funzione OrRegionRegion combina due regioni mentre la funzione OrRectRegion combina una regione e un rettangolo di delimitazione.

Si veda anche la spiegazione della funzione OrRectRegion.

A un certo punto del processo di disegno possono esserci due regioni di forma diversa. I loro confini sono per forza rettilinei e paralleli ma, a parte questo, potranno avere qualsiasi forma e misura. Combinando ora queste due regioni tramite la funzione OrRegionRegion, si produrrà una terza regione che risulterà più grande delle prime due e che potrebbe essere utile per la fase successiva del disegno. Questo è lo scopo della funzione OrRegionRegion.

OwnBlitter

Sintassi di chiamata della funzione

OwnBlitter ()

Scopo della funzione

Quando questa funzione restituisce il controllo, il task che ha effettuato la chiamata riceve il controllo esclusivo del Blitter. Prima d'iniziare a usare il Blitter, il task dovrebbe effettuare una chiamata alla funzione WaitBlit. WaitBlit attende che le operazioni di Blitter in coda abbiano termine, svuotando quindi la coda di richieste del Blitter.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Ci sono dieci funzioni video della libreria Graphics che interessano specificamente le operazioni del Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Si vedano anche le spiegazioni relative a queste funzioni.

Il canale DMA di Blitter, che fa parte del chip dedicato all'animazione, è il più veloce trasportatore di dati del sistema Amiga. Impostato nel modo corretto, può muovere un milione di bit d'informazioni al secondo. È evidente che il Blitter è molto più veloce del 68000 per gli spostamenti e per altri tipi di manipolazione che riguardino i dati.

Una volta che il task ha impostato i registri di controllo del Blitter, è possibile muovere dati ad alta velocità per svolgere diverse utili operazioni. Per esempio, il Blitter può essere impiegato per copiare dati e per eseguire disegni di linee. Inoltre può simulare animazioni utilizzando quella che è chiamata animazione di playfield. L'*animazione di playfield* consente al Blitter di spostare una sezione di quadro video in una nuova posizione mentre il pennello elettronico si trova in altre aree video. L'osservatore ha la sensazione che l'oggetto sul video si muova come fosse animato.

Ecco una lista di alcune delle mansioni che il Blitter è in grado di svolgere:

1. Può leggere dati provenienti da un massimo di tre blocchi di RAM, manipolare in 256 modi diversi i bit di dati provenienti da queste fonti e collocare i risultati in un blocco destinazione in RAM.

2. Nel muovere dati da un blocco di memoria a un altro, può lavorare con indirizzi crescenti o decrescenti.

3. Può effettuare scorrimenti di bit (shift), fino a 15 bit, relativamente a una o due delle sorgenti di dati, e poi eseguire operazioni logiche sui dati. Ciò permette il movimento d'immagini in memoria attraverso i confini delle word (indirizzi corrispondenti a byte pari).

4. Può mascherare parte delle word all'estrema sinistra e all'estrema destra di ciascuna linea orizzontale d'informazioni bitmap. Questo serve per svolgere operazioni logiche sui dati e muovere immagini in memoria attraverso i confini delle word (indirizzi corrispondenti a byte pari).

5. Può disegnare semplici linee con qualsiasi angolo d'inclinazione, anche utilizzando matrici di continuità.

6. Possiede modi operativi finalizzati al riempimento di aree.

Dal momento che il Blitter può fare così tante cose e così velocemente, non c'è da sorprendersi che i task grafici, che gestiscono grandi quantitativi di dati, ne facciano un uso intensivo. Tuttavia, quando in un sistema multitasking diversi task (grafici o no) competono per acquisire l'uso esclusivo del Blitter al fine di svolgere il loro lavoro nel modo più veloce possibile, diventa necessario provvedere in qualche modo all'assegnazione esclusiva del Blitter. Ciò è necessario soprattutto quando l'azione di un sotto-task è cruciale per l'esecuzione di un task più esteso. Questo è il compito della funzione OwnBlitter.

La funzione OwnBlitter concede al task che ha effettuato la chiamata l'uso esclusivo del Blitter. Si noti che il task non acquisisce l'uso immediato del Blitter: deve prima attendere che le operazioni in corso di svolgimento si concludano, per non lasciare trasferimenti di dati incompleti. Una volta che le operazioni di Blitter sono terminate, il task che ha chiamato la funzione OwnBlitter si appropria del canale DMA del Blitter, e diventa il solo che può porre richieste nella coda del Blitter o accedervi direttamente a livello hardware.

PolyDraw

Sintassi di chiamata della funzione

PolyDraw (rastPort, count, array)
AØ DØ A1

Scopo della funzione

Questa funzione disegna linee sullo schermo. Inizia con la prima coppia di valori x,y presente nell'array di definizione del poligono, e disegna linee tra quel punto e il successivo, fino a quando non esaurisce tutti i vertici contenuti nell'array.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo, nella cui bitmap viene disegnato il poligono.
count	Numero dei punti (coppie x,y) presenti nella tavola di definizione del poligono.
array	Indirizzo della prima coppia di word x,y contenuta nell'array di definizione del poligono.

Discussione

Nella libreria Graphics ci sono due funzioni che riguardano specificamente il disegno delle linee nel sistema Amiga: Draw e PolyDraw. Queste funzioni consentono d'inserire linee nelle definizioni di bitmap. La funzione Draw disegna una linea tra il punto in cui si trova la penna e il punto x,y. PolyDraw utilizza un array di punti x,y per disegnare un insieme di linee che definiscono una spezzata (eventualmente un poligono).

Nel corso della spiegazione daremo per scontato che i vertici dell'array rappresentino un poligono, ma sia chiaro che non è affatto obbligatorio. Le osservazioni si potranno eventualmente adattare anche al caso di linee spezzate di qualunque tipo.

La funzione PolyDraw lavora con una struttura RastPort; tale struttura

controlla le caratteristiche grafiche con cui la poligonale viene disegnata nella bitmap.

Nell'aggiungere poligoni alla bitmap, si possono alterare i valori delle impostazioni di disegno nella struttura RastPort che viene impiegata nella chiamata alla funzione PolyDraw: il modo di disegno, la penna, la matrice di continuità per le linee...

Questa parte del task consisterà perciò in una serie di chiamate alle funzioni SetDrMd, SetAPen, SetBPen e alle macro SetOPen, SetDrPt, SetWrMsk seguite dalla chiamata alla funzione PolyDraw che si occupa del vero e proprio lavoro di disegno del poligono. Ciascun poligono disegnato interessa i contenuti della bitmap controllata dalla struttura RastPort. Tutte le informazioni di controllo vengono conservate nella struttura RastPort, una struttura dinamica che può essere impostata in modo diverso di volta in volta. Per questa ragione bisogna usare le sue impostazioni per disegnare il poligono considerato, prima di procedere con il seguente, che può avere impostazioni diverse.

Si badi a non confondere la funzione PolyDraw e la funzione AreaDraw. AreaDraw costruisce un insieme di linee (vettori) per definire un'area da riempire poi con una matrice grafica; PolyDraw produce un poligono utilizzando una tavola di valori x,y senza che sia previsto il riempimento. Tuttavia è possibile impiegare la funzione Flood per riempire l'area racchiusa dal perimetro del poligono.

Si ricordi, inoltre, che la funzione AreaDraw richiede un buffer di memoria per fornire un'area temporanea di lavoro destinata alla costruzione di un insieme di vettori di definizione di aree (coppie di valori x,y). La funzione PolyDraw richiede anch'essa un buffer; tuttavia le operazioni per crearlo, tramite gli argomenti array e count della chiamata a PolyDraw, sono diverse da quelle impiegate per definire un buffer di riempimento di area che interessi la funzione AreaEnd.

Ricordiamo infine la differenza tra le funzioni PolyDraw e Draw. Come la funzione Draw, PolyDraw riguarda un insieme di linee che non costituiscono necessariamente poligoni chiusi. La funzione PolyDraw prevede che all'interno di un array siano definiti una serie di valori x,y che, oltre a essere i vertici di una linea spezzata continua, possono anche descrivere una spezzata chiusa non intrecciata, cioè un poligono; tale poligono non può essere riempito dalla funzione AreaEnd. La funzione Draw invece richiede che venga definita una coppia di valori x,y per volta; un'unica linea viene poi tracciata tra la posizione in cui si trova la penna e il punto.

Nessuna di queste due funzioni provvede direttamente al riempimento dell'area. Se si vuole riempire l'area definita tramite Draw o PolyDraw, è necessario impiegare la funzione Flood.

RASSIZE

Sintassi di chiamata della macro

```
num_bytes = RASSIZE (width, height)
DØ                   DØ     D1
```

Scopo della macro

Questa macro calcola e restituisce la quantità di memoria richiesta per un bitplane delle dimensioni (altezza e larghezza) indicate dal task come argomenti. Il valore della quantità di memoria è espresso in byte.

Argomenti della macro

width	Larghezza in pixel del bitplane.
height	Altezza in pixel del bitplane.

Discussione

Questa macro è stata realizzata per risparmiare al programmatore qualche calcolo quando deve conoscere la dimensione in byte dei bitplane di una bitmap.

Si può, per esempio, impiegare questa macro per azzerare tutti i bit di una bitmap senza dover calcolare quanti sono. Proponiamo a questo scopo un insieme di cicli FOR annidati:

```
#define WIDTH 32Ø
#define HEIGHT 2ØØ
#define DEPTH 5

for (i=Ø; i < DEPTH; i++){
     displaybyte = (UBYTE)MyBitMap.Planes[i];
     for (j=Ø; j < RASSIZE(WIDTH,HEIGHT);j++)
          displaybyte++ = Ø;
}
```

QBlit

Sintassi di chiamata della funzione

QBlit (bltnode)
A1

Scopo della funzione

Questa funzione aggiunge una richiesta di uso del Blitter alla fine della relativa coda. Nell'argomento bltnode il task deve indicare una struttura bltnode che descriva la richiesta. Questa struttura deve contenere l'indirizzo della routine di gestione del Blitter da chiamare quando il Blitter raggiunge la richiesta. La routine fa generalmente parte del task.

Argomenti della funzione

bltnode Indirizzo della struttura bltnode.

Discussione

Ci sono dieci funzioni video della libreria Graphics che interessano specificamente le operazioni di Blitter nel sistema Amiga: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Si vedano anche le spiegazioni di queste funzioni.

Ci sono due modi per porre in coda una richiesta per il Blitter. Il primo è usare la funzione QBSBlit. In tal caso la richiesta di Blitter sarà sincronizzata con la posizione del pennello elettronico durante la scansione di un quadro video. Quando il pennello elettronico raggiunge la posizione x,y specificata nella struttura bltnode, inizia l'esecuzione della routine Blitter che si trova in coda. Questa procedura consente di riscrivere certe parti della presentazione di schermo memorizzata in RAM, dopo che il pennello elettronico ha superato un determinato punto del quadro. Ciò previene lo sfarfallio dello schermo che spesso si verifica quando il task è impegnato a riscrivere la bitmap di una view mentre questa viene visualizzata.

Tuttavia, se la richiesta di Blitter non interessa simultaneamente la riscrittura e la presentazione video, si può usare la funzione QBlit per porre in

Nome del registro	Indirizzo del registro*	Funzione del registro
BLTDDAT	000	Indirizzo della RAM destinazione
BLTCON0	040	Registro di controllo 0
BLTCON1	042	Registro di controllo 1
BLTAFWM	044	Prima word-maschera per la sorgente A
BLTALWM	046	Ultima word-maschera per la sorgente A
BLTCPTH	048	Puntatore alla sorgente C (3 bit alti)
BLTCPTL	04A	Puntatore alla sorgente C (15 bit bassi)
BLTBPTH	04C	Puntatore alla sorgente B (3 bit alti)
BLTBPTL	04E	Puntatore alla sorgente B (15 bit bassi)
BLTAPTH	050	Puntatore alla sorgente A (3 bit alti)
BLTAPTL	052	Puntatore alla sorgente A (15 bit bassi)
BLTDPTH	054	Puntatore alla destinazione D (3 bit alti)
BLTDPTL	056	Puntatore alla destinazione D (15 bit bassi)
BLTSIZE	058	Inizio e dimensione (larghezza e altezza della bitmap)
BLTCMOD	060	Modulo per sorgente C
BLTBMOD	062	Modulo per sorgente B
BLTAMOD	064	Modulo per sorgente A
BLTDMOD	066	Modulo per destinazione D
BLTCDAT	070	Registro dati della sorgente C
BLTBDAT	072	Registro dati della sorgente B
BLTADAT	074	Registro dati della sorgente A

*l'indirizzo 000 corrisponde all'indirizzo di memoria $DFF000. Gli indirizzi per il 68000 sono dati da $DFF000 + (indirizzo registro)

Tavola 2.3: *Nomi dei registri del Blitter, indirizzi e funzioni*

coda la richiesta di Blitter. La funzione QBlit va usata per i trasferimenti di dati ad alta velocità tra due blocchi di RAM non visualizzati sullo schermo.

La funzione QBlit lavora con una routine di Blitter specificata dal task e collegata al sistema. Questa routine è molto simile a quella usata dalla funzione QBSBlit; entrambe sono routine di controllo e impiego del Blitter. Si veda anche la spiegazione della funzione QBSBlit.

Quando viene chiamata la routine di gestione del Blitter, è il task ad averne il controllo: il Blitter non sarà occupato con nessun'altra richiesta di task. Ciò significa che si può specificare il contenuto dei registri del Blitter e iniziare l'operazione. I registri del Blitter sono descritti nella Tavola 2.3.

Quando il pennello elettronico raggiunge il punto specificato dalla struttura bltnode impiegata con la funzione QBSBlit, qualsiasi richiesta di Blitter accodata dalla funzione QBSBlit avrà la precedenza su ogni richiesta accodata dalla funzione QBlit.

Si noti che per brevi operazioni di Blitter l'uso della la funzione QBlit è più lungo e complesso dell'uso delle funzioni OwnBlitter e DisownBlitter.

QBSBlit

Sintassi di chiamata della funzione

QBSBlit (bltnode)
A1

Scopo della funzione

Questa funzione sincronizza la richiesta di Blitter con una particolare posizione del pennello elettronico sul video. Quando il pennello elettronico raggiunge quella posizione durante la scansione del video, la richiesta sottoposta al blitter diventa attiva, e il sistema manda in esecuzione la routine del task indicata dalla struttura bltnode.

La richiesta viene messa in coda separatamente rispetto alle richieste accodate con la funzione QBlit.

QBSBlit è particolarmente utile quando si vogliono copiare alcuni dati grafici in una posizione della bitmap che fa parte dell'area visibile dello schermo e questi dati si vogliono spostare dopo che il raggio di scansione ha oltrepassato l'area visibile. Questo impedisce che venga simultaneamente visualizzata parte dello schermo vecchio e parte del nuovo. Le richieste di Blitter presenti nella coda QBSBlit hanno la precedenza su tutte quelle che si trovano nella normale coda Blitter.

Argomenti della funzione

bltnode Indirizzo della struttura bltnode.

Discussione

Ci sono dieci funzioni video della libreria Graphics che interessano specificamente le operazioni di Blitter: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Si vedano anche le spiegazioni di queste funzioni.

La struttura bltnode, definita nel file hardware/blit.h, è abbastanza breve:

```
struct bltnode {
        struct bltnode *n;
        int (*function)();
        char stat;
        short blitsize;
        short beamsync;
        int (*cleanup)();
};
```

La struttura bltnode contiene i seguenti elementi:

- un puntatore a un'altra struttura bltnode di una lista semplice.
- L'indirizzo della routine di gestione del Blitter che viene eseguita quando il pennello elettronico raggiunge una particolare posizione.
- Il valore di sincronizzazione del pennello elettronico, costituito dalla combinazione della sua posizione verticale e di quella orizzontale; quando il pennello raggiunge il punto stabilito, la routine viene mandata in esecuzione.
- Un flag di stato che informa il sistema se è necessario o meno eseguire la routine di pulizia quando l'ultima operazione di Blitter è stata completata.
- Indirizzo della routine di pulizia che dev'essere usata se risulta impostato il flag di stato.

La prima riga della definizione della struttura bltnode:

```
struct bltnode *n;
```

definisce un puntatore alla successiva struttura bltnode. Nella maggior parte dei casi, i nodi blit non sono legati tra loro, e questo parametro contiene il valore zero. La seconda riga:

```
int (*function)();
```

è l'indirizzo della routine che dev'essere chiamata quando la richiesta di Blitter raggiunge la cima della coda. La terza riga:

```
char stat;
```

è il valore che, quando risulta impostato a CLEANUP (0x40), determina l'esecuzione di una routine di cancellazione dopo la conclusione della routine di Blitter. Se il valore è zero, la routine di cancellazione non viene eseguita. La quarta riga:

```
short beamsync;
```

è il valore (denominato VBEAM) che deve trovarsi nel contatore del pennello elettronico dell'hardware di sistema. Serve per determinare quando deve verificarsi un'operazione Blitter sincronizzata con la posizione del pennello elettronico, come specificato dalla funzione QBSBlit. Il sistema adopera questo valore per prorogare l'esecuzione della routine di Blitter disposta dal task fino a che il pennello elettronico non oltrepassa la posizione di valore VBEAM. Questa caratteristica è particolarmente utile quando si definiscono schermi a buffer singolo; essa previene lo sfarfallio che si verifica quando un task presenta informazioni video mentre contemporaneamente cambia la loro definizione in RAM. Attendendo che il pennello elettronico raggiunga la posizione VBEAM, le informazioni di schermo che precedono quel punto del quadro video sono già state presentate e possono perciò essere alterate senza causare sfarfallio. L'ultima riga:

```
int (*cleanup)();
```

è l'indirizzo della routine di cancellazione. Tale routine viene chiamata quando il task riprende il controllo al termine della chiamata alla funzione QBSBlit; ciò accade quando la routine di gestione del Blitter restituisce il valore zero. In quel momento le routine di sistema chiamano automaticamente la routine di cancellazione. Si può usare questa routine per liberare la memoria assegnata alle routine di gestione del Blitter e per eseguire ogni operazione necessaria per un'uscita "pulita".

La routine chiamata dall'accodatore del Blitter (Blitter Queuer) dev'essere una subroutine terminante con una istruzione RTS del 68000. Se non si usa una routine in linguaggio C, il valore restituito deve trovarsi nel registro D0. Il sistema chiama più volte la routine, finché non restituisce il valore zero. Ciò consente alla routine di mantenere il controllo del Blitter per spostare più oggetti e contemporaneamente salvare le parti di bitmap da essi coperte. In una situazione simile, ci si deve assicurare che tutti i bitplane dell'oggetto siano opportunamente disposti prima che un altro oggetto venga collocato in quella posizione dello schermo. L'esigenza di ottenere di ritorno un valore zero, ritarda l'avanzamento della coda di Blitter finché non è stato trattato ogni aspetto della richiesta in corso di elaborazione.

In genere le routine per la funzione QBSBlit sono scritte in linguaggio Assembly piuttosto che in C. Ciò consente alle routine di gestione della coda di passare parametri direttamente nei registri di sistema. Ecco le convenzioni per il passaggio di valori tramite i registri:

- il sistema colloca in A0 l'indirizzo base dei registri hardware di sistema; tutti gli altri registri hardware possono essere indirizzati sommando un opportuno offset a questo indirizzo.

- Il sistema colloca in A1 l'indirizzo della struttura bltnode. Ciò consente ai task di sottoporre al Blitter richieste che fanno capo alla stessa routine di gestione, la quale riconosce la natura della richiesta esaminando il contenuto del registro A1 non appena riceve il controllo. I dati per queste richieste di Blitter correlate possono essere ottenuti ai vari offset dall'indirizzo in A1. Il task dovrebbe calcolare in anticipo i valori dei registri hardware e porli negli appropriati registri nel corso dell'esecuzione della routine.

ReadPixel

Sintassi di chiamata della funzione

```
pennum = ReadPixel (rastPort,    x,    y)
D0                  A1           D0    D1
```

Scopo della funzione

Date le coordinate di un pixel in una bitmap, questa funzione considera nel loro complesso i bit corrispondenti a quel pixel nei vari bitplane, e restituisce il numero di penna associato al pixel, cioè il suo colore.

Se viene eseguita con successo, ReadPixel restituisce un numero di penna (pennum) compreso tra 0 e 255. Se il pixel si trova fuori dai confini della bitmap, la funzione restituisce il valore –1.

Argomenti della funzione

rastPort Indirizzo della struttura RastPort di controllo nella cui bitmap il task desidera identificare il colore di un pixel.

x	Coordinata x del pixel nel sistema di coordinate della bitmap.
y	Coordinata y del pixel nel sistema di coordinate della bitmap.

Discussione

Ci sono due funzioni di disegno appartenenti alla libreria Graphics che riguardano specificamente i pixel: ReadPixel e WritePixel. Queste funzioni consentono di leggere e cambiare il numero di penna (cioè il numero di registro di colore) di un particolare pixel all'interno di una determinata bitmap.

Si ricordi che le coordinate per la funzione ReadPixel vengono misurate nel sistema di coordinate della bitmap, e possono quindi arrivare fino a 1024 per 1024 pixel. La posizione di origine 0,0 è l'angolo superiore sinistro della bitmap. Secondo tale convenzione le coordinate x positive vanno verso destra e le coordinate y positive vanno verso il basso. Si tenga sempre presente questo quadro di riferimento. Non si confondano il sistema di coordinate della bitmap e il sistema di coordinate dello schermo video (che è limitato a 640 per 512 pixel in alta risoluzione con interlace, e che ha l'origine nell'angolo superiore sinistro dello schermo video fisico).

Sebbene il valore di ritorno sui modelli Amiga attuali sia compreso tra 0 e 63, versioni future della macchina potrebbero consentire otto bitplane in uno schermo single-playfield, portando così a 256 i colori disponibili.

RectFill

Sintassi di chiamata della funzione

```
RectFill (rastPort, xmin, ymin, xmax, ymax)
          A1        DØ    D1    D2    D3
```

Scopo della funzione

Questa funzione riempie l'area rettangolare specificata dai parametri indicati, usando l'opportuno colore di penna, la matrice grafica di riempimento prescelta e il modo di disegno attivo.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
xmin, ymin	Coordinate dell'angolo superiore sinistro del rettangolo.
xmax, ymax	Coordinate dell'angolo inferiore destro del rettangolo.

Discussione

Ci sono sei funzioni di disegno della libreria Graphics che hanno a che fare specificamente con la creazione di aree: AreaDraw, AreaEnd, AreaMove, Flood, InitArea e RectFill. Si vedano anche le spiegazioni di queste funzioni.

Ci sono diversi modi per riempire un'area rettangolare con una matrice grafica di riempimento. Si possono impiegare le funzioni AreaDraw, AreaMove e AreaEnd; oppure le funzioni Draw, Move e Flood; o ancora le funzioni PolyDraw e Flood. Tuttavia la funzione RectFill fornisce il modo migliore e più veloce per riempire un'area rettangolare con una matrice grafica.

Il riempimento di un'area rettangolare viene svolto su una specifica bitmap. Si tratta della bitmap associata alla struttura RastPort di controllo indicata dal task come primo argomento.

Nel riempire aree differenti, si possono alterare i valori delle varie penne e delle matrici grafiche per definire le caratteristiche di disegno della successiva area da riempire con una nuova chiamata a RectFill. Il task sarà quindi composto da una serie di chiamate alle funzioni SetDrMd, SetAPen, SetBPen e alle macro SetDrPt, SetWrMsk, SetAfPt, immediatamente seguite da una o più chiamate alla funzione RectFill.

Si possono così variare il modo di disegno, la penna e la matrice grafica via via che si procede a riempire una serie di aree rettangolari. Tutte le informazioni di controllo sono conservate nella struttura RastPort.

Chiamando la funzione RectFill si può riempire un'intera bitmap. A questo scopo, tuttavia, è stata appositamente progettata la funzione SetRast. C'è un'importante differenza tra le funzioni RectFill e SetRast: la funzione SetRast consente di cambiare soltanto il colore di riempimento, preventivamente stabilito con la funzione SetAPen. La funzione RectFill, invece, consente di cambiare la matrice di riempimento, il modo di disegno (FgPen, BgPen, AOlPen), le matrici grafiche e quelle di continuità per le linee. Per questo, la quantità d'informazioni che si possono inserire in una bitmap con una sola chiamata di funzione risulta considerevolmente maggiore impiegando RectFill anziché SetRast. Tramite RectFill, una bitmap estesa, costituita da 1024 x 1024 pixel, può essere riempita molto rapidamente con la matrice grafica e con il colore che si preferisce.

Si noti che la funzione RectFill non richiede al task di specificare alcun buffer. Sebbene vengano usati alcuni buffer, essi sono interamente controllati

dal sistema. Questa costituisce un'ulteriore differenza tra la funzione RectFill e il terzetto AreaDraw/AreaMove/AreaEnd che richiedono un'area buffer esplicita e una struttura aggiuntiva (la struttura TmpRas) per controllare le operazioni di disegno. RectFill è diversa anche dalla funzione PolyDraw, che imposta un proprio buffer per i valori x,y che definiscono i vertici di quello che potrebbe eventualmente divenire un poligono chiuso.

Si noti infine che la funzione RectFill disegna anche le linee di contorno del rettangolo che riempie. Per aggiungere linee di contorno dotate di una determinata matrice di continuità si impiegano le informazioni contenute nella struttura RastPort. Se non si vogliono aggiungere i contorni, bisogna "ingannare" la funzione RectFill, modificando la matrice di continuità definita nella struttura RastPort affinché corrisponda alla matrice e al colore di fondo. In questo modo, i contorni vengono comunque disegnati, ma sono uguali allo sfondo.

ScrollRaster

Sintassi di chiamata della funzione

```
ScrollRaster (rastPort, dx, dy, xmin, ymin, xmax, ymax)
              A1        D0  D1  D2    D3    D4    D5
```

Scopo della funzione

Questa funzione sposta i bit di una bitmap di dx,dy pixel rispetto all'origine 0,0 del sistema di coordinate della bitmap. L'area resa libera viene riempita (impiegando RectFill) con il colore di BgPen. ScrollRaster limita le operazioni di scorrimento al rettangolo definito dai vertici xmin,ymin e xmax,ymax. Qualsiasi bit al di fuori di tale rettangolo non sarà interessato allo scorrimento.

Argomenti della funzione

rastPort	Indirizzo alla struttura RastPort di controllo.
dx,dy	Numero di pixel di cui dev'essere spostata la bitmap, rispettivamente in orizzontale e in verticale; si tratta di numeri interi che possono essere positivi, nulli o negativi.

xmin, ymin Coordinate dell'angolo superiore sinistro del sotto-rettangolo.

xmax, ymax Coordinate dell'angolo inferiore destro del sotto-rettangolo.

Discussione

Ci sono quattro funzioni video nella libreria Graphics che riguardano specificamente le bitmap associate alle strutture RastPort: AllocRaster, FreeRaster, SetRast e ScrollRaster. Le prime tre assegnano, liberano e impostano le aree di memoria per i bitplane e per le bitmap da essi formate. ScrollRaster fornisce il mezzo per effettuare lo scorrimento sullo schermo video (scroll) delle informazioni provenienti dalla bitmap.

La funzione ScrollRaster permette di ottenere lo scorrimento di un sotto-rettangolo all'interno di un rettangolo più esteso costituito dalla bitmap stessa. Questa funzione, insieme alla funzione ScrollVPort, è in parte responsabile della magia grafica che caratterizza l'Amiga, poiché fornisce la capacità di muovere informazioni grafiche dentro e fuori lo schermo. Per esempio, in un programma grafico, una view può scorrere entrando nello schermo dall'alto, mentre un'altra, già presente e completamente diversa, viene fatta scorrere in uscita dal basso. Con un appropriato calcolo dei tempi, questa procedura può dare all'utente l'illusione di fotogrammi di un film che scorrono lentamente sullo schermo dell'Amiga.

Per usare la funzione ScrollRaster si deve innanzitutto indicare con quale bitmap si sta lavorando. Quando si chiama la funzione ScrollRaster, si deve specificare la struttura RastPort che ha il controllo delle operazioni di disegno nella bitmap. Nella struttura RastPort è presente un puntatore alla struttura BitMap che definisce la bitmap considerata.

Si devono specificare anche altri sei argomenti richiesti dalla funzione ScrollRaster. Si tratta di dx,dy, xmin,ymin e xmax,ymax, cioè delle quantità di pixel di cui il sotto-rettangolo dev'essere spostato, e dei punti necessari per definire la grandezza del sotto-rettangolo. È importante ricordare che tutti questi valori sono espressi nel sistema di coordinate della bitmap, non nel sistema di coordinate dello schermo video. Gli argomenti dx,dy definiscono la distanza di spostamento del sotto-rettangolo nella bitmap. Questa definizione può trarre in inganno: valori positivi di dx significano che il sotto-rettangolo scorre verso sinistra; valori positivi di dy significano che scorre verso l'alto. In altre parole, incrementi positivi avvicinano il sotto-rettangolo al punto di origine 0,0 della bitmap.

Le variabili xmin,ymin e xmax,ymax fissano la misura del sotto-rettangolo. Si tenga presente che con una bitmap molto grande il sotto-rettangolo può essere molto esteso. In pratica può trattarsi di un'intera view. Per esempio, in bassa risoluzione senza interlace il sotto-rettangolo definito da xmin,ymin e xmax,ymax può rappresentare una view di 320 per 256 pixel (standard PAL).

L'area che lo spostamento del sotto-rettangolo lascia vuota viene riempita

con il colore definito dal valore del parametro BgPen nella struttura RastPort di controllo. Se lo scorrimento è esclusivamente a destra o a sinistra senza spostamento verticale, occorre associare alla struttura RastPort una struttura TmpRas inizializzata, di dimensioni sufficienti a contenere il rettangolo da spostare.

ScrollVPort

Sintassi di chiamata della funzione

ScrollVPort (viewPort)
A0

Scopo della funzione

Questa funzione sposta una viewport secondo le definizioni di RxOffset e RyOffset nella corrispondente struttura RasInfo. ScrollVPort cambia automaticamente la lista delle istruzioni di Copper per adeguare la posizione di viewport allo scorrimento effettuato. Quest'operazione viene eseguita dopo che il task ha aggiornato i valori di offset della definizione di viewport, valori presenti nella struttura RasInfo associata con la struttura ViewPort.

ScrollVPort modifica la lista delle istruzioni hardware del Copper per tener conto delle nuove informazioni presenti nella struttura RasInfo. Alterare il puntatore alla struttura BitMap, nella struttura RasInfo, senza alterare gli offset corrisponde a creare un effetto double-buffer.

Argomenti della funzione

viewPort	Indirizzo della struttura ViewPort che corrisponde a quanto si vede sullo schermo video.

Discussione

Ci sono cinque funzioni video nella libreria Graphics che riguardano specificamente le viewport dello schermo video: InitView, InitVPort, LoadView, MakeVPort e ScrollVPort. Si vedano anche le spiegazioni di queste funzioni.

Una delle più belle caratteristiche dell'Amiga è la rapidità con cui lo schermo video risponde ai comandi dell'utente. Ciò viene dimostrato innanzi tutto dalla velocità con cui le finestre possono essere spostate sullo schermo quando si sfrutta l'interfaccia utente Intuition. Così, quando si sposta il puntatore del mouse l'intera finestra lo segue, senza far attendere l'utente. L'immediata risposta dell'Amiga fa sentire l'utente più a suo agio con la macchina.

Lo stesso tipo di movimento istantaneo dello schermo potrebbe essere ottenuto da qualsiasi altro programma. Questa è la situazioni in cui entrano in gioco le funzioni ScrollVPort e ScrollRaster.

S'immagini di fare scorrere una viewport all'interno di una view complessiva di schermo. È sufficiente chiamare la funzione ScrollVPort indicando l'indirizzo della relativa struttura ViewPort modificata opportunamente nei parametri di offset RxOffset e RyOffset contenuti nella struttura RasInfo associata, ed eventualmente anche nel puntatore alla bitmap.

Per effettuare lo scorrimento di una viewport all'interno di una view devono accadere molte cose. In particolare, ogni volta che lo schermo viene in qualche modo alterato, devono essere riscritte la maggior parte delle istruzioni (se non tutte) della lista di Copper per la view considerata.

Come di consueto, la lista delle istruzioni Copper viene ridefinita durante l'intervallo di vertical-blanking (il periodo di tempo durante il quale il pennello elettronico si spegne e risale fino all'angolo in altro a sinistra dello schermo per iniziare il quadro successivo.). Durante l'intervallo di vertical-blanking può aver luogo la completa ricostruzione della lista delle istruzioni di Copper. Questa tecnica di sfruttamento del tempo (una completa ricostruzione della definizione di schermo si completa in molto meno di 1/60 di secondo) fornisce una tale velocità di aggiornamento video da collocare l'Amiga in una categoria a parte rispetto agli altri microcomputer.

Si noti infine che la funzione ScrollVPort si distingue dalla funzione ScrollRaster per il suo modo di trattare l'area di schermo che viene "scoperta" in seguito allo scorrimento. La funzione ScrollRaster riempie quest'area con il colore della penna di fondo, mentre la funzione ScrollVPort la riempie con le informazioni video contenute nella parte di bitmap che viene così resa visibile.

SetAfPt

Sintassi di chiamata della macro

SetAfPt (rastPort, areaPtrn, potenzaDiDue)
A0 A1 D0

Scopo della macro

Questa macro imposta in una struttura RastPort la matrice di riempimento per le aree da utilizzare con le apposite funzioni di disegno.

Argomenti della macro

rastPort Indirizzo della struttura RastPort di controllo nella quale si vuole cambiare la matrice grafica di riempimento.

areaPtrn Indirizzo della prima word di una matrice grafica di riempimento memorizzata in RAM.

potenzaDiDue Numero delle word nel pattern di riempimento, espresso come potenza di due.

Discussione

Ci sono quattro macro che riguardano specificamente la definizione dei parametri di controllo del disegno nella struttura RastPort: SetOPen, SetDrPt, SetAfPt e SetWrMsk. Queste macro servono di complemento alle funzioni di disegno.

Quando la macro SetAfPt restituisce il controllo, i valori dei parametri AreaPtrn e AreaPtSz, o di entrambi, risultano cambiati nella struttura RastPort indicata. Qualsiasi operazione di riempimento di aree che impieghi quella struttura RastPort verrà effettuata utilizzando le nuove definizioni.

La matrice di riempimento è un insieme di word i cui singoli bit controllano il modo in cui vanno riempite le aree. Il parametro AreaPtrn punta alla prima word dell'insieme. Il numero delle word dell'insieme è determinato dall'argomento potenzaDiDue nella chiamata alla macro. Se tale parametro vale 1, nella definizione del parametro di riempimento c'è soltanto una word; se vale 2 ci sono quattro word, se vale 3 ci sono otto word. Le dimensioni della matrice sono dunque 16 x 2^potenzaDiDue pixel.

Prima di chiamare la macro SetAfPt, si deve costruire in memoria l'insieme di word che costituisce la matrice grafica di riempimento. Se, per esempio, si vogliono inserire otto word in una determinata posizione di memoria, il parametro potenzaDiDue va impostato a 3. Si avrà in questo modo il controllo di 128 bit (8 volte 16) che possono essere impostati per produrre il tipo di riempimento di area che si desidera.

Quando, in seguito, si chiamano le funzioni di disegno per il riempimento (AreaEnd, Flood...), questa matrice verrà utilizzata per definire i pixel nella nuova area prescelta. Le impostazioni del modo di disegno (FgPen, BgPen...)

determineranno quali colori verranno assegnati ai pixel della bitmap. Per esempio, nel modo di disegno JAM1, ciascun bit di valore 1 riceverà il colore FgPen, mentre ciascun bit di valore zero non verrà alterato. La posizione di tutte le matrici di riempimento definite in tal modo viene sempre determinata rispetto all'angolo superiore sinistro del disegno definito dalla struttura RastPort e dalle funzioni di disegno.

Esiste un'opzione della funzione SetAfPt che consente di produrre più colori nella matrice di riempimento. Ciò viene ottenuto indicando un valore negativo nell'argomento potenzaDiDue. Se si usa questa opzione, si devono predisporre tanti piani di definizione per la matrice di riempimento quanti sono i bitplane nella bitmap nella quale si sta disegnando, ognuno composto di un numero di word pari a 2 elevato al valore assoluto di potenzaDiDue. Per esempio, se si vuole riempire un rettangolo che fa parte di una bitmap a cinque bitplane, si deve definire il riempimento di area con cinque insiemi di 2^5 word. Ciascun insieme di word può essere diverso, consentendo di definire i colori del rettangolo quanto più dettagliatamente si desidera.

SetAPen

Sintassi di chiamata della funzione

SetAPen (rastPort, pennum)
A1 DØ

Scopo della funzione

Questa funzione imposta nella struttura RastPort il colore della penna di primo piano per le linee, il riempimento di aree e il testo.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
pennum	Numero di penna tra 0 e 255; questo numero indica in pratica il registro di colore.

Discussione

Ci sono due funzioni di disegno della libreria Graphics che riguardano specificamente la definizione delle penne di disegno nel sistema Amiga: SetAPen e SetBPen. Inoltre il sistema prevede la macro SetOPen. Si vedano anche le spiegazioni di queste funzioni e macro.

Una volta che si è creata la struttura RastPort e la bitmap a essa associata, si definiscono i colori da assegnare a ciascun pixel della bitmap. Ci sono molti modi per raggiungere questo obiettivo; tutti i diversi metodi impiegano le tre penne di disegno (FgPen, BgPen e AOlPen) oltre ai quattro modi di disegno (JAM1, JAM2, COMPLEMENT e INVERSVID) in varie combinazioni per produrre il colore desiderato.

Dopo che sono stati definiti tutti i colori dei pixel (ed eventualmente altre caratteristiche, come le matrici e gli attributi del testo), si possono suddividere le varie parti della bitmap in diverse viewport e unire in seguito tali viewport in una view. Quando questa view viene caricata (fornita all'hardware video per la presentazione effettiva) con la funzione LoadView, il programma produce sullo schermo i colori desiderati.

La funzione SetAPen impiega FgPen (chiamato anche APen) per assegnare fino a 256 colori ai pixel della bitmap. FgPen è la penna di primo piano, ossia quella primaria per il disegno. Il modo di disegno JAM1 impiega esclusivamente questa penna; il modo JAM2 la utilizza insieme a BgPen. I modi di disegno COMPLEMENT e INVERSVID usano anch'essi questa penna.

Il parametro FgPen nella struttura RastPort contiene il valore della penna colore A. Si può assegnare un particolare colore a FgPen chiamando la funzione SetAPen, indicando l'indirizzo di una struttura RastPort e specificando il valore della penna di primo piano che si vuole usare per i pixel che appartengono alla bitmap. I numeri di penna possono variare da 0 a 255, per un totale di 256 valori assegnabili. L'associazione dei colori con i numeri di penna viene fissata dalle funzioni SetRGB4 e LoadRGB4.

Una volta che il nuovo numero di penna è stato impostato, si può chiamare qualsiasi funzione di disegno per disegnare linee, riempire aree o colorare testi con l'impiego del parametro FgPen. Per esempio, si chiama per prima cosa SetAPen, specificando una struttura RastPort e un numero di penna. Si chiama poi ripetutamente AreaDraw e AreaMove per definire un insieme di punti correlati a un insieme di aree che si vogliono riempire con il colore impostato attraverso SetAPen. Si chiama infine AreaEnd per riempire tali aree con il colore prescelto. Si ricordi che prima di chiamare AreaDraw, si devono chiamare InitArea per predisporre in RAM un'area di lavoro dove possano trovare spazio i vettori dei punti terminali che definiscono le linee che si vogliono disegnare, e InitTmpRas per predisporre un buffer di memoria adeguato.

Si continua allo stesso modo per le altre aree che si vogliono riempire. Basta eseguire appropriatamente le funzioni SetAPen, AreaDraw, AreaMove e AreaEnd finché non si sono disegnate tutte le aree da riempire previste per quella struttura RastPort.

Per il disegno di linee si segue un'analoga serie di operazioni. Basta chiamare SetAPen per impostare il numero di penna, chiamare Move per

spostare la penna e/o utilizzare Draw per disegnare tra due punti una linea del colore prescelto. Si può usare anche la funzione PolyDraw per disegnare il contorno di poligoni nel colore FgPen.

Si può procedere in tal modo per definire tutti gli elementi della bitmap. Si può poi alterare il puntatore alla struttura RastPort e fare la stessa cosa con un'altra bitmap. La stessa cosa si ripete per tutte le bitmap di cui si vogliono definire i colori dei pixel. Si tenga sempre presente che la bitmap altro non è che una serie di bitplane.

I colori che effettivamente appaiono sullo schermo vengono prodotti quando la bitmap viene caricata nell'hardware con la funzione LoadView. I registri di colore del sistema guardano poi alla tavola dei colori (si veda la funzione GetColorMap) per sapere come devono interpretare i valori dei bit della bitmap.

I numeri di penna consentono 256 diversi valori. D'altra parte, l'attuale versione della macchina è limitata all'impiego di sei bitplane ($2^6 = 64$) e quindi gli unici valori significativi sono quelli che vanno da 0 a 63. Se una versione successiva della macchina permetterà d'impiegare otto bitplane per bitmap, potrà impiegare tutti i 256 colori possibili.

Si veda anche la spiegazione della funzione SetDrMd.

SetBPen

Sintassi di chiamata della funzione

```
SetBPen (rastPort, pennum)
          A1       DØ
```

Scopo della funzione

Questa funzione imposta nella struttura RastPort il colore della penna di fondo per le linee, il riempimento di aree e il testo.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
pennum	Numero di penna (compreso tra 0 e 255). Questo numero indica in pratica il registro di colore.

Discussione

Ci sono due funzioni di disegno della libreria Graphics che riguardano specificamente la definizione delle penne di disegno: SetAPen e SetBPen. Inoltre il sistema prevede la macro SetOPen.

La funzione SetBPen impiega BgPen (chiamato anche BPen) per assegnare fino a 256 colori ai pixel di una particolare bitmap. BgPen è la penna di fondo, ossia quella secondaria.

Si può usare la funzione SetBPen praticamente nello stesso modo in cui si usa la funzione SetAPen, salvo per le differenze d'impiego nei modi di disegno. Il modo di disegno JAM1 non impiega per nulla BgPen. Il modo di disegno JAM2 impiega la penna FgPen dove esistono valori 1 nella definizione di linee o matrici di riempimento e BgPen dove ci sono valori 0. I modi di disegno COMPLEMENT e INVERSVID usano anch'essi in vari modi questa penna. Si veda anche la spiegazione della funzione SetDrMd.

SetDrMd

Sintassi di chiamata della funzione

SetDrMd (rastPort, drawmode)
A1 DØ

Scopo della funzione

Questa funzione imposta il modo di disegno per le linee, il riempimento di aree e il testo. Il modo di disegno dipende dai flag impostati nel parametro DrawMode della struttura RastPort.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
drawmode	Specificazione del modo di disegno.

Discussione

SetDrMd è la sola funzione di disegno della libreria Graphics che riguardi specificamente i modi di disegno impiegati per definire gli elementi dello schermo video dell'Amiga. SetDrMd consente d'impostare in una data struttura RastPort il modo di disegno per le linee, per il riempimento di aree e per il testo.

Il modo di disegno specificato si riferisce soltanto alla struttura RastPort sulla quale è intervenuta la funzione. Si veda anche l'introduzione di questo capitolo.

Se si esaminano tutte le chiamate alle funzioni di disegno si vedrà che richiedono sempre come argomento l'indirizzo della struttura RastPort nella cui bitmap devono agire. Questo significa che tutte le funzioni di disegno si rivolgono a una struttura RastPort. Possono esserci diverse strutture RastPort; alcune possono essere attive e altre passive. I comandi di disegno vengono poi diretti a quella scelta dal task in esecuzione.

I modi di disegno

Per definire le informazioni nella struttura RastPort sono disponibili quattro modi di disegno. Il primo è JAM1. Se si specifica questa costante nella struttura RastPort, significa che si sta utilizzando la penna di disegno primaria per colorare i pixel nella bitmap associata alla struttura RastPort. Si imposteranno (o si azzereranno) i bit dei bitplane, in accordo con la definizione di bit della penna di disegno primaria. Nel modo JAM1 si può inserire un solo colore nella bitmap di destinazione; ogni pixel attivo nella bitmap verrà disegnato con il colore FgPen. Variando il colore FgPen si possono cambiare i colori assegnati ai pixel della bitmap.

Il secondo modo di disegno è JAM2. Questo modo viene impiegato soltanto per il disegno di matrici di linea e di area. Se si specifica questa costante nella struttura RastPort significa che si stanno utilizzando entrambe le penne di disegno (primaria e secondaria) per impostare i bit nella bitmap associata alla struttura RastPort. Con questo modo si possono inserire due colori nella bitmap di destinazione. Quando nella matrice grafica di area o di linea c'è un bit a 1 il colore FgPen rimpiazza il colore del pixel. Quando nella matrice c'è un bit a 0 viene adoperato il colore BgPen. Variando i colori FgPen e BgPen, si possono cambiare i colori assegnati a ciascun pixel nella bitmap in cui disegnamo una linea o una matrice grafica.

Il terzo modo di disegno è COMPLEMENT. Questo modo rileva, per prima cosa, i bit nella definizione della bitmap ed esegue su di essi un'operazione di complemento. Ciascun bit del disegno di destinazione che corrisponde a un bit impostato a 1 nel parametro FgPen viene complementato: i bit 0 diventano 1 e i bit 1 diventano 0. Da ciò risulta ovvio che specificando due fasi di disegno in modo COMPLEMENT si ritornerà alla situazione originaria, dato che con la seconda operazione si ottiene l'inversione della prima.

L'argomento drawmode è definito come JAM1 o JAM2 nei casi semplici. Impostazioni di disegno che coinvolgano COMPLEMENT o INVERSVID sono in

genere scritte con istruzioni composte; ciò significa che un'impostazione di disegno in genere è scritta come:

COMPLEMENT | JAM1

oppure

COMPLEMENT | JAM2

La barra verticale, in queste espressioni, rappresenta semplicemente l'operatore logico OR. L'ultima impostazione, per esempio, informa la funzione SetDrMd d'impiegare la combinazione di JAM2 e COMPLEMENT.

Il quarto modo di disegno è INVERSVID. Questo modo dev'essere per forza usato in combinazione con JAM1 oppure con JAM2 e viene utilizzato principalmente per il testo. Se il modo di disegno composto è:

JAM1 | INVERSVID

i caratteri del testo appaiono come lettere trasparenti contornate con il colore FgPen. Invece, se il modo di disegno composto viene impostato come:

JAM2 | INVERSVID

i caratteri del testo appaiono nel colore di fondo contornato dal colore FgPen. La sola differenza tra questo modo e il precedente è che per disegnare il carattere viene usato BgPen. Da questa spiegazione si può osservare che il modo INVERSVID sostanzialmente inverte il ruolo e l'effetto dei colori FgPen e BgPen.

SetDrPt

Sintassi di chiamata della macro

SetDrPt (rastPort, linePtrn)
AØ DØ

Scopo della macro

Questa macro imposta nella struttura RastPort la matrice di continuità per le linee disegnate con le funzioni Draw, Move e PolyDraw.

Argomenti della macro

rastPort	Indirizzo della struttura RastPort di controllo.
linePtrn	Matrice di continuità delle linee, espressa su una word.

Discussione

Quando si usano le funzioni Draw, Move, AreaEnd, RectFill o PolyDraw, si disegnano linee nella bitmap. Per essere più precisi, si riempiono le linee con informazioni specifiche costituite da matrici di continuità. Questo genere di matrice è denominato *matrice di continuità delle linee*. La struttura di controllo RastPort indicata come primo argomento contiene nel parametro LinePtrn la matrice di continuità in uso. Per definire in modo preciso l'aspetto di questa matrice si usa la macro SetDrPt, la quale fornisce in pratica il mezzo per definire tutte le matrici di continuità necessarie per un programma e per i suoi task grafici.

Le matrici di continuità delle linee sono limitate a una larghezza di 16 bit perciò possono essere definite con una singola word di memoria. Per esempio, 0xFFFF definisce una matrice di continuità nella quale i bit sono tutti impostati al valore 1, il che produce una linea continua. Se si usa 0x0000 si definisce una matrice di continuità nel quale tutti i bit sono a 0; ciò produce (se il modo di disegno è JAM2) un disegno di linea nel colore BgPen.

La chiamata alla macro SetDrPt imposta la definizione della matrice di continuità nella struttura RastPort. Quando poi si chiama una delle funzioni di disegno che producono linee utilizzando quella struttura RastPort come struttura di controllo, il parametro LinePtrn informa la funzione di disegno di quale matrice deve fare uso.

Si noti che si può disegnare un numero indeterminato di matrici di continuità delle linee per i task del programma grafico. Possono essere definite speciali matrici per tutte le necessità di definizione di linea: da linee continue a linee composte da punti oppure corrispondenti alla matrice grafica di fondo nelle diverse posizioni dello schermo. Si possono cambiare i colori prodotti da questi diversi pattern chiamando al momento opportuno nel proprio task le funzioni SetAPen e SetBPen. In tal modo si possono utilizzare diverse chiamate a SetDrPt seguite da Move, Draw o PolyDraw per disegnare tutte le linee della bitmap.

SetOPen

Sintassi di chiamata della macro

SetOPen (rastPort, pennum)
A1 DØ

Scopo della macro

Questa macro imposta il colore della penna di contorno (AOlPen) per il testo e i contorni delle aree in relazione con una certa struttura RastPort.

Argomenti della macro

rastPort	Indirizzo della struttura RastPort di controllo.
pennum	Numero di penna (tra 0 e 255).

Discussione

Ci sono due funzioni di disegno della libreria Graphics che riguardano specificamente la definizione delle penne di disegno nel sistema Amiga: SetAPen e SetBPen. Inoltre il sistema prevede la macro SetOPen. Le due funzioni e la macro consentono di definire le caratteristiche delle penne di disegno utilizzabili per definire i colori e le altre caratteristiche dello schermo video.

Il parametro AOlPen della struttura RastPort, chiamato anche OPen, è la penna di contorno. Nessuno dei modi JAM1, JAM2, COMPLEMENT e INVERSVID impiega questa penna.

AOlPen viene usata per due applicazioni specifiche: il riempimento di aree e il riempimento in modalità flood. Nel primo caso serve per il colore del contorno; nel secondo, l'area che subisce il flood viene riempita con il colore FgPen fino a che questo raggiunge un pixel che abbia il colore corrispondente a AOlPen, e in quel momento l'operazione di flood viene interrotta.

Per disabilitare il contornamento delle aree con AOlPen si usa la macro BNDRYOFF.

SetRast

Sintassi di chiamata della funzione

SetRast (rastPort, pennum)
A1 DØ

Scopo della funzione

Questa funzione assegna un certo colore di penna all'intero contenuto della bitmap specificata. Quando SetRast restituisce il controllo, tutti i pixel appartenenti alla bitmap hanno il valore della penna colore prescelta.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
pennum	Numero di penna (compreso tra 0 e 255).

Discussione

SetRast è la sola funzione di disegno che consente d'impostare il colore di un'intera bitmap con una sola operazione.

Occasionalmente si troverà necessario impostare preventivamente il colore di un'intera bitmap che si trovi sotto il controllo di una certa struttura RastPort. Questo può essere necessario per impostare in un disegno il colore più diffuso. Lo scopo della funzione SetRast è proprio questo. L'operazione di disegno svolta da SetRast viene eseguita in accordo con l'impostazione di FgPen (determinata dall'ultima chiamata alla funzione SetAPen) e con l'impostazione del modo di disegno (determinata dall'ultima chiamata alla funzione SetDrMd).

Il sistema di coordinate della bitmap può gestire una bitmap di 1024 x 1024 pixel. La posizione 0,0 dell'origine è l'angolo superiore sinistro della bitmap. Secondo tale convenzione, i valori positivi di x vanno verso destra, mentre i valori positivi di y vanno verso il basso.

Si tenga sempre presente questo quadro di riferimento. Non si confonda il sistema di coordinate della bitmap con il sistema di coordinate dello schermo

video, che è limitato a 640 x 512 pixel in alta risoluzione con interlace (nel sistema PAL); quest'ultimo sistema di coordinate ha il punto di riferimento 0,0 nell'angolo superiore sinistro dello schermo video fisico.

Dal momento che la bitmap può essere così estesa, si possono sviluppare da essa diverse viewport e view. Perciò, se le si assegna un colore specifico con la funzione SetRast, ciascuna delle sue viewport e view assume quello stesso colore finché non verranno effettuate ulteriori modifiche con altre funzioni d'impostazione del colore.

SetRGB4

Sintassi di chiamata della funzione

```
SetRGB4 (viewPort,    n,    red_intensity, green_intensity, blue_intensity)
          A0          D0    D1             D2               D3
```

Scopo della funzione

Questa funzione inserisce il valore RGB di definizione di un colore nell'array di colori relativo alla viewport specificata. Nel registro colore selezionato, la nuova impostazione RGB interesserà dodici bit.

Argomenti della funzione

viewPort	Indirizzo della struttura ViewPort.
n	Numero del registro colore da modificare; n è compreso tra 0 e 31.
red_intensity	Numero intero compreso tra 0 e 15 che definisce l'intensità del rosso nel registro di colore.
green_intensity	Numero intero compreso tra 0 e 15 che definisce l'intensità del verde.
blue_intensity	Numero intero compreso tra 0 e 15 che definisce l'intensità del blu.

Discussione

Ci sono quattro funzioni di disegno, nella libreria Graphics, che riguardano specificamente i colori di penna utilizzati nelle viewport: GetRGB4, LoadRGB4, SetRGB4 e SetRGB4CM. LoadRGB4 carica un insieme di valori di colore RGB (red, green, blue; rosso, verde, blu) da un opportuno array, consentendo di modificare i colori della tavola assegnata alle penne. GetRGB4 consente di leggere il valore del colore assegnato a una certa penna. Una volta che si conosce quel colore, è possibile usarlo o cambiarlo per definire i dettagli di disegno in una viewport. SetRGB4 consente d'impostare uno specifico registro di colore in una bitmap di viewport.

Le funzioni LoadRGB4 e SetRGB4 lavorano entrambe con una determinata viewport.

È importante collocare la funzione SetRGB4 in un contesto appropriato. Si ricordi che ciascuna chiamata a SetRGB4 imposta il valore di colore di uno solo tra i 32 registri di colore del sistema Amiga. Inoltre riguarda una sola viewport, quella specificata nell'argomento viewPort.

Si noti ancora che la funzione SetRGB4 permette di controllare ciascun colore primario (rosso, verde e blu) in modo individuale per la viewport e il registro colore considerati. Si possono variare separatamente i bit che regolano il rosso, il verde e il blu (gli argomenti red_intensity, green_intensity, blue_intensity) per il registro colore specificato, regolando così l'intensità di ciascun colore primario nelle sue 16 gradazioni. Se, per esempio, si vuole il rosso alla massima intensità, s'imposta l'argomento red_intensity a 15. Se si vuole il rosso all'intensità minima, lo stesso argomento va impostato a 0. Per gli altri due colori valgono le stesse considerazioni.

I registri di colore dell'Amiga impiegano soltanto 12 dei loro 16 bit; i bit da 12 a 15 non vengono utilizzati. Quando s'impostano i tre argomenti d'intensità colore della funzione SetRGB4, vengono impostati i bit corrispondenti in uno di questi registri. I bit da 11 a 8 controllano il rosso, quelli da 7 a 4 il verde, quelli da 3 a 0 il blu.

Se si confronta la funzione SetRGB4 con LoadRGB4 e GetRGB4, si notano alcune differenze. In particolare, si osserva che la funzione LoadRGB4 agisce sull'intera tavola colore, consentendo di richiamare in una volta sola tutti e 32 i registri di colore. La funzione GetRGB4 consente di determinare il valore del contenuto di un solo registro, ma non può comunque alterarlo.

La funzione SetRGB4 consente la regolazione fine dei colori di una viewport; consente cioè di cambiare le assegnazioni di colore di una viewport senza ridefinire l'intera tavola colore con la funzione LoadRGB4. L'assegnazione iniziale dei colori per la viewport va fatta utilizzando la funzione LoadRGB4 con ciascuna viewport. Se in seguito è necessario modificare uno o più registri di colore relativi a una viewport, si utilizza la funzione SetRGB4.

Tale procedura evita di dover ridefinire ogni volta l'intera tavola dei colori, e consente di risparmiare tempo nell'esecuzione del task. Il tempo spesso è un fattore critico: a volte si devono alterare le assegnazioni dei registri nel breve intervallo di tempo che trascorre durante la scansione di una singola linea tra due viewport. È chiaro che più breve è il codice da eseguire in questo intervallo,

Valore del registro	Colore prodotto	Valore del registro	Colore prodotto
$FFF	Bianco	$6FE	Blu cielo
$D00	Rosso mattone	$6CE	Blu chiaro
$F00	Rosso	$00F	Blu
$F80	Rosso-arancio	$61F	Blu vivo
$F90	Arancio	$06D	Blu scuro
$FB0	Arancio-oro	$91F	Porpora
$FD0	Giallo cadmio	$C1F	Violetto
$FF0	Giallo limone	$F1F	Magenta
$BF0	Giallo-verde	$FAC	Rosa
$8E0	Verde chiaro	$DB9	Marrone-rosso
$0F0	Verde	$C80	Marrone
$2C0	Verde scuro	$A87	Marrone scuro
$0B1	Verde foresta	$CCC	Grigio chiaro
$0BB	Verde-blu	$999	Grigio medio
$0DB	Verde acqua	$000	Nero
$1FB	Verde acqua chiaro		

Tavola 2.4: *Tipiche impostazioni dei registri di colore*

più alte saranno le possibilità di successo del task e del programma al quale esso è associato.

Impiegando il modo Hold And Modify, si possono produrre 4096 colori interpretando i valori assegnati ai primi 16 registri di colore secondo particolari convenzioni. Si veda la spiegazione del modo Hold And Modify nell'appendice B.

La Tavola 2.4 riporta alcuni dei valori più frequentemente assegnati ai registri di colore e il colore prodotto. È utile ricordare quali sono i valori estremi delle intensità di colore. Si osservi nella Tavola 2.4 che il valore zero ($000) fornisce il nero. Se risultano impostati tutti i bit del registro colore ($FFF), viene prodotto il bianco.

SetRGB4CM

Sintassi di chiamata della funzione

```
SetRGB4CM (colorMap, color_number, red_intensity, green_intensity,
           A0        D0            D1:4           D2:4
     blue_intensity)
     D3:4
```

Scopo della funzione

Questa funzione cambia un elemento nella definizione dei registri di colore nell'array puntato dalla struttura ColorMap. Si può usare SetRGB4CM per definire una tavola di colori prima di assegnarla a una viewport. Questa funzione non restituisce alcun valore.

Argomenti della funzione

colorMap	Indirizzo della struttura ColorMap nel cui array di colori si desidera alterare un valore RGB.
color_number	Numero del registro colore (da 0 a 31) da modificare.
red_intensity	Numero intero compreso tra 0 e 15 che definisce l'intensità del rosso nel registro di colore.
green_intensity	Numero intero compreso tra 0 e 15 che definisce l'intensità del verde.
blue_intensity	Numero intero compreso tra 0 e 15 che definisce l'intensità del blu.

Discussione

Ci sono vari modi per cambiare le definizioni dei registri di colore, al fine di produrre effetti particolari sullo schermo video dell'Amiga. Primo: si può impiegare la funzione LoadRGB4 per associare un insieme di definizioni di registri a una viewport, impiegando la struttura ColorMap per puntare a un

array di colori.

Secondo: si può utilizzare la funzione SetRGB4 per definire uno specifico registro di colore (specificando l'intensità del rosso, del verde, del blu) e assegnarlo a una determinata struttura ViewPort. Tutte le bitmap controllate da tale struttura ricorreranno a quella definizione.

Terzo: si può usare la funzione SetRGB4CM per impostare un colore direttamente all'interno di un particolare array definito da una struttura ColorMap. Si può poi usare questa nuova mappa di colori con qualsiasi struttura ViewPort. Quest'ultimo metodo fornisce il modo più diretto per alterare i colori in una palette.

SetWrMsk

Sintassi di chiamata della macro

```
SetWrMsk (rastPort, newmask)
          AØ        DØ
```

Scopo della macro

Questa macro imposta, per una specifica struttura RastPort, la maschera di scrittura che stabilisce su quali bitplane verranno effettuate le operazioni di scrittura.

Argomenti della macro

rastPort	Indirizzo della struttura RastPort di controllo.
newmask	Valore a 16 bit nel quale i sei bit meno significativi determinano quali bitplane verranno effettuate le operazioni di scrittura.

Discussione

Nella libreria Graphics ci sono quattro macro che riguardano specificamente la definizione dei parametri di controllo del disegno nella struttura RastPort: SetOPen, SetDrPt, SetAfPt e SetWrMsk. Tali macro integrano le altre funzioni

di disegno della libreria Graphics.

In genere le operazioni di scrittura agiscono in tutti i bitplane di una bitmap, senza restrizioni. Tuttavia ci sono occasioni in cui si vuole impedire che un'operazione di disegno cambi i bit in determinati bitplane. Questo è lo scopo della macro SetWrMsk. Essa assegna un valore specifico al parametro Mask di una struttura RastPort, in modo che quando verrà eseguita una qualsiasi funzione di disegno che ricorra a quella struttura RastPort, vengano modificati soltanto i bitplane della bitmap che hanno il bit corrispondente nel parametro Mask impostato a 1.

Per esempio, se la macro SetWrMsk imposta il parametro Mask a 1111111111110111, tutti i bitplane saranno interessati alle operazioni di disegno eccetto il bitplane 3. Si noti che per il modo video single-playfield risultano significativi solo i sei bit inferiori. I bit di ordine maggiore saranno eventualmente impiegati per evoluzioni future (aumento dei bitplane in una bitmap) se verranno aumentati i colori del sistema Amiga. Si ricordi inoltre che si può impiegare una semplice istruzione di assegnazione di parametro di struttura per modificare il parametro Mask in una struttura RastPort. Quest'ultimo metodo e l'impiego della macro SetWrMsk sono equivalenti.

SyncSBitMap

Sintassi di chiamata della funzione

SyncSBitMap (layer)
AØ

Scopo della funzione

Questa funzione copia il contenuto di una bitmap di layer nella superbitmap associata al layer.

Argomenti della funzione

layer	Indirizzo della struttura Layer "bloccata" dal task.

Discussione

Ci sono due funzioni della libreria Graphics che riguardano specificamente le superbitmap e i layer a esse associati: CopySBitMap e SyncSBitMap. Queste funzioni operano con una struttura BitMap che definisce una superbitmap, e con una struttura Layer che definisce una bitmap di layer. Le funzioni CopySBitMap e SyncSBitMap utilizzano entrambe l'argomento puntatore layer, che punta alla struttura Layer alla quale è associata la superbitmap considerata. Nella struttura Layer esiste un puntatore alla struttura BitMap che definisce la superbitmap associata al layer.

La tecnica di refresh dello schermo tramite superbitmap è uno dei tre modi per ricostruire le porzioni coperte dello schermo video dell'Amiga. Gli altri metodi sono il refresh semplice, che non impiega bitmap di riserva, e il refresh avanzato, che utilizza una serie di piccole bitmap di riserva assegnate dal sistema.

Se si progetta un layer a refresh avanzato, il sistema assegna automaticamente un insieme di piccoli blocchi RAM per conservare le porzioni coperte della bitmap di layer. Non viene "tenuto di riserva" l'intero layer (tanto le porzioni nascoste quanto quelle visibili) come accade con un layer di superbitmap, ma soltanto le porzioni nascoste.

La tecnica di refresh di superbitmap differisce dalla tecnica di refresh avanzato per diversi motivi. Primo, viene usata una sola bitmap di riserva. Secondo, bisogna assegnare esplicitamente la memoria RAM per la superbitmap di riserva relativa al task grafico (a questo scopo si può impiegare, ad esempio, la funzione AllocRaster). Terzo, la superbitmap di riserva può essere più estesa della bitmap di layer a essa associata. Per vedere una porzione più estesa di una superbitmap nella bitmap di layer sullo schermo, si utilizza la funzione SizeLayer. Per vedere una porzione diversa della superbitmap nella bitmap di layer sullo schermo, si utilizza la funzione ScrollLayer. Entrambe si trovano nella libreria Layers, che viene trattata nel capitolo 5.

Si ricorre alla funzione SyncSBitMap quando è necessario predisporre un'intera bitmap di layer di riserva. Per farlo si deve copiare l'intera bitmap in un'appropriata sotto-sezione della superbitmap associata. È il sistema stesso che si occupa di collocare opportunamente la bitmap di layer all'interno della superbitmap. Questo è esattamente il compito della funzione SyncSBitMap. Si noti che questa operazione è l'opposto di quanto avviene con la funzione CopySBitMap, che copia parte di una superbitmap nella bitmap di layer a essa associata. Si veda anche la spiegazione della funzione CopySBitMap.

UnlockLayerRom

Sintassi di chiamata della funzione

UnlockLayerRom (layer)
A5

Scopo della funzione

Questa funzione sblocca la struttura Layer indicata come parametro. Una chiamata alla funzione UnlockLayerRom dovrebbe corrispondere a una precedente chiamata alla funzione LockLayerRom. UnlockLayerRom decrementa il contatore di blocco, presente nella struttura Layer. Quando tutti i task che hanno bloccato un layer hanno anche provveduto a sbloccarlo, il contatore di blocco contiene il valore zero. Quando una struttura Layer è completamente sbloccata, qualsiasi task può modificarla e alterare la bitmap di layer a essa associata.

Argomenti della funzione

layer	Indirizzo della struttura Layer.

Discussione

Ci sono tre funzioni video della libreria Graphics che riguardano specificamente le operazioni in multitasking e le strutture Layer: AttemptLockLayerRom, LockLayerRom e UnlockLayerRom. Queste funzioni hanno lo scopo di bloccare e sbloccare le strutture Layer impiegate per definire una presentazione video multi-layer. Una volta che un task ha bloccato una struttura Layer, gli altri non possono modificarla. Ciò è fondamentale in un sistema multitasking, in cui più di un task può tentare di definire informazioni grafiche nella stessa bitmap di layer. Quando una struttura Layer viene bloccata, nessun task, a parte quello che ha eseguito il blocco, può modificare i contenuti della bitmap.

Inoltre nessun altro task può alterare i parametri della corrispondente struttura Layer. Queste funzioni posseggono un solo argomento costituito dal puntatore alla struttura Layer; il suo valore risulta noto da una precedente chiamata alle funzioni CreateUpfrontLayer o CreateBehindLayer (descritte nel capitolo 5).

L'interfaccia grafica Intuition costituisce il più evidente impiego di layer nel sistema grafico dell'Amiga. Una volta che ci si trova in Intuition, si può utilizzare il mouse per aprire diverse finestre sullo schermo video dell'Amiga. Si possono poi impiegare i gadget che Intuition mette a disposizione per agire sulle finestre. In particolare, si possono usare i gadget nell'angolo superiore destro di ogni finestra per portarla in primo piano sullo schermo, rendendola attiva, o per disporla dietro a tutte le altre finestre presenti sullo schermo.

Quando sotto il controllo di Intuition si effettuano queste operazioni, si ha a che fare con uno schermo video multi-layer.

Dal momento che ciascuna nuova finestra viene generata da una diversa bitmap di layer, le nuove finestre potrebbero sovrapporsi ad altre già visibili sullo schermo. Quando ciò accade, qualche porzione di finestra precedentemente visibile risulterà nascosta nella view presente sullo schermo. Altre parti verranno nascoste dagli spostamenti delle finestre. Tutti questi effetti vengono gestiti internamente tramite le funzioni della libreria Layers, a cui fa ricorso il sistema Intuition.

Tornando alla situazione precedente, con UnlockLayerRom ciascun task può creare e disegnare in qualsiasi layer. La sola limitazione è lo spazio RAM richiesto per le bitmap (si tenga presente che le informazioni di bitmap restano confinate nella chip RAM dell'Amiga). Ciascuna chiamata alla funzione LockLayerRom dovrebbe essere seguita, all'interno dello stesso task, da una chiamata alla funzione UnlockLayerRom. Così, quando il task conclude definitivamente la sua esecuzione, il layer viene sbloccato.

Se un task esegue una chiamata a LockLayerRom per un layer specifico, un altro task può ugualmente eseguire una chiamata a LockLayerRom o a UnlockLayerRom per quello stesso layer. Infatti, un task non può prevedere l'esatta sequenza che un altro task seguirà, una volta che abbia preso il controllo della macchina. Tuttavia le chiamate alle funzioni LockLayerRom o UnlockLayerRom effettuate dal secondo task non avranno conseguenze sullo stato del layer considerato. Esso rimarrà bloccato per tutti a eccezione del task che ha chiamato per primo LockLayerRom.

VBeamPos

Sintassi di chiamata della funzione

```
vertbpos = VBeamPos ()
DØ
```

Scopo della funzione

Questa funzione chiede all'hardware qual è la coordinata verticale del pennello elettronico e la restituisce al task chiamante. Il valore richiesto è il numero di linea di scansione in modo video senza interlace.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Ci sono tre funzioni della libreria Graphics che riguardano specificamente la posizione verticale del pennello elettronico di scansione video sullo schermo dell'Amiga: VBeamPos, WaitBOVP e WaitTOF. Tutte e tre le funzioni hanno lo scopo di rilevare la posizione del pennello, in modo da poter in seguito intraprendere azioni specifiche a seconda della posizione rilevata.

La conoscenza della posizione del pennello elettronico di scansione video risulta vitale per le operazioni del sistema grafico dell'Amiga. La funzione VBeamPos lavora insieme con le funzioni WaitBOVP e WaitTOF per utilizzare la posizione verticale del pennello elettronico di scansione video come una variabile che consenta il controllo dell'hardware video di sistema.

Ci sono diversi momenti chiave, nello sviluppo di un quadro video, in cui il sistema ha bisogno di conoscere la coordinata verticale del pennello. Il primo e più importante momento è quando il pennello elettronico raggiunge il fondo del quadro video e ritorna nella posizione di partenza per iniziare il quadro seguente. In questo lasso di tempo, il sistema hardware deve aggiornare le liste di istruzioni Copper che stabiliscono i contenuti video del quadro seguente. Le nuove istruzioni Copper provengono da tre sorgenti: quelle fornite dalla funzione MakeVPort, quelle fornite dalle routine di animazione e qualsiasi istruzione Copper che il programmatore abbia scritto per controllare i registri direttamente, senza ricorrere alle funzioni della libreria Graphics.

Se i contenuti del quadro video successivo differiscono da quelli del precedente, dovranno essere riscritti i registri dei chip dedicati al controllo video. Queste alterazioni di registri consentono al sistema di cambiare le caratteristiche (per esempio il modo video e la palette dei colori) della view seguente, di stabilire una nuova posizione per gli sprite (nelle animazioni), d'impostare il Blitter per le operazioni successive.

Si può utilizzare la posizione verticale del pennello elettronico di scansione restituita dalla funzione VBeamPos anche per le scelte del task in esecuzione. Per esempio, si possono impiegare le funzioni LoadRGB4 o SetRGB4 per cambiare i valori assegnati ai registri di colore del sistema quando il pennello elettronico si trova tra due viewport confinanti verticalmente sullo schermo.

Questo è il motivo per cui le viewport devono avere almeno una linea di scansione che le separi: questa linea consente ai registri di controllo hardware di avere il tempo di effettuare i cambiamenti, in modo che la viewport seguente possa mostrare i nuovi colori selezionati.

Il valore della posizione verticale del pennello elettronico può anche essere utilizzato per decidere quando modificare uno sprite con la funzione ChangeSprite o quando rilasciarlo con la funzione FreeSprite (si veda il capitolo 3). In generale, il valore in questione s'impiega per decidere quando devono essere alterate le impostazioni hardware del sistema.

WaitBlit

Sintassi di chiamata della funzione

WaitBlit ()

Scopo della funzione

Questa funzione serve per arrestare l'attività di un task in esecuzione fino a quando non siano state soddisfatte tutte le richieste di movimento dati sottoposte al Blitter. Si ricordi che il Blitter viene impiegato per porre in coda, sotto il controllo delle funzioni QBlit e QBSBlit, diverse richieste in attesa di esecuzione.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Nella libreria Graphics ci sono dieci funzioni che riguardano specificamente le operazioni del Blitter nel sistema Amiga: BltClear, BltPattern, BltBitMap, BltTemplate, ClipBlit, DisownBlitter, OwnBlitter, QBlit, QBSBlit e WaitBlit. Si vedano anche le spiegazioni di queste funzioni.

La funzione WaitBlit restituisce il controllo al task quando il Blitter ha terminato il trasferimento dati in corso (o immediatamente, se il Blitter è inattivo). La funzione viene normalmente usata quando il task utilizza il Blitter

in maniera sincrona, ad esempio tramite le funzioni OwnBlitter e DisownBlitter. WaitBlit *non* attende, infatti, che la coda formata dalle funzioni QBlit e QBSBlit sia vuota prima di restituire il controllo.

WaitBOVP

Sintassi di chiamata della funzione

WaitBOVP (viewPort)
A0

Scopo della funzione

Questa funzione mette il task in attesa che il pennello elettronico raggiunga la fine della viewport indicata come argomento.

Argomenti della funzione

viewPort	Indirizzo della struttura ViewPort.

Discussione

Nella libreria Graphics ci sono tre funzioni che riguardano specificamente la posizione verticale del pennello elettronico: VBeamPos, WaitBOVP e WaitTOF. Tutte e tre le funzioni hanno lo scopo di rilevare la posizione del pennello elettronico sullo schermo e d'intraprendere azioni specifiche in relazione alla posizione rilevata. Si vedano anche le spiegazioni delle funzioni VBeamPos e WaitTOF.

Uno dei momenti chiave per l'alterazione delle impostazioni grafiche è l'intervallo di tempo tra la fine della visualizzazione di una viewport e l'inizio della successiva. Si ricordi che l'Amiga richiede almeno una linea di scansione tra due viewport.

Mentre il pennello elettronico passa su questa linea, il sistema aggiorna i registri di controllo hardware che definiscono le caratteristiche della nuova viewport da visualizzare. Questo si verifica quando si altera la risoluzione, la palette, gli sprite o altre caratteristiche della viewport.

Lo scopo della funzione WaitBOVP è di attendere che il pennello elettronico

raggiunga il fondo della viewport specificata. In quell'istante il task riottiene il controllo e può così procedere con un insieme di istruzioni per fissare le impostazioni hardware relative alla viewport successiva. La funzione WaitBOVP sospende quindi l'esecuzione del task fino a quando il pennello elettronico non raggiunge l'ultima linea della viewport indicata. A quel punto le istruzioni del task alterano le caratteristiche dello schermo video per la nuova viewport da visualizzare. Quasi tutte le caratteristiche della viewport possono essere alterate in questo modo; una cosa che non è possibile fare, tuttavia, è cambiare la risoluzione verticale tra una viewport e l'altra nella stessa view. Il flag LACE della struttura View controlla la risoluzione verticale di tutte le viewport comprese nella stessa view.

WaitTOF

Sintassi di chiamata della funzione

WaitTOF ()

Scopo della funzione

Questa funzione attende che il pennello elettronico di scansione video raggiunga la sommità del successivo quadro video. Ciò si verifica non appena è stata completata la sequenza di vertical-blanking. WaitTOF attende infatti il completamento dell'esecuzione di tutte le routine di vertical-blanking che si verificano durante l'intervallo.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Ci sono tre funzioni video della libreria Graphics che riguardano specificamente la posizione verticale del pennello elettronico sullo schermo dell'Amiga: VBeamPos, WaitBOVP e WaitTOF. Tutte e tre hanno lo scopo di rilevare la posizione sullo schermo del pennello elettronico di scansione video, in modo da poter intraprendere azioni specifiche a seconda della posizione rilevata.

Le funzioni VBeamPos e WaitBOVP consentono di alterare le caratteristiche dello schermo al raggiungimento di una certa posizione verticale da parte del pennello elettronico. La funzione WaitTOF si spinge un po' più avanti: consente a un task di attendere che il pennello elettronico raggiunga la cima del quadro successivo.

Questa procedura assicura che un complesso insieme di istruzioni Copper conduca a una massiccia alterazione dei registri di controllo prima che cominci effettivamente la presentazione del quadro video successivo. Si ricordi che a volte il sistema deve effettuare parecchi cambiamenti nei registri di controllo e altre impostazioni hardware nel corso dell'intervallo di vertical-blanking. La funzione WaitTOF sospende l'esecuzione del task fino a che tutte le impostazioni hardware di sistema non sono state effettuate.

Se le alterazioni del quadro devono essere maggiori, il sistema avrà bisogno di un tempo leggermente più lungo per cambiare le impostazioni hardware. Per questa ragione si ritarda l'esecuzione di ulteriori istruzioni del task fino a quando non si è certi che tutti i cambiamenti sono stati effettuati. Per esempio, una scelta di menu effettuata all'interno di un programma può condurre all'apertura di un insieme completamente nuovo di finestre. In questo caso il sistema ha bisogno di un certo tempo per definire i registri di controllo e le altre impostazioni hardware necessarie per visualizzare tutte le finestre. Un programma che funziona sotto Intuition sospende temporaneamente l'esecuzione fino a quando l'hardware di sistema non è adeguatamente impostato.

Ci sono parecchi particolari che si dovrebbero conoscere sul coprocessore Copper e sul suo impiego all'interno del sistema. Il Copper è un coprocessore video dedicato. Ciò significa che possiede un proprio insieme di istruzioni. Come avviene per le istruzioni CPU 68000, le istruzioni Copper sono scritte e conservate in RAM. Il Copper possiede soltanto tre istruzioni: WAIT, SKIP e MOVE. Esattamente come accade per gli altri elementi hardware dedicati a scopi speciali, le istruzioni Copper devono essere memorizzate nella chip RAM.

Queste istruzioni Copper consentono, ad esempio, di cambiare la palette dei colori a metà schermo, suddividendo lo schermo in sezioni multiple orizzontali (per esempio le finestre di Intuition o le viewport di una view), ciascuna delle quali possiede una propria risoluzione orizzontale; permettono inoltre di cambiare la profondità della bitmap, di produrre interrupt sincronizzati con il pennello per il 68000 e altro ancora.

Una delle istruzioni chiave del Copper, WAIT, serve per attendere che il pennello elettronico raggiunga una specifica posizione verticale per poi spostare informazioni (con l'istruzione Copper MOVE) nei registri per scopi speciali.

Durante il periodo di attesa, il Copper continua a esaminare la posizione del pennello elettronico. Ciò è importante per l'efficienza del sistema perché significa che mentre il Copper attende che il pennello raggiunga una certa posizione non impegna in alcun modo il bus di memoria. Durante l'attesa il bus risulta libero per l'impiego da parte di altri canali DMA o da parte della CPU. Una volta che la condizione di attesa è stata soddisfatta, il Copper sottrae cicli di memoria al Blitter o al 68000 per spostare i dati richiesti verso i registri indicati dalle istruzioni Copper.

Le istruzioni Copper lavorano in simultanea con le istruzioni del 68000. La maggior parte delle volte queste istruzioni non competono per i cicli di memoria disponibili nel sistema. Il 68000 funziona a 7,09 MHz (sistemi europei) ed è costretto a usare la memoria RAM a cicli alterni, precisamente quelli con numero pari. Ciò significa che il Copper può usare i cicli di memoria con numero dispari per svolgere il suo lavoro. In tale situazione il 68000 funziona al massimo delle sue possibilità. Solo in rare circostanze il Copper (o, per quel che riguarda questa discussione, il Blitter) sottrae cicli di memoria al 68000. Accade soltanto quando possono svolgere un lavoro particolare (per esempio, un trasferimento di dati) più velocemente del 68000.

WritePixel

Sintassi di chiamata della funzione

```
WritePixel (rastPort,     x,     y)
            A1            DØ     D1
```

Scopo della funzione

Questa funzione cambia il colore di un pixel della bitmap specificata, assegnandogli il colore fissato con la funzione SetAPen e gli attributi presenti nella struttura RastPort di controllo.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
x	Coordinata x del pixel considerato.
y	Coordinata y del pixel considerato.

Discussione

Ci sono due funzioni di disegno della libreria Graphics che riguardano i singoli pixel delle bitmap: ReadPixel e WritePixel. Queste funzioni consentono di leggere e cambiare il colore di un pixel in una specifica bitmap.

Occasionalmente si rivela necessario impostare il colore assegnato a uno specifico pixel nella bitmap, eventualmente per controllare il disegno in una parte ristretta dello schermo. Questo è lo scopo della funzione WritePixel. L'azione di WritePixel si svolge in accordo con l'impostazione di FgPen derivante dall'ultima chiamata della funzione SetAPen e con il modo di disegno impostato dalla funzione SetDrMd.

XorRectRegion

Sintassi di chiamata della funzione

XorRectRegion (region, rectangle)
A0 A1

Scopo della funzione

Questa funzione svolge un'operazione di XOR bidimensionale tra un rettangolo di delimitazione e una regione, lasciando il risultato nella regione. XorRectRegion esclude dalla regione qualsiasi area comune al rettangolo di delimitazione e alla regione. In pratica, nella nuova regione le parti che si sovrapponevano sono rimosse.

Argomenti della funzione

region	Indirizzo della struttura Region.
rectangle	Indirizzo della struttura Rectangle.

Discussione

Ci sono sei funzioni di disegno della libreria Graphics che riguardano specificamente le regioni e i rettangoli di delimitazione: AndRectRegion, ClearRegion, DisposeRegion, NewRegion, OrRectRegion e XorRectRegion. Si vedano anche le spiegazioni di queste funzioni.

La funzione XorRectRegion svolge un'operazione di XOR fra una regione e un qualunque rettangolo di delimitazione. Il risultato è la definizione di una nuova regione nella quale vengono cancellate le parti comuni fra il rettangolo

di delimitazione e la regione. Si ricordi sempre che, come per le altre funzioni di gestione delle regioni, queste operazioni logiche vengono sempre effettuate a un puro livello geometrico.

Si supponga di voler disegnare due rettangoli monocromi che si sovrappongano parzialmente, in modo che nella parte di bitmap comune il colore sia quello di fondo. Ci sono almeno due modi per disegnare una figura simile.

Il primo è ovvio. Si tratta semplicemente di decidere la collocazione della figura nel layer, e di usare le funzioni AreaDraw, AreaMove e AreaEnd (o la funzione RectFill) per definirla e disegnarla. Si impiegano le funzioni di disegno (SetAPen, SetBPen...) per preparare la struttura RastPort in modo che controlli adeguatamente i colori, i contorni e gli altri elementi della figura. Poi, quando finalmente viene eseguita la funzione AreaEnd (o la funzione RectFill), la figura viene disegnata nella bitmap di layer.

In alternativa, si può usare la funzione XorRectRegion con una regione definita da due rettangoli di delimitazione e disegnare in tale porzione della bitmap di layer. Per farlo bisogna prima definire una struttura Rectangle adoperando quattro semplici istruzioni di programma per definire i vertici del primo rettangolo. Si chiama poi la funzione NewRegion per assegnare e impostare una nuova regione di misura zero. Si utilizza in seguito la funzione XorRectRegion o OrRectRegion per includere il rettangolo di delimitazione nella nuova regione. Si definisce poi il secondo rettangolo impiegando altre quattro istruzioni di programma che ne stabiliscono i vertici, in modo tale che si sovrapponga al primo in un angolo. A questo punto si chiama una seconda volta la funzione XorRectRegion per inserire il secondo rettangolo nella definizione di regione.

Ora disponiamo di una definizione di regione e dobbiamo agganciarla al layer inserendo l'indirizzo della struttura Region nel parametro DamageList della struttura Layer (il vecchio indirizzo contenuto in questo parametro lo salviamo temporaneamente, per poi ripristinarlo quando avremo terminato di disegnare nel layer).

Ora occorre eseguire una chiamata alla funzione BeginUpdate, la quale predispone il layer perché in fase di disegno si possa accedere solo alle parti danneggiate da altri layer e tornate alla luce. Dal momento che al posto della lista delle aree danneggiate abbiamo inserito provvisoriamente l'indirizzo della nostra regione, la funzione predispone il layer perché le funzioni di disegno accedano soltanto a essa. Questo significa che tutte le operazioni di disegno che verranno effettuate prima di chiamare la funzione EndUpdate disegneranno solo all'interno della regione che abbiamo creato (per le spiegazioni delle funzioni BeginUpdate e EndUpdate si veda il capitolo 5). Per essere precisi, le operazioni di disegno avverranno nei due quadrati aggiunti alla regione tramite la funzione XorRectRegion, ma non nella parte comune.

Ora, come in una normale procedura di disegno, si usano le funzioni di disegno (SetAPen, SetBPen...) per preparare la struttura RastPort del layer, perché controlli i colori per il disegno, i bordi eccetera, per il rettangolo da far apparire. Si usa poi la funzione RectFill per riempire il layer con il colore che si desidera. Normalmente, l'intera bitmap del layer riceverebbe questo colore, ma, avendo definito un'opportuna regione di clip, solo l'area della bitmap da

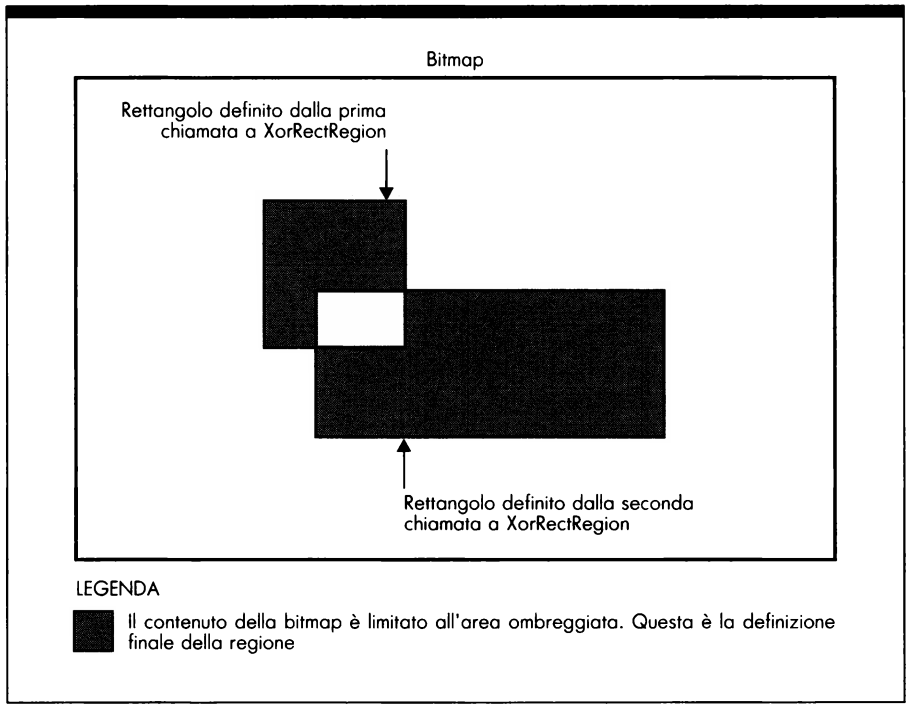


Figura 2.6: *Come opera la funzione XorRectRegion*

essa circoscritta sarà riempita con il colore prescelto. L'angolo in cui i due quadrati si sovrapponevano resta del colore di fondo.

Come mostra la Figura 2.6, si può estendere questo processo di disegno indiretto chiamando più volte la funzione XorRectRegion. Per esempio, prima di collegare la regione di clip al layer e chiamare la funzione BeginUpate, si può definire un altro rettangolo di delimitazione. Si può poi eseguire ancora la funzione XorRectRegion fra l'ultima regione ottenuta e il rettangolo di delimitazione per creare una nuova regione. Ovviamente, per giungere alla regione finale si possono anche chiamare le funzioni AndRectRegion e OrRectRegion oltre che XorRectRegion.

In conclusione, si noti che l'ordine con il quale i rettangoli di delimitazione vengono aggiunti alla regione non ha importanza.

XorRegionRegion

Sintassi di chiamata della funzione

```
success = XorRegionRegion (region1, region2)
D0                         A0       A1
```

Scopo della funzione

Questa funzione esegue un'operazione di XOR bidimensionale tra una regione e un'altra. XorRegionRegion esclude le parti comuni delle due regioni. Ciò rende possibile creare un buco in una regione. Il risultato viene lasciato nella seconda regione mentre la prima non viene alterata.

Argomenti della funzione

region1	Indirizzo della prima struttura Region.
region2	Indirizzo della seconda struttura Region.

Discussione

Ci sono quattro funzioni dell'Amiga che riguardano specificamente le regioni. Lo scopo e l'uso di questa funzione sono molto simili a quelli della funzione XorRectRegion.

Si ricordi che le regioni sono semplici insiemi di rettangoli di delimitazione. Le regioni vengono definite impiegando le funzioni AndRectRegion, OrRectRegion e XorRectRegion. La sola differenza tra XorRegionRegion e XorRectRegion è che la funzione XorRegionRegion combina due regioni mentre XorRectRegion combina una regione e un rettangolo di delimitazione. Si veda anche la spiegazione della funzione XorRectRegion.

Nel processo di disegno può accadere di ottenere due regioni di forma particolare (ma sempre con contorni rettilinei e paralleli) che combinate tra loro grazie alla funzione XorRegionRegion ne produrrebbero una terza che potrebbe essere utile in una successiva fase di disegno. La nuova regione risulterebbe "bucata" nelle zone comuni alle due regioni. Qualsiasi operazione di disegno successiva risulterebbe limitata al di fuori di tali buchi. Questo è lo scopo della funzione XorRegionRegion.

Le funzioni di animazione della libreria Graphics

Introduzione

Questo capitolo definisce e tratta le funzioni d'animazione contenute nella libreria Graphics dell'Amiga. Queste funzioni costituiscono un sistema esteso per la gestione di oggetti mobili rappresentati da sprite hardware, sprite virtuali, bob (oggetti del Blitter), componenti d'animazione e oggetti d'animazione.

Le funzioni d'animazione ricadono in otto categorie.

- Le funzioni per gli sprite hardware: ChangeSprite, FreeSprite, GetSprite e MoveSprite; usate per cambiare, rimuovere, aggiungere e spostare dal sistema sprite hardware.

- Le funzioni per gli sprite virtuali: AddVSprite, InitMasks e RemVSprite; usate per aggiungere, gestire e rimuovere sprite virtuali dalle liste di elementi grafici.

- Le funzioni e le macro per la gestione dei bob: le funzioni AddBob e RemIBob e la macro RemBob; usate per aggiungere e rimuovere bob dalle liste di elementi grafici.

- Le funzioni e le macro per gli oggetti d'animazione: le funzioni InitGMasks, AddAnimOb e la macro InitAnimate; usate per impostare e per aggiungere oggetti d'animazione alla relativa lista di sistema.

- La funzione Animate, che produce l'animazione effettiva sullo schermo video.

- Le funzioni per la gestione delle collisioni di elementi grafici con elementi grafici: DoCollision e SetCollision, usate per rilevare il verificarsi di collisioni tra elementi grafici e avviare l'esecuzione di routine di gestione delle collisioni definite dal programmatore.

- Le funzioni per la gestione della lista degli elementi grafici: InitGels, SortGList e DrawGList, usate per impostare, ordinare e definire gli elementi grafici nella relativa lista.

- Le funzioni per la gestione della memoria dedicata agli oggetti d'animazione: FreeGBuffers e GetGBuffers, usate per assegnare e liberare i buffer di memoria richiesti dagli oggetti d'animazione.

Gli elementi delle animazioni

Il sistema d'animazione dell'Amiga tratta con cinque elementi: gli sprite hardware, gli sprite virtuali, i bob, i componenti d'animazione e gli oggetti d'animazione. Segue una breve descrizione di ogni tipo di elemento.

Gli sprite hardware

Sono gli sprite propriamente detti, controllati dai relativi canali DMA. Se ne possono definire fino a otto. Ciascuno di essi non può essere più largo di 16 pixel ma può assumere l'altezza dell'intero schermo.

Gli sprite hardware sono accoppiati, per quanto riguarda i colori. Entrambi gli sprite della coppia avranno in comune i medesimi 4 colori. Uno di questi colori è sempre quello trasparente, per consentire l'osservazione dei pixel appartenenti ai playfield sottostanti. Gli sprite hardware sono utili per semplici effetti d'animazione che si servano di oggetti mobili elementari. Per questa ragione sono anche detti sprite semplici. Se sono necessari più di otto diversi oggetti mobili si dovranno impiegare gli sprite virtuali e i bob.

Ciascuno sprite hardware è definito dalle strutture SimpleSprite e SpriteImage. Per gestire gli sprite hardware si usano le funzioni ChangeSprite, GetSprite, FreeSprite e MoveSprite.

Attenzione: non si possono raggruppare gli sprite hardware in entità d'animazione di livello più alto, come i componenti d'animazione o gli oggetti d'animazione. E non si può nemmeno chiedere al sistema di controllare e registrare collisioni tra sprite hardware e altri oggetti del sistema.

Gli sprite virtuali

Gli sprite virtuali sono definiti associando la definizione software di uno sprite a un effettivo sprite hardware. Questo meccanismo di associazione tra sprite software e sprite hardware consente di definire un numero esteso di sprite virtuali e di creare sullo schermo sprite colorati in modo molto vario. Tuttavia si è soggetti al limite di otto sprite virtuali per una data coordinata y dello schermo. Come gli sprite hardware, gli sprite virtuali hanno una larghezza massima di 16 pixel e un'altezza massima pari all'intero schermo. Il vantaggio principale degli sprite virtuali rispetto a quelli hardware è la maggiore varietà di forme e di colori.

Ciascuno sprite virtuale risulta assegnato a un diverso canale di sprite hardware. In ogni determinato punto del quadro video, gli sprite virtuali vengono assegnati a sprite hardware non in uso. L'assegnazione è controllata dal sistema. Come per gli sprite hardware, gli sprite virtuali possono assumere quattro colori; uno dei quattro deve sempre essere il trasparente. Ogni sprite virtuale è definito da una struttura VSprite. Per la gestione degli sprite virtuali s'impiegano le funzioni AddVSprite e RemVSprite.

Gli sprite, tanto quelli virtuali quanto quelli hardware, non possono essere

riuniti in entità d'animazione di livello superiore, come componenti d'animazione o oggetti d'animazione. Tuttavia, al contrario di quanto accade con gli sprite hardware, si può chiedere al sistema di controllare e registrare collisioni tra sprite virtuali e altri oggetti.

I bob

I bob (Blitter objects, oggetti del Blitter) sono simili agli sprite virtuali se non per il fatto che, per disegnarli, il sistema impiega il Blitter anziché i canali DMA per gli sprite hardware. Un bob è una sezione rettangolare di una bitmap di playfield che può essere spostata da una bitmap all'altra oppure da una posizione all'altra nella stessa bitmap. Per farsi un'idea di come lavora il Blitter, si vedano le spiegazioni delle relative funzioni nel capitolo 2. Sebbene il Blitter sia molto veloce, uno sprite virtuale si muove molto più rapidamente di un bob con la stessa definizione d'immagine (i canali DMA per gli sprite hardware sono più veloci del Blitter).

Tuttavia i bob godono di tre vantaggi rispetto agli sprite virtuali. Il primo vantaggio è il numero dei colori che è possibile assegnare loro. Un bob può avere un numero di colori uguale a quello dello schermo in cui si trova, mentre uno sprite virtuale è limitato a quattro. Il secondo vantaggio è che il bob può avere qualsiasi larghezza, può quindi superare i 16 pixel degli sprite. Il terzo vantaggio è che i bob possono essere raggruppati per creare componenti d'animazione, che a loro volta possono essere riuniti per creare oggetti d'animazione.

I componenti d'animazione

I componenti d'animazione sono gruppi di bob collegati. Un esempio è costituito dalle figure di braccia, gambe, corpo e testa che insieme formano l'immagine di una persona intera. Ciascuno di questi elementi può essere definito come bob e poi collegato a un insieme che nel suo complesso costituisce un componente d'animazione. Ripetiamo che i componenti d'animazione usano soltanto i bob, non gli sprite hardware o virtuali. Ciascun componente d'animazione è definito da una struttura AnimComp, la quale possiede un insieme di parametri di temporizzazione per controllarne il comportamento quando il componente d'animazione viene usato come parte di un oggetto d'animazione. Non esistono funzioni specifiche per i componenti d'animazione; essi vengono semplicemente definiti concatenando fra loro le strutture Bob dei bob che li costituiscono.

Gli oggetti d'animazione

Gli oggetti d'animazione sono gli elementi più articolati del sistema: sono gruppi di componenti d'animazione collegati insieme. Il sistema può produrre il movimento di un oggetto d'animazione, ottenendo di conseguenza il movimento di tutti i componenti d'animazione che lo costituiscono. Con gli oggetti d'animazione si possono ottenere due tipi di disegno: le animazioni a disegni in sequenza e le animazioni a controllo di movimento.

Le liste del sistema d'animazione

Il sistema d'animazione mantiene diverse liste: la lista degli elementi grafici, la lista dei bob, la lista dei componenti d'animazione, la lista degli oggetti d'animazione. Segue una breve trattazione di ciascun tipo di lista.

Le liste di elementi grafici

Ciascuna lista di elementi grafici è a doppia concatenazione, costituita da tutti gli elementi grafici a essa pertinenti (sprite virtuali e bob). La lista degli elementi grafici è necessaria affinché la funzione SortGList possa disporli seguendo l'ordine crescente delle coordinate y,x; tale ordinamento dev'essere effettuato prima della chiamata alla funzione DrawGList. Questa disposizione è necessaria perché le istruzioni Copper generate dalla funzione DrawGList definiscono il quadro video dall'alto verso il basso e da sinistra verso destra.

Ciascuna lista di elementi grafici viene inizializzata con una chiamata alla funzione InitGels. Ciò aggiunge nel sistema una nuova lista attiva di elementi grafici. Ogni volta che viene chiamata la funzione AddBob, oppure la funzione AddVSprite, un nuovo elemento grafico viene inserito in una delle liste attive di elementi grafici. I parametri NextVSprite e PrevVSprite, presenti nella struttura VSprite, definiscono i collegamenti nella lista degli elementi grafici, tanto per gli sprite virtuali quanto per i bob.

Le liste di bob

Ogni lista di bob è formata da tutti i singoli bob che costituiscono un oggetto d'animazione. La lista determina la priorità di disegno (quali bob devono essere disegnati per primi e quali per ultimi) relativamente a un particolare oggetto d'animazione. L'ordine di disegno decide quali bob "coprono" gli altri sullo schermo video. I puntatori Before e After nella struttura Bob definiscono i collegamenti esistenti all'interno della lista dei bob. I puntatori di sistema DrawPath e ClearPath, presenti nella struttura VSprite, sono anch'essi correlati all'ordine di disegno dei bob.

Le liste di oggetti d'animazione

Ogni lista di oggetti d'animazione viene impostata con un insieme di semplici istruzioni di task o con una chiamata alla funzione InitAnimate. Entrambe queste procedure specificano il primo oggetto d'animazione nella lista degli oggetti d'animazione e creano una nuova lista attiva di oggetti d'animazione. Ogni volta che viene chiamata la funzione AddAnimOb, un nuovo oggetto d'animazione viene aggiunto a una delle liste attive. Una volta che l'oggetto d'animazione iniziale è stato definito, i puntatori NextOb e PrevOb, nella struttura AnimOb, definiscono i collegamenti interni in ciascuna lista.

Le liste di componenti d'animazione

Il sistema mantiene due tipi di liste di componenti d'animazione. La prima è quella che elenca i componenti d'animazione che fanno parte dello stesso oggetto d'animazione. La seconda definisce l'ordine di disegno dei componenti d'animazione. Ciascuna lista del primo tipo viene inizializzata quando sono stati definiti i componenti d'animazione di un determinato oggetto; l'operazione richiede che venga specificato il parametro di puntamento HeadComp della struttura AnimOb. Gli elementi che vanno ad aggiungersi in una lista di componenti d'animazione, sono definiti dai puntatori NextComp e PrevComp della struttura AnimComp (sono i parametri che definiscono i collegamenti).

Ciascuna lista di componenti d'animazione del secondo tipo aiuta a definire l'ordine di disegno nell'animazione a disegni in sequenza. I puntatori NextSeq e PrevSeq, presenti nella struttura AnimComp, impostano e definiscono i collegamenti in questo tipo di lista.

Le operazioni con gli sprite virtuali

Ecco un succinto sommario della procedura da seguire per le operazioni con gli sprite virtuali:

1. Si definisce una struttura View le cui istruzioni video per il Copper verranno in seguito unite alle istruzioni Copper per gli sprite virtuali.

2. Si imposta la struttura GelsInfo impiegando la funzione InitGels; questo passo è necessario soltanto una volta.

3. Si definiscono i parametri dello sprite virtuale nella struttura VSprite (vedere la funzione AddVSprite).

4. Si aggiunge lo sprite virtuale a una lista di elementi grafici impiegando la funzione AddVSprite.

5. Lo sprite virtuale viene presentato sullo schermo tramite le chiamate alle macro ON_DISPLAY e ON_SPRITE e alle funzioni SortGList, DrawGList, MrgCop e LoadView.

6. La forma dello sprite virtuale viene alterata cambiando il puntatore alla struttura ImageData, la sua altezza oppure il puntatore alla sua struttura dati per il colore.

7. Lo sprite virtuale viene mosso modificandone le coordinate in due parametri della struttura VSprite.

8. Si ripete il passo numero 5 per presentare, ancora una volta, lo sprite virtuale modificato o spostato.

Le operazioni con i bob

Ecco un succinto sommario della procedura da seguire per le operazioni con i bob:

1. Si definisce una struttura View le cui istruzioni Copper contengono le informazioni per presentare il bob.

2. Si imposta la struttura GelsInfo impiegando la funzione InitGels; questo passo è necessario una volta soltanto.

3. Si creano e si collegano una struttura Bob e una struttura VSprite con una semplice istruzione di collegamento di strutture.

4. Si definiscono i parametri della struttura Bob (si veda la funzione AddBob).

5. Si assegna spazio di memoria per i dati di ImageShadow.

6. Si inizializza il parametro puntatore DBufPacket, se si vuole impiegare il sistema double-buffer. Altrimenti si assegna a questo puntatore il valore zero.

7. Si chiama la funzione InitMasks per creare i dati ImageShadow per il bob.

8. Si aggiunge il bob a una lista di elementi grafici impiegando la funzione AddBob.

9. Il bob viene presentato attraverso le chiamate alle macro ON_DISPLAY e ON_SPRITE e alle funzioni SortGList, DrawGList, MrgCop e LoadView.

10. La forma del bob viene alterata cambiando il puntatore alla struttura ImageData oppure l'altezza, la larghezza o la profondità.

11. Si cambiano i colori del bob variando la definizione dell'insieme dei colori di playfield o i parametri PlanePick e PlaneOnOff, della struttura Bob.

12. Si ripete il passo numero 9 per presentare il bob spostato e alterato.

tipi d'animazione

Il sistema Amiga può effettuare due tipi d'animazione. L'*animazione di sprite*, che impiega gli sprite hardware e quelli virtuali; l'*animazione di playfield*, che si suddivide in tre sotto-categorie: l'animazione di bob singolo, l'animazione a controllo di movimento di bob multipli e l'animazione a disegni in sequenza di bob multipli.

Gli sprite virtuali non possono essere collegati insieme per definire forme geometriche più complesse. Gli sprite vengono spostati rispetto allo sfondo statico chiamato playfield. Sia gli sprite hardware sia quelli virtuali impiegano, per il loro movimento, i canali DMA degli sprite hardware. Questo tipo d'animazione implica alcune limitazioni. In particolare si rimane vincolati al movimento di una singola entità grafica; non è possibile definire entità complesse che presentino movimenti relativi tra le parti che le compongono. L'animazione di sprite può essere impiegata tanto nel modo single-playfield quanto in quello dual-playfield.

L'animazione di playfield è molto più potente dell'animazione di sprite: impiega i canali DMA del Blitter per spostare particolari aree del playfield chiamate bob. I bob creati per le animazioni di playfield hanno ciascuno una propria priorità video. Tale priorità viene controllata dai puntatori Before e After, presenti nelle strutture Bob che definiscono ciascun bob. Un bob costituisce una parte fisica del playfield. A mano a mano che i bob vengono mossi lungo lo schermo, il sistema conserva e ricostituisce le informazioni di background della bitmap contenute nei buffer individuati dalle strutture Bob.

L'effetto più interessante che i bob consentono di ottenere è l'unione in componenti d'animazione; questi ultimi possono, a loro volta, essere riuniti in oggetti d'animazione. Una volta che un oggetto d'animazione è stato definito è possibile impiegarlo in animazioni a controllo di movimento, oppure in animazioni a disegni in sequenza.

Le animazioni a disegni in sequenza richiedono che sia definita una sequenza di posizioni per gli oggetti d'animazione. Tali oggetti vengono quindi presentati sullo schermo seguendo l'ordine definito dalla struttura AnimOb. È anche possibile predisporre un insieme di variabili di temporizzazione per controllare la durata della permanenza sullo schermo di ciascun oggetto d'animazione.

Le animazioni a controllo di movimento richiedono che venga definita la posizione, la velocità iniziale e l'accelerazione degli oggetti.

Si vedano le spiegazioni delle funzioni e le definizioni delle strutture Bob, AnimComp e AnimOb per capire come si effettuano le animazioni a disegni in sequenza e a controllo di movimento. È anche possibile combinare questi due tipi d'animazione con routine definite dal programmatore in grado di rilevare e controllare il movimento.

Le routine d'animazione definite dal programmatore

Il sistema d'animazione dell'Amiga può essere largamente personalizzato realizzando routine predisposte dal programmatore e rivolte al controllo del movimento e alla gestione delle collisioni. Ciascuna struttura AnimComp possiede un parametro che punta all'indirizzo di una routine AnimCRoutine, definita dal programmatore, per il componente d'animazione. Allo stesso modo, ciascuna struttura AnimOb possiede un parametro che punta all'indirizzo di una routine AnimORoutine, sempre definita dal programmatore, per l'oggetto d'animazione. Entrambe queste routine vengono chiamate automaticamente quando viene eseguita la funzione Animate.

Si possono progettare queste routine per svolgere praticamente qualsiasi cosa. In particolare possono modificare le strutture Bob, AnimComp e AnimOb durante l'animazione. Basta prevedere una routine che controlli i valori contenuti in ciascuna struttura (posizione, velocità, accelerazione...) e impieghi poi un algoritmo particolare per alterarli.

In particolar modo si possono organizzare routine proprie per rilevare e cambiare le posizioni di bob, oggetti o componenti d'animazione.

Si possono anche usare queste routine per alterare i colori dei bob negli oggetti d'animazione o per produrre effetti sonori in determinate circostanze. Il noto programma dimostrativo con la palla che rimbalza producendo effetti sonori fornisce un buon esempio di ciò che può essere ottenuto con queste routine. Qualsiasi cambiamento di posizione si ottenga, verrà aggiunto ai cambiamenti prodotti dalle animazioni a disegni in sequenza o a controllo di movimento, a seconda delle definizioni dei parametri di posizione presenti nelle strutture VSprite, AnimComp e AnimOb.

Alle routine AnimCRoutine e AnimORoutine viene automaticamente passato un solo argomento, nel momento in cui vengono chiamate dalla funzione Animate. Questo argomento è un puntatore alle strutture di controllo AnimComp o AnimOb per leggere e scrivere i parametri che interessano le routine in questione.

Estensione delle strutture Bob e VSprite

Le strutture Bob e VSprite possiedono nella parte finale un insieme di parametri che possono essere impiegati per estendere la loro definizione. Questi parametri sono chiamati BUserStuff e BUserExt per la struttura Bob, e VUserStuff e VUserExt per la struttura VSprite.

Le estensioni definite dal programmatore possono essere impiegate per interfacciare l'attività di sistema con le routine AnimCRoutine e AnimORoutine. Possono anche essere usate per interfacciare l'attività di sistema con le routine di gestione delle collisioni trattate nel corso della spiegazione della funzione SetCollision.

Un semplice esempio può illustrare il meccanismo di questa procedura. Si supponga di avere una struttura chiamata MieVariabili che si vuole aggiungere alla struttura Bob. Ecco la definizione di MieVariabili:

```
struct MieVariabili {
SHORT xvelocità;
SHORT yvelocità;
SHORT xaccelerazione;
SHORT yaccelerazione;
};
```

Questa struttura consente di aggiungere i valori in uso di velocità e accelerazione di un bob al termine della struttura Bob. Quando poi il bob viene spostato come parte di un oggetto d'animazione, si possono aggiornare tali valori dopo ciascuno spostamento scrivendone di nuovi in questi parametri di struttura. Gli stessi valori possono poi essere usati dalle routine di gestione delle collisioni, AnimCRoutine o AnimORoutine. Per raggiungere tale risultato si deve inserire nel programma un'istruzione come:

```
#DEFINE BUserStuff struct MieVariabili
```

e in seguito:

```
struct Bob * mioBob;
...
mioBob -> BUserExt.xvelocità = 30;
mioBob -> BUserExt.yvelocità = 25;
```

e così via.

Una volta che ciò è stato fatto, la struttura Bob può ottenere una rilevazione personalizzata dei mutamenti di velocità e accelerazione nel programma d'animazione. Si può applicare lo stesso concetto anche alla struttura VSprite.

Strutture del sistema d'animazione

Il sistema d'animazione lavora con diverse strutture che consentono di definire, cambiare e comunque gestire tutti gli elementi che intervengono nelle animazioni.

La struttura SpriteImage contiene i bit di dati necessari per definire l'immagine di uno sprite hardware. Per la definizione della struttura SpriteImage si veda la spiegazione della funzione ChangeSprite.

La struttura SimpleSprite contiene l'altezza e i dati per il controllo della posizione richiesti per gestire l'immagine di uno sprite hardware. Per una definizione completa della struttura SimpleSprite si veda la spiegazione della funzione GetSprite.

La struttura VSprite contiene tutti i parametri necessari per definire uno sprite virtuale. Tali parametri includono i puntatori alla precedente e alla successiva struttura VSprite nella lista degli elementi grafici, oltre alla posizione dello sprite virtuale. La definizione della struttura VSprite si trova nelle spiegazioni della funzione AddVSprite.

La struttura Bob contiene tutti i parametri necessari per definire un bob. Tali parametri includono i puntatori per la definizione dell'ordine di disegno dei bob e un puntatore alla struttura VSprite associata alla struttura Bob. La definizione della struttura Bob si trova nella spiegazione della funzione AddBob.

La struttura AnimComp contiene tutti i parametri necessari per definire un componente d'animazione. Tali parametri includono i puntatori al precedente e al successivo componente d'animazione nella relativa lista; vi sono poi i puntatori alla precedente e alla successiva struttura AnimComp nella sequenza dei componenti d'animazione, i parametri di temporizzazione e i valori di traslazione spaziale. La definizione della struttura AnimComp si trova nella spiegazione della funzione Animate.

La struttura CollTable contiene un insieme di puntatori, fino a 16, relativi alle routine di gestione delle collisioni definite dal programmatore. Queste routine vengono chiamate quando il sistema automatico di verifica delle collisioni rileva un contatto tra due elementi grafici. La definizione della struttura CollTable si trova nella spiegazione della funzione SetCollision.

La struttura GelsInfo serve per la gestione della lista degli elementi grafici. Contiene un byte per indicare quali sprite hardware possano essere assegnati agli sprite virtuali. Contiene inoltre i puntatori alla prima e all'ultima struttura VSprite nella lista degli elementi grafici, oltre a un puntatore alla struttura CollTable. La definizione della struttura GelsInfo si trova nella spiegazione della funzione InitGels.

La struttura DBufPacket contiene tutte le informazioni necessarie per gestire la tecnica di double-buffer per i bob. La definizione della struttura DBufPacket si trova nella spiegazione della funzione GetGBuffers.

L'animazione a controllo di movimento

La progettazione di un programma che esegua animazioni a controllo di movimento consiste, in gran parte, nel gestire i collegamenti tra le strutture AnimOb, AnimComp, Bob e VSprite che definiscono gli oggetti e i componenti d'animazione, i bob e gli sprite virtuali relativi all'animazione considerata.

La Figura 3.1 mostra tutti i collegamenti che si possono instaurare tra questi quattro tipi di strutture (rappresentate da quattro grandi rettangoli).

La figura illustra la seguente situazione:

- il solo puntatore di struttura definito dal programmatore è HeadComp, che punta alla prima struttura AnimComp per un oggetto d'animazione. I puntatori PrevOb e NextOb sono definiti e controllati dal sistema.
- In ciascuna struttura AnimComp ci sono quattro puntatori che devono essere definiti per ottenere un'animazione a controllo di movimento: PrevComp, NextComp, HeadOb e AnimBob. Per le animazioni a disegni in sequenza si devono definire anche i puntatori PrevSeq e NextSeq.

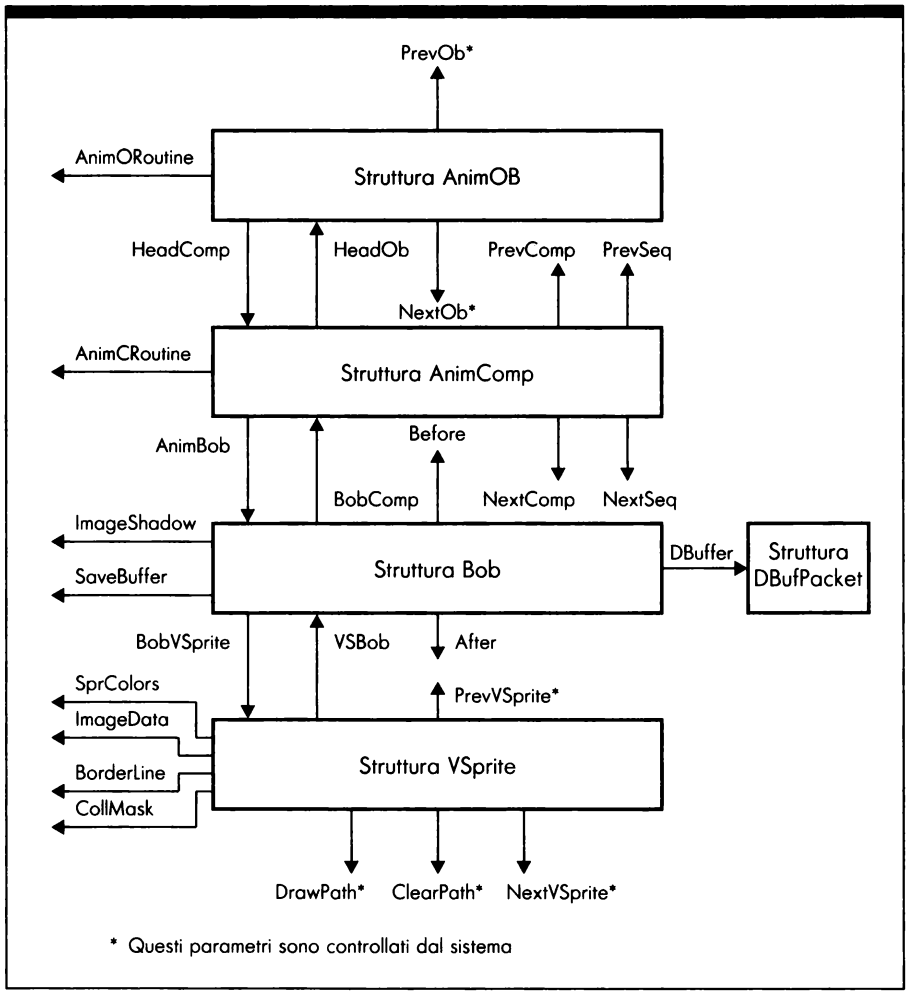


Figura 3.1: *Legami fra le strutture che intervengono nelle animazioni*

- Ci sono due puntatori in ciascuna struttura Bob che devono essere definiti tanto per l'animazione a controllo di movimento quanto per quella a disegni in sequenza: BobComp e BobVSprite. I puntatori Before e After sono opzionali; se non si provvede a specificarli, il sistema procede al disegno dei bob fissando esso stesso l'ordine di precedenza.

- Nella struttura VSprite c'è un solo parametro puntatore da definire per ciascuno sprite virtuale. Si tratta del parametro VSBob. Gli altri quattro parametri di puntamento (PrevVSprite, NextVSprite, DrawPath e ClearPath) presenti in ogni struttura VSprite, sono definiti e controllati dal sistema. Si noti che ciascuno di questi parametri controllati dal sistema, punta a una particolare struttura VSprite.

Oltre a questi parametri di puntamento, ci sono diversi parametri di controllo di posizione nelle strutture AnimOb, AnimComp e VSprite e che sono richiesti per un'animazione a controllo di movimento. Non esistono tuttavia parametri di controllo della posizione nella struttura Bob. Si vedano anche le spiegazioni delle relative funzioni. Facciamo per il momento una carrellata generale sui parametri.

- In ciascuna struttura AnimOb si devono definire i parametri AnY,AnX YVel,XVel e YAccel,XAccel. Tali coppie rappresentano la posizione di riferimento, la velocità e l'accelerazione dell'oggetto d'animazione. Si noti che per l'animazione a disegni in sequenza vengono impiegati due parametri addizionali: RingYTrans e RingXTrans. Gli altri parametri di controllo di posizione presenti nella struttura AnimOb sono definiti e controllati dal sistema.

- In ciascuna struttura AnimComp si devono definire i parametri YTrans e XTrans. Questa coppia costituisce la traslazione iniziale del componente d'animazione. Per l'animazione a disegni in sequenza vengono impiegati due parametri in più: Timer e TimeSet.

- In ciascuna struttura VSprite si devono definire i parametri per il controllo di posizione Y,X. Tale coppia rappresenta la posizione desiderata per lo sprite virtuale. Gli altri parametri per il controllo della posizione sono definiti e controllati dal sistema.

Oltre ai parametri di puntamento a strutture e per il controllo della posizione, queste quattro strutture d'animazione puntano anche a diverse altre aree di memoria, come mostra il lato sinistro dei rettangoli che rappresentano le strutture nella Figura 3.1. I puntatori a AnimORoutine e AnimCRoutine appartengono a questa categoria. Rimandiamo alla lettura delle singole funzioni per chiarimenti sul significato di ogni parametro.

Un esempio d'animazione a controllo di movimento

La Figura 3.2 mostra l'abbozzo del disegno di un robot. L'idea è quella di mettere la figura in grado di camminare e muoversi. Si tratta di un esempio d'animazione a controllo di movimento che impiega un oggetto d'animazione,

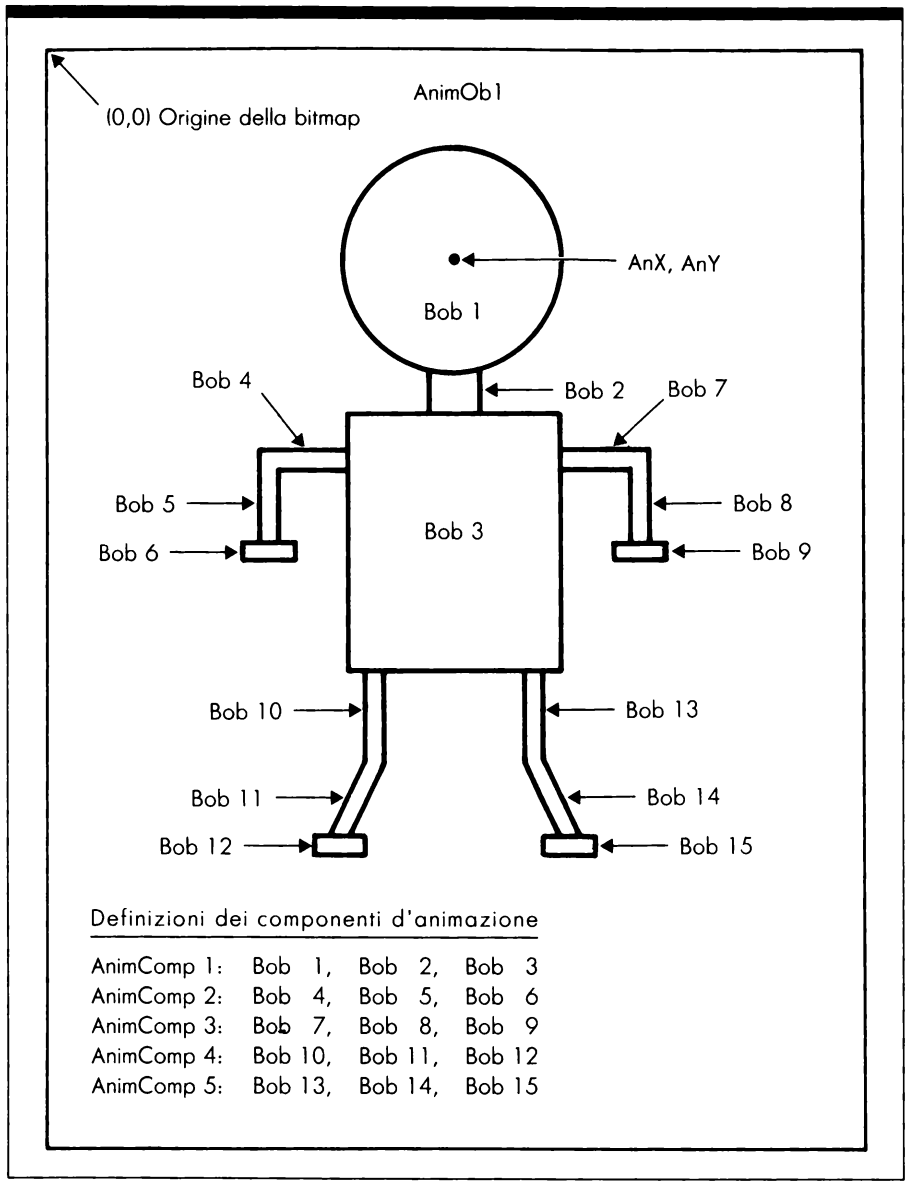


Figura 3.2: *Come raggruppare bob in componenti d'animazione e questi in oggetti d'animazione*

cinque componenti d'animazione, 15 bob e perciò 15 sprite virtuali. Le definizioni dei componenti d'animazione sono fornite nella figura.

AddAnimOb

Sintassi di chiamata della funzione

AddAnimOb (animOb, animKey, rastPort)
A0 A1 A2

Scopo della funzione

Questa funzione aggiunge una struttura AnimOb alla lista concatenata degli oggetti d'animazione.

Argomenti della funzione

animOb	Indirizzo della struttura AnimOb da aggiungere alla lista degli oggetti d'animazione.
animKey	Indirizzo di un puntatore alla prima struttura AnimOb nella lista degli oggetti d'animazione.
rastPort	Indirizzo della struttura RastPort di controllo.

Discussione

Nella libreria Graphics ci sono due funzioni e una macro che agiscono con gli oggetti d'animazione. Si tratta ripettivamente di AddAnimOb, InitGMasks e InitAnimate. Si vedano anche InitGMasks e InitAnimate.

I sistemi d'animazione appartenenti alla libreria Graphics per la gestione degli sprite virtuali e dei bob sono due. Esiste infatti il sistema degli elementi grafici, che agisce sia con gli sprite virtuali sia con i bob, e il sistema degli oggetti animati che riguarda soltanto i bob. Ciascuno di questi due sistemi mantiene liste proprie.

La funzione AddAnimOb aggiunge un nuovo oggetto d'animazione alla corrispondente lista di oggetti mantenuta dal sistema. L'argomento animKey

contiene l'indirizzo di un puntatore alla prima struttura AnimOb, in una specifica lista di oggetti d'animazione. AddAnimOb chiama la funzione AddBob per impostare ciascuna delle strutture Bob relative ai propri componenti. Inoltre AddAnimOb imposta a 0 tutti i parametri di tempo in ciascuna struttura AnimComp dell'oggetto d'animazione appena aggiunto. Tale operazione viene effettuata soltanto per i componenti d'animazione che appartengono all'oggetto appena aggiunto; non vale quindi per gli altri oggetti d'animazione già presenti nella lista degli oggetti d'animazione.

Si noti che la struttura RastPort dev'essere opportunamente inizializzata prima di effettuare la chiamata alla funzione AddAnimOb. La trattazione della funzione InitRastPort nel capitolo 2 contiene informazioni su come ciò possa essere ottenuto. In particolare, la struttura GelsInfo, presente nella struttura RastPort, deve risultare appropriatamente inizializzata. Tale risultato viene conseguito con la chiamata alla funzione InitGels, come viene descritto più avanti in questo stesso capitolo.

La struttura AnimOb

La definizione della struttura AnimOb è la seguente:

```
struct AnimOb {
    struct AnimOb *NextOb, *PrevOb;
    LONG Clock;
    WORD AnOldY, AnOldX;
    WORD AnY, AnX;
    WORD YVel, XVel;
    WORD YAccel, XAccel;
    WORD RingYTrans, RingXTrans;
    WORD (*AnimORoutine) ();
    struct AnimComp *HeadComp;
    AUserStuff AUserExt;
};
```

Il sistema imposta e impiega automaticamente nella struttura AnimOb i seguenti parametri:

- i parametri PrevOb e NextOb puntano al precedente e al successivo oggetto d'animazione in una lista di oggetti d'animazione. Si tenga presente che tale lista rimane distinta da quella degli elementi grafici. Questi parametri sono nulli per l'ultimo (NextOb nullo) e per il primo (PrevOb nullo) degli oggetti d'animazione della lista.

- Il parametro Clock contiene il numero delle chiamate alla funzione Animate avvenute per questo oggetto. All'inizio vale zero, e s'incrementa ogni volta che la funzione Animate elabora l'oggetto d'animazione.

- I parametri AnOldY e AnOldX contengono i valori precedenti delle coordinate y e x dell'oggetto animato. Si tratta dei valori precedenti

all'ultima chiamata alla funzione Animate. Tali parametri vengono misurati sulla risoluzione verticale e orizzontale della bitmap destinata a ricevere l'oggetto d'animazione in questione.

Tanto il sistema quanto i task grafici sono in grado d'impostare e usare i seguenti parametri appartenenti alla struttura AnimOb:

- i parametri AnY e AnX contengono le coordinate y e x dell'oggetto animato. Tali parametri vengono misurati sulla risoluzione verticale e orizzontale della bitmap destinata a ricevere l'oggetto d'animazione.

I task grafici predisposti dal programmatore possono impostare e impiegare i seguenti parametri della struttura AnimOb:

- i parametri YVel e XVel contengono le componenti y e x della velocità dell'oggetto d'animazione. Le velocità possono essere rappresentate da un numero positivo, negativo o dal valore zero. Il sistema tratta il valore a 16 bit della velocità come se fosse un numero o virgola fissa. Si considera che il punto decimale si trovi tra i bit 5 e 6 (i bit sono numerati da 0 a 15). Ogni chiamata alla funzione Animate aggiunge i valori di YAccel e XAccel ai rispettivi valori YVel e XVel. Si può usare simultaneamente l'animazione a disegni in sequenza e a controllo di movimento. In genere questi valori sono uguali a zero nel caso che l'animazione sia esclusivamente a disegni in sequenza.

- I parametri YAccel e XAccel contengono le accelerazioni dell'oggetto d'animazione. Tali valori possono essere positivi o negativi ma sono normalmente zero per animazioni esclusivamente a disegni in sequenza.

- I parametri YRingTrans e XRingTrans contengono gli incrementi della posizione di riferimento dell'oggetto d'animazione. Sono i valori y e x di traslazione per l'oggetto, e vengono misurati sulla risoluzione verticale e orizzontale della bitmap destinata a riceverlo. Tali valori sono aggiunti a quelli di AnY e AnX per produrre nuovi valori aggiornati in questi due parametri, in vista della presentazione del successivo quadro video. Consentono l'esecuzione di una sequenza di disegni che viene definita come un anello (ring); l'ultima posizione dell'anello coincide con la prima. Si noti che YRingTrans e XRingTrans hanno effetto soltanto quando viene impostato il flag RINGTRIGGER nel parametro Flags della struttura Bob e che devono essere utilizzati solamente per le animazioni a disegni in sequenza.

- Il parametro AnimORoutine contiene l'indirizzo di una speciale routine per l'oggetto d'animazione che verrà chiamata ogni volta che si esegue la funzione Animate. Se tale valore è zero non viene chiamata alcuna routine.

- Il parametro HeadComp punta alla prima struttura AnimComp dell'oggetto d'animazione definito dalla struttura AnimOb.

- Il parametro AUserExt può essere ridefinito a piacere dal programmatore modificando il tipo di variabile AUserStuff. Esso non viene usato dal sistema.

Si veda anche l'introduzione a questo capitolo.

AddBob

Sintassi di chiamata della funzione

AddBob (bob, rastPort)
A0 A1

Scopo della funzione

Questa funzione aggiunge una struttura Bob alla lista degli elementi grafici. AddBob imposta prima di tutto i flag appropriati nella struttura Bob e poi collega questa struttura Bob alla lista degli elementi grafici, tramite la funzione AddVSprite.

Argomenti della funzione

bob	Indirizzo della struttura Bob.
rastPort	Indirizzo della struttura RastPort di controllo.

Discussione

Ci sono due funzioni e una macro della libreria Graphics che lavorano direttamente con i bob (Blitter objects, oggetti del Blitter); si tratta rispettivamente di AddBob, RemIBob e RemBob. Si vedano anche le spiegazioni relative a RemIBob e RemBob.

Il sistema mantiene nella lista degli elementi grafici sia gli sprite virtuali sia i bob. Tale lista è diversa da quelle che il sistema mantiene per gli oggetti e per i componenti d'animazione. La funzione AddBob aggiunge una nuova struttura Bob alla lista degli elementi grafici.

La struttura Bob

Ecco la definizione della struttura Bob:

```
struct Bob {
    WORD Flags;
    WORD *SaveBuffer;
    WORD *ImageShadow;
    struct Bob *Before;
    struct Bob *After;
    struct VSprite *BobVSprite;
    struct AnimComp *BobComp;
    struct DBufPacket *DBuffer;
    BUserStuff BUserExt;
};
```

I task grafici possono impostare e impiegare tutti i parametri della struttura Bob. Tanto i task grafici previsti dal programmatore quanto quelli di sistema possono impostare e usare il parametro Flags. Ecco una descrizione di ciascuno dei parametri della struttura Bob.

- Il parametro Flags contiene un gruppo di flag che possono essere impostati e azzerati per controllare i vari aspetti del comportamento di un bob. I singoli flag sono elencati più avanti.

- Il parametro SaveBuffer punta al buffer in RAM da utilizzare per il salvataggio della porzione di bitmap sulla quale viene disegnato il bob (cioè lo sfondo). Quando il bob viene spostato, i dati presenti in questo buffer di sfondo vengono scambiati con la bitmap di schermo per ricostituire l'area del video dov'era disegnato il bob. La memoria per questo buffer viene assegnata e liberata dalle funzioni GetGBuffers e FreeGBuffers.

- I parametri Before e After puntano alle strutture Bob i cui bob vengono disegnati rispettivamente prima e dopo il bob considerato. Questi parametri vengono impiegati per assegnare priorità di disegno a un gruppo di bob che costituiscono un oggetto d'animazione.

- Il parametro BobVSprite punta alla struttura VSprite del bob, che definisce alcuni parametri dello stesso bob. In particolare, la struttura VSprite contiene le coordinate della sua posizione.

- Il parametro BobComp punta a un'eventuale struttura AnimComp. Il bob diventa parte del componente d'animazione rappresentato da questa struttura AnimComp.

- Il parametro DBuffer punta al buffer DBuffer del bob. Tale puntatore dev'essere nullo, a meno che non si voglia impiegare la tecnica

double-buffer. La memoria per questo buffer viene allocata e liberata dalle funzioni GetGBuffers e FreeGBuffers.

- Il parametro BUserExt può essere modificato dal programmatore per contenere dati d'interesse relativo alla struttura Bob.

Si veda anche l'introduzione a questo capitolo.

Presentiamo ora l'elenco dei singoli flag del parametro Flags della struttura Bob. Si noti che alcuni di questi flag sono definiti per la struttura Bob e alcuni per la struttura VSprite associata alla struttura Bob considerata.

- VSPRITE. Questo flag appartiene alla struttura VSprite. Va impostato se la struttura viene utilizzata per definire uno sprite virtuale; va azzerato se la struttura viene impiegata per definire un bob.

- SAVEBACK. Questo flag appartiene alla struttura VSprite. Va impostato se si vogliono salvare le informazioni della bitmap di sfondo nel punto in cui viene disegnato un bob. Se si sceglie di salvare lo sfondo si deve assegnare il buffer SaveBuffer con la funzione GetGBuffers. Ciò produce anche l'automatica ricostruzione dello sfondo quando il bob viene spostato.

- OVERLAY. Questo flag appartiene alla struttura VSprite. Va impostato se si vuole che il bob venga disegnato impiegando i dati di ImageShadow. In questo caso, si deve prima di tutto assegnare e impostare, tramite le funzioni GetGBuffers e InitMasks, l'array di dati ImageShadow. Se non s'imposta questo flag, il sistema usa l'intero rettangolo delle word di definizione del bob per disegnare sopra alla bitmap dello sfondo. Ciò significa che i colori del playfield (cioè i colori di sfondo) sono visibili per trasparenza in ogni punto in cui vi siano bit impostati a zero nella maschera d'ombra del bob.

- GELGONE. Questo flag appartiene alla struttura VSprite. Il sistema lo imposta quando un bob è stato spostato verso una regione di disegno al di fuori della regione di delimitazione (clipping-region). La regione di delimitazione per il bob è definita nella struttura GelsInfo. Per una definizione della struttura GelsInfo e della regione di delimitazione si veda la spiegazione della funzione InitGels.

- SAVEBOB. Questo flag appartiene alla struttura Bob. Se è impostato, l'immagine del bob non verrà cancellata durante gli spostamenti, e la bitmap finirà per contenere una sequenza d'immagini del bob. Con un'adeguata spaziatura può essere simulato un effetto tipo pennellata, tipico di molti programmi di disegno.

- BOBISCOMP. Questo flag appartiene alla struttura Bob. Va impostato qualora il bob considerato sia parte di un componente d'animazione. In

questo caso si deve anche definire il parametro di puntamento BobComp nella struttura Bob.

- BWAITING. Questo flag appartiene alla struttura Bob. Se è impostato, il bob resta in attesa di essere disegnato. Ciò si verifica soltanto quando il sistema trova un puntatore Before nella definizione della struttura Bob relativa al bob considerato. Il sistema azzera questo bit al ritorno da ogni chiamata alla funzione DrawGList.

- BDRAWN. Questo flag appartiene alla struttura Bob. È il sistema che provvede a impostarlo e azzerarlo. Se BDRAWN è impostato, significa che il bob è già stato disegnato. Il sistema può poi esaminare i puntatori Before e After in ciascuna struttura Bob per verificare la correttezza della sequenza di disegno per i bob. Il sistema azzera questo parametro al ritorno da ogni chiamata alla funzione DrawGlist.

- BOBSAWAY Questo flag appartiene alla struttura Bob. Può essere impostato e azzerato tanto dal sistema quanto dai task previsti dal programmatore. Va impostato qualora si voglia rimuovere il bob dalla lista degli elementi grafici quando viene chiamata la funzione DrawGList. In questo caso, il sistema ricostruisce le informazioni della bitmap di sfondo dove il bob è stato disegnato per l'ultima volta. Inoltre il sistema, quando viene di nuovo chiamata la funzione DrawGList (a meno che non venga impiegata la tecnica double-buffer), sgancia la struttura Bob dalla lista degli elementi grafici. Se invece s'impiega questa tecnica, il bob non viene rimosso dalla lista degli elementi grafici fino alla *seconda* chiamata della funzione DrawGList. A quel punto l'immagine del bob risulta rimossa da entrambi i buffer.

- BOBNIX. Questo flag appartiene alla struttura Bob. È il sistema che provvede a impostarlo e azzerarlo. Quando il bob è stato rimosso, il sistema imposta questo parametro al ritorno dalla chiamata alla funzione DrawGList. Ciò si verifica soltanto dopo che il bob è stato rimosso dalla lista degli elementi grafici e l'immagine della bitmap di sfondo è stata ricostituita. Questo parametro è particolarmente importante quando si impiega la tecnica double-buffer. In tal caso, infatti, il sistema dev'essere sicuro che l'immagine del bob sia stata rimossa sia dal buffer di disegno attivo sia dal buffer video.

- SAVEPRESERVE. Questo flag della struttura Bob viene impostato e azzerato dal sistema. Costituisce la versione double-buffer del flag SAVEBACK. SAVEPRESERVE viene usato dal sistema per indicare se il bob nel secondo buffer è stato ripristinato.

AddVSprite

Sintassi di chiamata della funzione

AddVSprite (vSprite, rastPort)
AØ A1

Scopo della funzione

Questa funzione aggiunge una struttura VSprite alla lista degli elementi grafici. La nuova struttura VSprite viene collocata nella lista seguendo i valori delle sue coordinate y,x. AddVSprite imposta anche i flag di sistema della struttura VSprite.

Argomenti della funzione

VSprite	Indirizzo della struttura VSprite.
rastPort	Indirizzo della struttura RastPort di controllo.

Discussione

Ci sono tre funzioni della libreria Graphics che lavorano direttamente con gli sprite virtuali: AddVSprite, InitMasks e RemVSprite. Si vedano anche le spiegazioni relative alle funzioni InitMasks e RemVSprite.

Il sistema mantiene nella lista degli elementi grafici tanto gli sprite virtuali quanto i bob. La funzione AddVSprite aggiunge un'altra struttura VSprite alla lista degli elementi grafici. Si noti che la funzione AddBob chiama indirettamente la funzione AddVSprite per aggiungere strutture Bob alla lista degli elementi grafici.

La struttura VSprite

Ecco la definizione della struttura VSprite:

```
struct VSprite {
        struct VSprite *NextVSprite;
        struct VSprite *PrevVSprite;
        struct VSprite *DrawPath;
        struct VSprite *ClearPath;
        WORD OldY, OldX;
        WORD Flags;
        WORD Y,X;
        WORD Height;
        WORD Width;
        WORD Depth;
        WORD MeMask;
        WORD HitMask;
        WORD *ImageData;
        WORD *BorderLine;
        WORD *CollMask;
        WORD *SprColors;
        struct Bob *VSBob;
        BYTE PlanePick;
        BYTE PlaneOnOff;
        VUserStuff VUserExt;
};
```

Il sistema imposta automaticamente e impiega i seguenti parametri della struttura VSprite:

- i parametri NextVSprite e PrevVSprite puntano rispettivamente alla successiva e alla precedente struttura VSprite della lista degli elementi grafici. Tutti gli sprite virtuali vengono conservati in ordine crescente di coordinate y,x.

- I parametri DrawPath e ClearPath puntano a due strutture VSprite. Questi parametri sono impiegati dal sistema per controllare il processo di disegno.

- I parametri OldY e OldX contengono la precedente posizione y e x dello sprite virtuale, cioè i valori impiegati per l'ultima chiamata della funzione Animate. Per i bob, questi parametri sono misurati sulla base della risoluzione verticale e orizzontale della bitmap che accoglie il bob. Per gli sprite virtuali il parametro OldY viene misurato in relazione al modo video senza interlace; il parametro OldX viene misurato sempre nel modo video a bassa risoluzione.

I task grafici possono impostare e impiegare i seguenti parametri:

- i parametri Y e X contengono le coordinate y e x dello sprite virtuale. Sono le coordinate che verranno utilizzate la prossima volta che verrà effettuata una chiamata alla funzione Animate. Per i bob, questi parametri si misurano sulla risoluzione verticale e orizzontale della bitmap che accoglie il bob. Per gli sprite virtuali il parametro Y viene misurato in relazione al modo video senza interlace; il parametro X viene misurato in relazione al modo video senza interlace in bassa risoluzione.

- I parametri Height e Width contengono l'altezza e la larghezza dell'immagine dello sprite virtuale. L'altezza viene misurata in termini di modo video senza interlace ed è compresa tra 0 e 256 pixel (sistema PAL). La larghezza invece viene ignorata se la struttura VSprite definisce uno sprite virtuale, dato che la larghezza degli sprite virtuali è sempre di 16 pixel. Per i bob, invece, questo parametro deve contenere la larghezza dell'immagine del bob espressa in numero di word.

- Il parametro Depth contiene il numero dei bitplane impiegati per definire i dati dell'immagine per lo sprite virtuale. Se la struttura VSprite definisce un bob, possono essere impiegati fino a cinque bitplane. Se la struttura definisce invece uno sprite virtuale, vengono sempre impiegate due word per definire ciascuna linea dell'immagine di sprite, e questo parametro viene ignorato.

- I parametri MeMask e HitMask contengono le maschere che determinano quale routine di gestione delle collisioni dev'essere chiamata quando due o più elementi grafici collidono. Il sistema esegue un AND logico tra il parametro HitMask dell'oggetto più in alto e a sinistra e il parametro MeMask dell'oggetto più in basso e a destra. I bit che rimangono impostati a 1 dopo l'operazione di AND determinano quale delle 16 routine di gestione delle collisioni dev'essere chiamata. A questo proposito si vedano le spiegazioni delle funzioni SetCollision e DoCollision.

- Il parametro ImageData punta a un insieme per l'immagine dello sprite virtuale considerato. I dati d'immagine sono costituiti da un insieme di word con almeno due word per ciascuna linea dell'immagine dello sprite virtuale. I bit in ciascuna di queste due word vengono combinati per determinare il colore assegnato a un particolare pixel dello sprite virtuale. I dati immagine per lo sprite virtuale sono molto simili ai dati della struttura SpriteImage che vengono impiegati per definire gli sprite hardware.

- Il parametro BorderLine punta a una locazione di memoria che contiene la combinazione logica OR di tutti i bit dei dati d'immagine di uno sprite virtuale. I dati delle linee di contorno vengono usati dal sistema per

rendere più veloce il processo di rilevamento delle collisioni. La memoria per questo buffer viene assegnata e liberata dalle funzioni GetGBuffers e FreeGBuffers.

- Il parametro CollMask punta a una posizione RAM contenente la maschera di collisione per lo sprite virtuale. I dati che definiscono una maschera di collisione coincidono in genere con i dati che definiscono una maschera d'ombra per lo stesso sprite virtuale. Entrambi sono costituiti come combinazione logica OR di tutti i bit impostati a 1 su tutti i bitplane dei dati d'immagine. La definizione del parametro ImageShadow della struttura Bob viene fornita nella spiegazione della funzione AddBob. La maschera di collisione è simile alla maschera di contorno, ma la prima è una matrice mentre la seconda è una word (per uno sprite virtuale) oppure un insieme di più word (per un bob). La memoria per il buffer della maschera di collisione viene allocata e liberata dalle funzioni GetGBuffers e FreeGBuffers.

- Il parametro SprColors punta al primo di tre valori a 16 bit conservati in sequenza. Questi valori formano una breve mappa dei colori, configurata come le tavole dei colori a 32 elementi che sono state trattate nel capitolo 2. Ciascuno di questi tre valori a 16 bit rappresenta il contenuto di un registro colore; i bit da 0 a 3 rappresentano l'intensità del blu, i bit dal 4 al 7 l'intensità del verde, i bit da 8 a 11 l'intensità del rosso. I bit da 12 a 15 non vengono impiegati. Il sistema adopera questi valori per determinare l'insieme dei tre colori fondamentali assegnati allo sprite virtuale. Questo parametro non viene usato se la struttura VSprite definisce un bob.

- Il parametro VSBob punta a una struttura Bob; viene usato solamente quando la struttura VSprite considerata definisce un bob.

- Il parametro PlanePick contiene una maschera a otto bit che seleziona i bitplane di un bob e viene usata per determinarne il colore. Questo parametro non viene usato con gli sprite virtuali.

- Il parametro PlaneOnOff contiene una maschera a otto bit che determina come trattare i bitplane non selezionati dal parametro PlanePick. Questo parametro non viene usato con gli sprite virtuali.

- Il parametro VUserText può essere ridefinito a piacere dal programmatore per includere nella struttura altri eventuali dati relativi allo sprite virtuale o al bob. Si veda anche l'introduzione di questo capitolo.

Tanto il sistema quanto i task grafici possono impostare e impiegare il parametro Flags della struttura VSprite, il quale può contenere i seguenti valori:

- VSPRITE. Questo flag va impostato se la struttura VSprite considerata definisce uno sprite virtuale; non va invece impostato qualora si stia definendo un bob.

- MUSTDRAW. Se viene impostato questo flag, il sistema sa che deve disegnare sullo schermo lo sprite virtuale rappresentato dalla struttura VSprite attiva nel momento in cui viene effettuata la chiamata alla funzione DrawGList. Se il sistema esaurisce gli sprite hardware che può assegnare a questo sprite virtuale, esso è in grado di convertirlo in un bob per questa sola chiamata alla funzione DrawGList. Se il sistema decide che deve provvedere a convertire uno sprite virtuale in un bob, i colori assegnati con il parametro SprColors della struttura VSprite non vengono impiegati. Per determinare i colori del bob vengono invece impiegati i parametri PlanePick e PlaneOnOff. Nota: nella versione 1.3 del sistema, questo flag non è ancora operativo.

- VSOVERFLOW. Questo flag viene trattato dal sistema. Viene impostato se il sistema osserva che il numero degli sprite virtuali definiti per una linea orizzontale dello schermo supera il numero degli sprite hardware disponibili. Tale numero massimo di sprite hardware di solito vale otto, a meno che non venga disposto altrimenti attraverso il parametro SprRsvd della struttura GelsInfo. Ciò significa che, a meno che non si imposti il bit MUSTDRAW per un particolare sprite virtuale, questo potrebbe non essere disegnato quando viene effettuata la successiva chiamata alla funzione DrawGList.

- GELGONE. Il sistema imposta questo bit quando un bob (o uno sprite virtuale) viene spostato interamente al di fuori della regione di delimitazione in uso. La regione di delimitazione è definita da quattro parametri nella struttura GelsInfo. Per la definizione di regione di delimitazione si veda la spiegazione della funzione InitGels. Dato che il sistema non disegna nessun bob o sprite virtuale che si trovi al di fuori della regione di delimitazione, è possibile rimuovere tale entità grafica dalla lista degli elementi grafici (questo per quanto riguarda la chiamata alla funzione DrawGList). Ciò rende più veloce l'elaborazione della lista degli elementi grafici. È importante rendersi conto che gli sprite virtuali o i bob rimossi non vengono più gestiti o controllati dal sistema.

La struttura VSprite contiene il parametro ImageData che punta ai dati d'immagine per lo sprite virtuale o per il bob. L'organizzazione dei dati non cambia molto se la struttura definisce sprite virtuali o bob. Tuttavia le differenze sono tali da richiedere una trattazione separata per i due casi.

I dati d'immagine per uno sprite virtuale vengono definiti come segue:

```
struct ImageData {
    UWORD sprdata[2][height];
};
```

sprdata[2][height] sono i dati effettivi che definiscono l'immagine dello sprite virtuale. Tali dati consistono in due word (indicate da [2]) per ciascuna linea dell'immagine dello sprite virtuale (il valore indicato dal parametro height). Se,

per esempio, si vuole definire uno sprite virtuale di una sola linea, si devono predisporre due word di dati. Se si vuole invece definire l'immagine di uno sprite virtuale che abbia l'altezza di uno schermo a 256 linee, si dovranno predisporre 512 word di dati.

Ciascuna linea dello sprite virtuale è definita da due word. I colori per ciascun pixel dello sprite virtuale sono determinati dalla combinazione dei bit provenienti da ciascuna di queste word. Illustriamo questa idea con un semplice esempio. Si supponga di voler definire uno sprite virtuale composto di due linee. I dati d'immagine vengono definiti nella struttura ImageData nel modo seguente:

```
linea 1 word 1:1111111111111111
linea 1 word 2:1111000000001111
linea 2 word 1:1111000000001111
linea 2 word 2:1111111111111111
```

Questo sprite virtuale è largo 16 pixel, il valore massimo tanto per gli sprite virtuali quanto per quelli hardware. Il colore per il pixel più a destra (cioè il pixel 0) è determinato dalla combinazione dei bit in posizione 0 della linea 1, rilevata dalle word 1 e 2. Per questa particolare immagine si ha il valore 11, che viene tradotto nel decimale 3. A questo pixel verrà perciò assegnato il colore 3. Ciò si verifica anche per i pixel da 1 a 3 e da 12 a 15 in entrambe le linee dello sprite; tutti questi riceveranno il colore 3.

Invece i pixel da 4 a 11 appartenenti alla linea 1 hanno il valore binario 01, che si traduce nel decimale 1. Questi pixel riceveranno il colore 1. I pixel da 4 a 11 della linea 2 hanno il valore binario 10, che si traduce nel decimale 2. Tali pixel riceveranno quindi il colore 2.

L'assegnazione effettiva dei registri ai colori da 1 a 3 viene determinata dal parametro SprColors della struttura VSprite. Il colore 0 corrisponde sempre alla trasparenza. Ogni pixel con questo colore lascerà vedere, in corrispondenza della sua posizione, il colore dello sfondo.

La struttura ImageData per i bob è simile alla struttura ImageData per gli sprite virtuali. Tuttavia i dati sono distribuiti in modo diverso. Supponiamo di voler definire un'immagine di un bob a tre linee da 16 pixel, con una profondità di tre bitplane (cioè con il parametro Depth della struttura VSprite uguale a 3). I dati d'immagine per definire questo bob sarebbero simili ai seguenti:

```
word 1 = linea 1, bitplane 1: 1111111111111111
word 2 = linea 2, bitplane 1: 1111000000001111
word 3 = linea 3, bitplane 1: 1111000000001111
word 4 = linea 1, bitplane 2: 1111000000001111
word 5 = linea 2, bitplane 2: 1111000000001111
word 6 = linea 3, bitplane 2: 1111000000001111
word 7 = linea 1, bitplane 3: 1111000000001111
word 8 = linea 2, bitplane 3: 1111000000001111
word 9 = linea 3, bitplane 3: 1111111111111111
```

Ciò mostra che i dati d'immagine del bob sono distribuiti in posizioni di memoria consecutive con i dati del bitplane 1 per primi, seguiti poi dai dati per il bitplane 2 e così via. Si noti che questa è la stessa disposizione di dati utilizzata per la struttura ImageData della struttura Image di Intuition. Si veda, a questo proposito, la spiegazione della funzione DrawImage nel capitolo 6.

I colori assegnati a ciascun pixel di questo bob vengono determinati dai parametri PlanePick e PlaneOnOff della struttura VSprite. Entrambi i parametri sono valori a un byte, i cui bit vengono usati dal sistema per controllare il disegno dei bob nelle bitmap in corrispondenza della posizione del bob. Il parametro PlanePick rileva la condizione dei bitplane della bitmap destinata a ricevere l'immagine puntata dal puntatore ImageData, presente nella struttura VSprite.

Questo concetto di selezione di bitplane può essere illustrato con un semplice esempio. Si supponga di avere un'immagine di bob a due bitplane e definita da una struttura ImageData, e di voler realizzare il disegno in una bitmap a cinque bitplane. È evidente che si possono disporre i bit di bitplane per il bob in una qualunque coppia di bitplane tra i cinque che costituiscono la bitmap in questione. Ciò comporta un numero esteso di combinazioni. Per esempio, è possibile disporre l'immagine del bob nei primi due bitplane della bitmap (cioè nei bitplane 0 e 1). In tal caso, per il parametro PlanePick si userà il valore 00000011. Nel caso si voglia invece disporre l'immagine del bob negli ultimi due bitplane della bitmap (cioè nei bitplane 3 e 4), s'impiegherà per il parametro PlanePick il valore 00011000. Tale parametro controlla semplicemente quali bitplane sono destinati a ricevere i bit di dati per l'immagine del bob.

Quest'idea può essere estesa ad altre combinazioni. Per esempio, se il bob richiede tre bitplane, il parametro PlanePick può assumere valori come 00000111, 00001110 oppure 00011100, e sono possibili anche combinazioni che interessino valori non consecutivi.

Questo schema è utile perché consente d'impiegare i dati dell'immagine del bob presenti nella struttura ImageData per definire uno o più bob che abbiano tutti la stessa forma ma colori diversi nella presentazione su schermo. Una volta che i dati del bob sono stati inseriti nella bitmap, in corrispondenza della posizione stabilita, i colori assegnati ai suoi pixel risultano determinati nel solito modo. Viene cioè impiegato il valore binario ricavato dai bit appartenenti a tutti i bitplane in quella determinata posizione di pixel. È facile rendersi conto che i bit e i loro valori binari possono essere alterati in modo semplice con appropriate scelte del parametro PlanePick.

Il parametro PlaneOnOff lavora insieme con il parametro PlanePick per determinare i valori dei bit che andranno posti nei bitplane non selezionati dal parametro PlanePick. Se si ha un bob a due bitplane e una bitmap a cinque bitplane, e se si specifica il valore 00000011 per il parametro PlanePick, si otterrà l'immagine del bob nei bitplane 0 e 1 della bitmap. Se si specifica poi un valore di 00011100 per il parametro PlaneOnOff, ciascun bit appartenente ai bitplane da 2 a 4 è destinato a ricevere il valore 1 in corrispondenza di ogni pixel del bob. Se invece si precisa per il parametro PlaneOnOff un valore di 00000000, ciascun bit nelle bitmap appartenenti ai bitplane da 2 a 4 è destinato a ricevere il valore 0 in corrispondenza di ogni pixel del bob.

Il sistema, però, va oltre: provvede infatti a consultare i dati del bob per determinare la precisa posizione in cui collocare i valori specificati dal parametro PlaneOnOff. In ogni punto in cui c'è un bit impostato a 1 nella maschera d'ombra del bob, la bitmap di destinazione rileverà per quella posizione il valore presente nel parametro PlaneOnOff. Si immagini, per esempio, che la maschera d'ombra per un bob a quattro linee sia la seguente:

```
0000000000000000
0000001111000000
0000001111000000
0000000000000000
```

Se il parametro PlanePick vale 00000011, l'immagine del bob sarà disegnata nei primi due bitplane della bitmap. Si immagini ora che il parametro PlaneOnOff sia impostato a 00011100. La bitmap riceverà bit impostati a 1 nei bitplane da 2 a 4. Tuttavia saranno soltanto gli otto pixel in posizione centrale a ricevere effettivamente questi valori a 1. Tutte le altre posizioni di pixel, pure appartenenti ai bitplane da 2 a 4, manterranno il valore preesistente. Qualunque fosse la condizione di questi bit prima che venisse inserita l'immagine del bob nella bitmap, non risulteranno alterati al termine dell'operazione di scrittura.

Si noti che i parametri PlanePick e PlaneOnOff non interessano il disegno degli sprite virtuali. Infatti gli sprite virtuali non vengono disegnati nella bitmap. È il sistema che si occupa di assegnare ciascuno sprite virtuale a uno sprite hardware e genera a questo scopo una lista di istruzioni Copper. Si tratta di una distinzione da tener presente. È inoltre importante ricordare che i colori degli sprite virtuali sono sempre determinati dal parametro di puntamento SprColors della struttura VSprite, non dalla combinazione binaria dei valori di bit dei bitplane.

I parametri PlanePick e PlaneOnOff forniscono un modo molto semplice per disegnare in una bitmap un rettangolo di un solo colore. Si procede nel modo seguente:

1. Si impostano a zero, nella posizione del rettangolo, tutti i bit dei bitplane nella bitmap destinazione.

2. Si definisce una struttura VSprite per un bob fittizio. I parametri Height e Width vanno inizializzati alla misura del rettangolo desiderato. Si colloca nei parametri Depth e ImageData della struttura VSprite il valore zero. Ciò informa il sistema che non esistono dati d'immagine per il bob. Dato che il bob non possiede dati d'immagine, tutti i bit nella maschera d'ombra risulteranno pari a zero, dal punto di vista del sistema. Si definisce perciò, per questo bob, una maschera d'ombra fittizia con tutti i bit impostati a 1.

3. Il parametro PlanePick va impostato a 00000000. Ciò informa il sistema che non esistono bitplane in cui disporre i dati per l'immagine del bob.

4. Il parametro PlaneOnOff va impostato al colore scelto per il rettangolo. Per esempio, se la tavola dei colori assegna il rosso al registro di colore 5, si imposta PlaneOnOff a 00000101 (si assegnano cioè bit impostati a 1 nei bitplane 0 e 2 della bitmap).

Questo modo di procedere è utile anche in ambiente Intuition, per creare immagini di un solo colore con la struttura Image. Tuttavia, quando si usano i parametri PlanePick e PlaneOnOff in Intuition non ci si deve preoccupare della maschera d'ombra, elemento che entra in gioco soltanto come parte del sistema d'animazione. La struttura Image viene trattata insieme alla funzione DrawImage nel capitolo 6.

Animate

Sintassi di chiamata della funzione

Animate (animKey, rastPort)
A0 A1

Scopo della funzione

Questa funzione elabora le strutture AnimOb presenti nella lista degli elementi grafici per produrre l'animazione.

Argomenti della funzione

animKey	Indirizzo di un puntatore alla prima struttura AnimOb nella lista degli oggetti d'animazione.
rastPort	Indirizzo della struttura RastPort di controllo.

Discussione

La funzione Animate è l'unica della sua categoria. Elabora l'intera lista degli oggetti d'animazione per produrre l'animazione. La funzione Animate aggiorna la posizione e la velocità di ciascun oggetto d'animazione appartenente alla lista degli elementi grafici associata alla bitmap definita dalla struttura

RastPort che il task ha indicato come secondo argomento. Inoltre chiama le eventuali routine speciali d'animazione associate all'oggetto nelle strutture AnimComp e AnimOb. Si vedano anche l'introduzione al capitolo, e le spiegazioni delle funzioni AddAnimOb e InitGMasks.

La funzione Animate svolge per ciascuna struttura AnimOb le seguenti operazioni:

- aggiorna la posizione e la velocità dell'oggetto d'animazione in accordo con le indicazioni presenti nelle strutture Bob, VSprite e AnimComp.

- Chiama la routine speciale dell'oggetto d'animazione, qualora il parametro AnimORoutine della struttura AnimOb preveda un opportuno puntatore. Questa routine può anche interessare la posizione dei vari elementi dell'oggetto d'animazione.

- Per ciascun componente d'animazione, Animate passa automaticamente alla nuova sequenza di temporizzazione, se quella precedente è terminata. La temporizzazione dei componenti d'animazione viene determinata dai parametri Timer e TimeSet, della struttura AnimComp. La funzione chiama anche una speciale routine per i componenti d'animazione, se il parametro AnimCRoutine nella struttura AnimComp contiene un opportuno puntatore. Animate imposta poi le coordinate y,x relative alla posizione successiva del componente d'animazione.

Le routine AnimORoutine e AnimCRoutine vengono chiamate automaticamente dalla funzione Animate. Entrambe possono essere predisposte per intervenire sulla posizione degli elementi grafici. Ciò può produrre un ordinamento non corretto nella lista degli elementi grafici.

Quando la funzione Animate restituisce il controllo, le posizioni nella bitmap dei vari oggetti d'animazione risultano alterate. In alcuni casi, la nuova posizione non sarà ordinata in modo crescente rispetto alle coordinate y,x. Bisogna perciò, prima di chiedere al sistema di ridisegnare gli oggetti d'animazione, riordinare gli oggetti secondo le coordinate y,x crescenti; ciò significa che si dovrà chiamare ancora una volta la funzione SortGList prima di effettuare la chiamata alla funzione DrawGList.

La struttura AnimComp

Ecco la definizione della struttura AnimComp:

```
struct AnimComp {
    WORD Flags;
    WORD Timer;
    WORD TimeSet;
    struct AnimComp *NextComp;
    struct AnimComp *PrevComp;
    struct AnimComp *NextSeq;
```

```
    struct AnimComp *PrevSeq;
    WORD (*AnimCRoutine) ();
    WORD YTrans;
    WORD XTrans;
    struct AnimOb *HeadOb;
    struct Bob *AnimBob;
};
```

Ecco la descrizione dei parametri della struttura AnimComp:

- il parametro Flags contiene un insieme di flag che determinano il comportamento del componente d'animazione. Per una dettagliata descrizione delle possibili scelte si consulti il file INCLUDE graphics/gels.h.

- I parametri Timer e TimeSet fissano la durata delle attività del componente d'animazione nel sistema. Tali parametri vengono usati nell'animazione a disegni in sequenza. Il valore di TimeSet corrisponde al valore iniziale del parametro Timer. Se TimeSet è diverso da zero, Timer viene impostato a TimeSet e decrementato ogni volta che viene chiamata la funzione Animate. Quando il parametro Timer arriva a zero, il sistema d'animazione passa al successivo componente, individuato dal parametro NextSeq.

- I parametri NextComp e PrevComp puntano rispettivamente alla successiva e alla precedente struttura AnimComp del componente d'animazione dell'oggetto.

- I parametri NextSeq e PrevSeq puntano rispettivamente alla successiva e alla precedente struttura AnimComp nella sequenza di disegno.

- Il parametro AnimCRoutine punta a una speciale routine per il componente d'animazione definita dal programmatore, destinata a essere chiamata quando va in esecuzione la funzione Animate. Questo parametro va impostato a zero se non esiste nessuna routine speciale.

- I parametri YTrans e XTrans contengono i valori delle traslazioni iniziali y e x per il componente d'animazione definito dalla struttura AnimComp. Tali parametri vengono misurati sulle risoluzioni verticali e orizzontali della bitmap destinata a ricevere il bob.

- Il parametro HeadOb punta alla struttura AnimOb alla quale è associato il componente d'animazione.

- Il parametro AnimBob punta alla struttura Bob che definisce uno dei bob del componente d'animazione. Gli altri bob che costituiscono il componente d'animazione sono collegati a questo dai puntatori Before e After della struttura Bob.

ChangeSprite

Sintassi di chiamata della funzione

```
ChangeSprite (viewPort, simpleSprite, spriteImage)
              A0        A1           A2
```

Scopo della funzione

Questa funzione cambia il contenuto dei pixel di una definizione d'immagine per sprite hardware, in modo che vengano impiegati i dati conservati in una nuova struttura SpriteImage. La struttura indicata nell'argomento spriteImage deve trovarsi nella chip RAM.

Argomenti della funzione

viewPort	Indirizzo della struttura ViewPort alla quale è collegato lo sprite.
simpleSprite	Indirizzo della struttura SimpleSprite.
spriteImage	Indirizzo della struttura SpriteImage.

Discussione

Nel sistema Amiga ci sono quattro funzioni della libreria Graphics che lavorano specificamente con gli sprite hardware: ChangeSprite, FreeSprite, GetSprite e MoveSprite. Si vedano anche l'introduzione del capitolo e le spiegazioni di queste funzioni.

La funzione ChangeSprite cambia la presentazione sullo schermo di un particolare sprite hardware. Consente di cambiare colori e forma dello sprite. Impiega l'argomento viewPort per puntare a una specifica struttura ViewPort. Lo sprite hardware "appartiene" alla relativa viewport. La funzione ChangeSprite lega questo sprite hardware con una nuova serie di dati, contenuta nella struttura SpriteImage, in una definizione di viewport già esistente.

I dati contenuti nella struttura SpriteImage consistono in due word riservate e nella definizione dei pixel dello sprite hardware fornita dal programmatore. La definizione dei pixel comprende due word per ogni linea

dello sprite. La prima word contiene i bit che definiscono per ogni pixel il bit meno significativo della selezione di colore; la seconda word contiene i bit che definiscono per ogni pixel il bit più significativo della selezione di colore.

Si osservi che la scelta dei colori per gli sprite hardware è ristretta a quattro colori per ciascuno sprite. Ogni linea di dati dello sprite, infatti, viene definita soltanto con due word di dati, il che porta ad associare soltanto due bit a ogni pixel; pertanto ogni pixel può avere quattro colori (corrispondenti alle combinazioni di bit 00, 01, 10, 11). La combinazione 00 corrisponde sempre alla trasparenza, che consente di vedere lo sfondo. È tuttavia possibile cambiare le assegnazioni di colore da un quadro video all'altro e anche tra le diverse viewport di una view. Si veda anche la spiegazione della funzione MoveSprite.

Non si confondano la struttura dati SpriteImage e la struttura SimpleSprite impiegata con la funzione GetSprite. La struttura SimpleSprite contiene un puntatore ai dati per il controllo della posizione dello sprite, della sua altezza, della posizione e del numero a esso assegnato (un valore compreso tra 0 e 7). La struttura SpriteImage contiene due elementi. Il primo è costituito da due word di dati per il controllo della posizione dello sprite. I bit di queste due word vengono controllati dal sistema. Il secondo serve per definire i pixel di bitmap per lo sprite, e impiega due word per ogni linea dello sprite. Questi sono i dati che definiscono la struttura ImageData di uno sprite hardware.

La struttura SpriteImage

La struttura SpriteImage viene definita come segue:

```
struct SpriteImage {
      UWORD posctldata[2];
      UWORD sprdata[2][height];
      UWORD reserved[2];
};
```

Il significato dei parametri della struttura SpriteImage è il seguente:

- il parametro posctldata[2] contiene due word di dati per il controllo della posizione. Si tratta dei dati ai quali la struttura SimpleSprite si riferisce con il suo primo parametro. Si veda, a questo proposito, la spiegazione della funzione GetSprite.

- Il parametro sprdata[2][height] contiene i dati che definiscono l'effettiva immagine dello sprite. Questi dati consistono in due word (indicate da [2]) per ciascuna linea dell'immagine dello sprite hardware (il valore indicato dal parametro [height]). Si noti che il parametro height nella struttura SimpleSprite dello sprite hardware risulta impostato. Per esempio, se si vuole definire uno sprite hardware di una linea bisogna fornire due word di dati. Se si vuole invece definire uno sprite hardware alto come l'intero schermo video da 256 linee, bisognerà fornire 512 word di dati.

Gli ultimi due byte della struttura SpriteImage sono riservati per future espansioni e vanno inizializzati a zero.

La prima word dei dati di controllo di posizione nel parametro posctldata ha la seguente definizione di bit:

- i bit da 15 a 8 rappresentano gli otto bit meno significativi della posizione verticale di partenza dello sprite hardware.
- I bit da 7 a 0 rappresentano gli otto bit più significativi della posizione orizzontale di partenza dello sprite hardware.

La seconda word dei dati di controllo di posizione ha la seguente definizione di bit:

- i bit da 15 a 8 rappresentano gli otto bit meno significativi della posizione verticale di arresto per lo sprite hardware.
- Il bit 7 è il bit di collegamento. Esso indica al sistema di collegare questo sprite insieme all'altro della coppia per ottenere un singolo sprite che può utilizzare 15 colori al posto dei normali tre. Questo bit va impostato nello sprite avente numero dispari.
- I bit da 6 a 3 sono inutilizzati (porre a zero).
- Il bit 2 rappresenta il bit più significativo della posizione verticale di partenza dello sprite hardware.
- Il bit 1 rappresenta il bit più significativo della posizione verticale di arresto dello sprite hardware.
- Il bit 0 rappresenta il bit meno significativo della posizione orizzontale di partenza dello sprite hardware.

In generale, è il sistema che si occupa di controllare i valori di queste due word; tuttavia i loro valori sono controllati indirettamente anche dalle funzioni ChangeSprite e MoveSprite. Ciascuna linea dello sprite hardware è definita da due word. I colori per i pixel dello sprite hardware sono determinati dalla combinazione di bit di queste due word. Facciamo un esempio. Vogliamo definire uno sprite hardware a due linee. I dati dell'immagine possono essere definiti nella struttura SpriteImage come:

```
linea 1 word 1: 1111111111111111
linea 1 word 2: 1111000000001111
linea 2 word 1: 1111000000001111
linea 2 word 2: 1111111111111111
```

Questo sprite è largo 16 pixel (il che costituisce la misura massima per uno sprite hardware). Il colore per il pixel più a destra (il pixel 0) è determinato dalla

combinazione dei bit nella posizione 0 delle word 1 e 2 per la linea 1. Per questa particolare immagine, la combinazione è pari al valore binario 11, che si traduce nel numero decimale 3. A questo pixel verrà perciò assegnato il colore 3. La stessa cosa vale per i pixel da 1 a 3 e da 12 a 15 in entrambe le linee dello sprite: riceveranno tutti il colore 3.

Invece i pixel da 4 a 11 della linea 1 hanno un valore binario 01, che si traduce nel numero decimale 1. Tali pixel riceveranno il colore 1. Allo stesso modo i pixel da 4 a 11 della linea 2 hanno un valore binario 10, che si traduce nel numero decimale 2. Tali pixel riceveranno quindi il colore 2.

Le effettive assegnazioni dei registri ai colori da 1 a 3 per gli sprite hardware sono determinate dalla seguente tavola:

Numero di sprite hardware	**Registri di colore**
0 e 1	da 16 a 19
2 e 3	da 20 a 23
4 e 5	da 24 a 27
6 e 7	da 28 a 31

Questo significa che per gli sprite hardware 0 e 1, la combinazione di bit 01 risulterà assegnata al registro di colore 17, la combinazione di bit 10 sarà assegnata al registro 18, e quella con il valore binario 11 al registro 19. Lo stesso tipo di assegnazione si applica anche alle altre coppie di sprite. Il colore 0 corrisponde sempre al trasparente (nei registri 16, 20, 24 e 28): nella relativa posizione non viene disegnato nulla e il pixel lascia visibile il corrispondente pixel di playfield dello sfondo.

DoCollision

Sintassi di chiamata della funzione

DoCollision (rastPort)
A1

Scopo della funzione

Questa funzione controlla ogni elemento grafico inserito nella relativa lista per individuare eventi di collisione. Il controllo viene eseguito tanto per le collisioni fra elemento grafico e confine, quanto per quelle fra coppie di elementi grafici.

Argomenti della funzione

rastPort Indirizzo della struttura RastPort di controllo.

Discussione

DoCollision e SetCollision sono le due funzioni della libreria Graphics che riguardano direttamente le collisioni fra due elementi grafici e fra elementi grafici e confine.

Le collisioni fra elementi grafici si verificano quando le immagini di due elementi grafici (sprite virtuali oppure bob) si sovrappongono a causa dei movimenti generati dalla funzione Animate. Le collisioni fra elementi grafici e confine si verificano quando l'immagine di un elemento grafico va a sovrapporsi a uno dei confini del playfield. Si veda anche la spiegazione della funzione SetCollision.

Quando DoCollision individua una collisione, chiama l'appropriata routine di gestione.

Ci sono diverse operazioni da svolgere prima di chiamare la funzione DoCollision:

1. Si chiama la funzione SetCollision per inizializzare la tavola di gestione delle collisioni in modo che a ogni possibile collisione corrisponda un'opportuna routine di gestione.

2. Si chiama la funzione SortGList per ordinare la lista degli elementi grafici in ordine crescente di coordinate y,x.

3. Si preparano, nella struttura VSprite, i parametri di puntamento BorderLine, CollMask, HitMask e MeMask. Si veda anche la spiegazione della funzione InitGMasks. I parametri HitMask e MeMask consentono al sistema di determinare quale routine dev'essere chiamata quando viene rilevata una collisione. Per maggiori informazioni a proposito di questi due parametri, si veda la spiegazione della funzione AddVSprite.

La funzione DoCollision richiede come unico argomento l'indirizzo della struttura RastPort sulla quale deve agire. Tale struttura RastPort punta a una struttura BitMap, che a sua volta punta a una specifica bitmap. Ciascuna struttura è associata anche a una struttura GelsInfo che definisce l'ordine per la lista degli elementi grafici relativa alla bitmap. Ciascuna struttura GelsInfo possiede poi un parametro di puntamento, CollTable, che individua in memoria la tavola di collisione associata alla lista di elementi grafici.

Le collisioni che si verificano nel sistema ricadono in due categorie: collisioni fra elementi grafici, e collisioni fra elemento grafico e confine (cioè quelle di un bob o di uno sprite virtuale con un confine di bitmap).

Quando la funzione DoCollision rileva una collisione, il sistema esegue un'operazione di AND logico tra il parametro a sedici bit HitMask dell'oggetto più in alto a sinistra della coppia impegnata nella collisione, e il parametro a sedici bit MeMask dell'oggetto in basso a destra; questa operazione viene svolta bit per bit. I bit impostati a 1 come risultato dell'operazione di AND determinano quale delle sedici routine di collisione verrà chiamata. Per esempio, se la funzione Animate causa la collisione di due bob, verrà eseguito un AND tra HitMask del primo bob e MeMask del secondo bob. Se il parametro HitMask nella struttura VSprite del primo bob (che supponiamo trovarsi in alto e a sinistra rispetto al secondo bob) è 1111111111111111 e il parametro MeMask nella struttura VSprite del secondo bob è 0000000000000010, dall'operazione di AND risulterà 0000000000000010. Viene quindi chiamata la seconda routine di gestione delle collisioni.

Quando il bit 0 del risultato di AND è impostato a 1, si tratta di una collisione con un confine. Il sistema imposta il flag BORDERHIT per indicare che un elemento grafico è stato spostato al di là dei confini dell'area di disegno. La funzione DoCollision chiama poi la routine di gestione delle collisioni corrispondente al bit 0.

Se nel risultato dell'operazione AND risulta impostato uno degli altri bit (da 1 a 15), il sistema chiama la routine di gestione di collisione a esso corrispondente.

Se l'operazione AND produce l'impostazione di più di un bit, il sistema chiama la routine corrispondente al bit più a destra. Se, per esempio, il risultato è 0000000000000110, il sistema chiama comunque la seconda routine di gestione delle collisioni.

Nel caso di una collisione col bordo, i parametri passati alla routine corrispondente saranno: un puntatore alla struttura VSprite che ha colpito il bordo e una word contenente alcuni flag che indicano quale bordo è stato colpito. Essi sono definiti in graphics/collide.h.

Nel caso di una collisione tra due VSprite, invece, i parametri saranno un puntatore alla struttura VSprite più in alto e a sinistra della coppia e un puntatore alla seconda VSprite.

Inoltre, se si controllano le collisioni fra elementi grafici dopo che la funzione Animate ha spostato gli oggetti, le stesse routine di gestione delle collisioni possono intervenire sulla posizione degli elementi grafici, nel caso che siano state predisposte per questo scopo. La funzione SortGList dovrebbe perciò essere chiamata dopo ogni routine di gestione delle collisioni. Consigliamo questa sequenza di chiamate:

```
Animate (animKey, rastPort);
SortGList (rastPort);
DoCollision (rastPort);
SortGList (rastPort);
DrawGList (rastPort);
Animate (animKey, rastPort);
```

DrawGList

Sintassi di chiamata della funzione

DrawGList (rastPort, viewPort)
A1 AØ

Scopo della funzione

Questa funzione svolge una fase di disegno della lista degli elementi grafici. Per tutti gli sprite virtuali della lista, le corrispondenti definizioni vengono inserite nella lista del Copper per il quadro video successivo. Per tutti i bob della lista, le corrispondenti strutture Bob vengono usate per disegnare i bob nella bitmap.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
viewPort	Indirizzo della struttura ViewPort alla quale vanno associate le istruzioni Copper per gli sprite virtuali.

Discussione

Nella libreria Graphics ci sono tre funzioni che lavorano direttamente con la lista degli elementi grafici: DrawGList, InitGels e SortGList. Si vedano anche le spiegazioni delle funzioni InitGels e SortGList.

La funzione DrawGList prepara le istruzioni Copper e le aree RAM necessarie per presentare sullo schermo tutti gli elementi grafici definiti nella relativa lista. Si osservi che la funzione DrawGList disegna effettivamente i bob nella bitmap. Per quanto riguarda gli sprite virtuali, invece, DrawGList definisce semplicemente un insieme di istruzioni Copper; gli sprite virtuali non vengono disegnati nella bitmap. Il sistema può impiegare queste istruzioni Copper per associare gli sprite virtuali, nel momento in cui vengono visualizzati, agli sprite hardware. L'assegnazione degli sprite virtuali a quelli hardware è basata sulla posizione fissata per i primi quando è stata impostata la struttura VSprite. Queste istruzioni Copper vengono in seguito unite tramite la funzione MrgCop alle istruzioni Copper per la viewport. L'intera figura viene

poi portata sullo schermo con la funzione LoadView.

La funzione DrawGList utilizza l'indirizzo della struttura RastPort per trovare i bob nella lista degli elementi grafici. Si ricordi che ciascuna struttura Bob è collegata a una specifica struttura RastPort e perciò all'area di disegno di una determinata bitmap. Ciascuna struttura VSprite, invece, è legata a una struttura View.

La struttura View punta a un insieme di istruzioni Copper per la realizzazione dello schermo video. Si veda anche la spiegazione della funzione InitView nel capitolo 2. Attraverso una struttura ViewPort a essa collegata, la struttura View include anche un insieme di puntatori a un gruppo di istruzioni Copper per gli sprite virtuali, definito da una chiamata alla funzione DrawGList.

Per inserire nell'area video gli sprite virtuali definiti, è necessario chiamare la funzione MrgCop al fine di unire le istruzioni Copper per gli sprite virtuali alle altre istruzioni per la definizione del video. Si veda, a questo proposito, la spiegazione della funzione MrgCop nel capitolo 2. La chiamata a MrgCop va fatta seguire da una chiamata alla funzione LoadView.

FreeGBuffers

Sintassi di chiamata della funzione

FreeGBuffers (animOb, rastPort, doubleBuf)
A0 A1 D0

Scopo della funzione

Questa funzione libera tutti i buffer precedentemente assegnati dalla funzione GetGBuffers a un oggetto d'animazione.

Argomenti della funzione

animOb	Indirizzo della struttura AnimOb.
rastPort	Indirizzo della struttura RastPort di controllo.
doubleBuf	Indicatore di double-buffer; va impostato a TRUE (vero) se per l'oggetto d'animazione originario era stata impiegata la tecnica double-buffer.

Discussione

Ci sono due funzioni della libreria Graphics che si occupano direttamente della gestione della memoria per gli oggetti d'animazione: FreeGBuffers e GetGBuffers.

Per ciascuna sequenza di disegno relativa a ogni componente di AnimOb, la funzione FreeGBuffers libera la memoria per i seguenti buffer:

- il buffer SaveBuffer, impiegato per salvare la bitmap di sfondo quando viene disegnato un bob.
- I buffer BorderLine e CollMask, che contengono i dati per la rilevazione delle collisioni.
- Il buffer ImageShadow, che contiene i dati per il disegno del bob nella bitmap.

Si noti che i buffer CollMask e ImageShadow condividono in genere lo stesso spazio RAM.

Se, inoltre, la variabile doubleBuf è impostata a TRUE, il task grafico usa la tecnica double-buffer e la funzione FreeGBuffers provvederà a liberare anche la memoria assegnata ai seguenti buffer:

- il buffer DBuffer che contiene la struttura DBufPacket.
- Il buffer BufBuffer, che contiene la bitmap per il bob quando viene usata la tecnica double-buffer.

Per altre informazioni su questi buffer, si veda la spiegazione della funzione GetGBuffers.

FreeSprite

Sintassi di chiamata della funzione

FreeSprite (spritenum)
DØ

Scopo della funzione

Questa funzione libera uno sprite hardware, lasciandolo a disposizione degli altri task e per l'assegnazione a sprite virtuali. Un task se ne può impossessare chiamando la funzione GetSprite.

Argomenti della funzione

spritenum Numero intero compreso tra 0 e 7, che identifica uno degli otto sprite hardware.

Discussione

Nel sistema Amiga ci sono quattro funzioni della libreria Graphics che si occupano specificamente degli sprite hardware: ChangeSprite, FreeSprite, GetSprite e MoveSprite. Queste funzioni rendono possibile la condivisione di uno sprite hardware e gestiscono casi semplici di movimento e uso di sprite. Il sistema fornisce otto sprite hardware che possono essere impiegati per creare immagini in movimento sullo schermo video.

La funzione FreeSprite libera uno sprite hardware, precedentemente vincolato. Se si vogliono adoperare gli sprite hardware per un nuovo uso è necessario liberarli, altrimenti il sistema non ha la possibilità di assegnarli, a meno che non si effettui un nuovo boot.

Quando si usano insieme le funzioni GetSprite e FreeSprite, si ricordi che non è possibile superare il limite di otto sprite hardware. A causa di questa restrizione, l'hardware assegnato agli sprite hardware dovrebbe essere reso liberato appena possibile. Ciò è particolarmente vero se si ha a che fare soltanto con sprite di questo genere, non volendo ricorrere ai più complessi sprite virtuali.

Supponiamo di avere in un quadro video due viewport, e di voler impiegare quattro sprite hardware per la viewport superiore; vogliamo inoltre adoperare *gli stessi* quattro sprite per la viewport inferiore, ridefinendone l'aspetto e i colori. È quindi necessario liberare gli sprite hardware nel passaggio tra la prima e la seconda viewport.

Per definire la prima viewport si usano le funzioni GetSprite e ChangeSprite, ciascuna per quattro volte, per acquisire e definire i quattro sprite. Con GetSprite si definisce una struttura SimpleSprite per ciascuno degli sprite; tale struttura fornisce al sistema i dati per il controllo dello sprite, definisce la sua altezza, la posizione y,x e il numero (compreso tra 0 e 7) che lo identifica. Quando si chiama la funzione ChangeSprite si devono anche definire otto strutture SpriteImage per i colori dei pixel e la forma degli sprite.

Si devono poi liberare gli sprite con quattro chiamate alla funzione FreeSprite prima di fare la stessa cosa con la seconda viewport. Si assegnano

gli stessi sprite hardware alla seconda viewport, permettendo di ridefinire le forme, i colori e le nuove posizioni degli sprite.

In altre parole, predisporre queste strutture (in totale si tratta di otto strutture SimpleSprite e otto strutture SpriteImage) e chiamare le funzioni GetSprite, ChangeSprite e FreeSprite, significa associare ogni sprite con una particolare viewport della view considerata. Inoltre, se si impiega la funzione MoveSprite, si può cambiare la posizione delle otto immagini di sprite passando da un quadro video all'altro, producendo così un effetto d'animazione.

GetGBuffers

Sintassi di chiamata della funzione

GetGBuffers (animOb, rastPort, doubleBuf)
A0 A1 D0

Scopo della funzione

Questa funzione alloca memoria per tutti i buffer di un oggetto d'animazione. Restituisce il valore TRUE (vero) se tutte le allocazioni si sono concluse con successo, altrimenti restituisce FALSE (falso).

Argomenti della funzione

animOb	Indirizzo della struttura AnimOb.
rastPort	Indirizzo della struttura RastPort di controllo.
doubleBuf	Indicatore di double-buffer; va impostato a TRUE (vero) se si vuole impiegare la tecnica double-buffer.

Discussione

Ci sono due funzioni della libreria Graphics che si occupano direttamente della gestione della memoria per gli oggetti d'animazione: FreeGBuffers e GetGBuffers. Ciascun oggetto d'animazione può usare diversi buffer per gestire il disegno, l'impiego di double-buffer, il rilevamento delle collisioni. Tali buffer

sono indicati da vari puntatori in diverse strutture del sistema d'animazione.

La funzione GetGBuffers alloca memoria per i seguenti buffer: BorderLine e CollMask, puntati nella struttura VSprite; SaveBuffer, DBuffer e ImageShadow, puntati nella struttura Bob; BufBuffer puntato nella struttura DBufPacket. I buffer DBuffer e BufBuffer vengono impiegati soltanto se viene specificata la tecnica del double-buffer per l'oggetto d'animazione. Si noti che ciascuna struttura Bob possiede un puntatore a una diversa struttura DBufPacket.

La struttura DBufPacket

La struttura DBufPacket ha questa definizione:

```
struct DBufPacket {
    WORD BufY, BufX;
    struct VSprite *BufPath;
    WORD *BufBuffer;
};
```

BufY e BufX sono parametri di sistema e rappresentano le coordinate sullo schermo dell'ultima posizione assunta dal bob. Vengono usate per consentire la corretta ricostruzione della bitmap di playfield dello sfondo, impiegando i double-buffer assegnati al bob.

BufPath è un parametro di sistema che serve a controllare l'ordine di disegno che il sistema impiega per presentare un determinato bob nella bitmap del playfield di sfondo. La struttura RastPort contiene un puntatore alla struttura BitMap che definisce la bitmap per i playfield di sfondo. Il parametro BufPath assicura che il sistema ricostruisca le bitmap per i playfield di sfondo, seguendo la corretta sequenza. Il parametro BufPath è correlato ai parametri di sistema DrawPath e ClearPath presenti nella struttura VSprite.

Il parametro BufBuffer punta al blocco di memoria impostato dal task grafico. La funzione GetGBuffers tenta di assegnare uno spazio RAM sufficiente per entrambi i buffer di playfield adoperati con un bob double-buffer. Ciascuno di questi buffer separati dev'essere largo almeno quanto l'immagine del bob.

GetSprite

Sintassi di chiamata della funzione

```
spritenum = GetSprite (simpleSprite, spritenum)
D0                     A0            D0
```

Scopo della funzione

Questa funzione assegna uno degli otto sprite hardware a uno specifico task d'animazione. Spritenum rappresenta il numero dello sprite hardware che GetSprite tenta di assegnare. Se il valore di spritenum è -1, GetSprite assegna il primo sprite hardware disponibile. Se non vi sono sprite liberi o se lo sprite hardware richiesto è già stato assegnato, GetSprite restituisce il valore -1. Se invece lo sprite hardware è disponibile per l'assegnazione, la funzione lo contrassegna come vincolato, poi aggiorna il parametro num della struttura SimpleSprite e restituisce il numero dello sprite.

Argomenti della funzione

simpleSprite	Indirizzo della struttura SimpleSprite.
spritenum	Numero intero compreso tra 0 e 7; oppure -1 per chiedere il primo sprite disponibile.

Discussione

Nel sistema Amiga ci sono quattro funzioni della libreria Graphics che agiscono specificamente con gli sprite hardware: ChangeSprite, FreeSprite, GetSprite e MoveSprite. Si vedano anche le spiegazioni di queste funzioni e l'introduzione del capitolo.

La funzione GetSprite consente di acquisire un nuovo sprite hardware dal sistema. Si potrebbe anzi parlare di "appropriazione di sprite". Il termine "appropriazione" viene usato perché la funzione GetSprite assegna lo sprite hardware esclusivamente a un particolare sprite. Si rimane perciò vincolati a otto sprite hardware per volta. Questa restrizione, tuttavia, permette di approfittare di funzioni di gestione particolarmente semplici. Il numero di sprite virtuali è invece illimitato, ma per loro non è possibile usare le stesse semplici funzioni di gestione.

Quando si definiscono sprite hardware, la funzione GetSprite è la prima funzione da chiamare. GetSprite non fa altro che assegnare uno sprite hardware a una specifica immagine di sprite; tale sprite non risulta più disponibile fino a quando non si chiama la funzione FreeSprite per liberarlo.

Si noti che se si assegna uno sprite hardware, in realtà se ne vincolano due. Infatti, per convenzione di sistema, gli sprite hardware sono appaiati: le coppie sono formate dagli sprite 0 e 1, 2 e 3, 4 e 5, 6 e 7.

Se si assegna uno sprite hardware, vengono automaticamente assegnati anche i suoi registri di colore. I registri di colore da 16 a 19 sono assegnati agli sprite 0 e 1; quelli da 20 a 23 agli sprite 2 e 3; quelli da 24 a 27 agli sprite 4 e 5; quelli da 28 a 31 agli sprite 6 e 7.

Inoltre i registri di colore sono condivisi con le viewport di un playfield. Per esempio, se una viewport ha una profondità di cinque bitplane, i 16 registri di colore superiori, sul totale di 32, verranno usati anche dall'hardware video per le viewport di playfield. Ciò significa che 16 colori del playfield saranno identici ai colori degli sprite hardware. Naturalmente è possibile caricare nuovi valori nei registri colore mentre si passa da una viewport all'altra; questa è una delle ragioni per cui deve esistere almeno una linea di scansione video tra ogni viewport della stessa view.

Si osservi che uno degli argomenti della funzione GetSprite è un puntatore alla struttura SimpleSprite. Non si confonda la struttura SimpleSprite con la struttura SpriteImage, usata dalla funzione ChangeSprite.

La struttura SimpleSprite

Questa è la definizione della struttura SimpleSprite:

```
struct SimpleSprite {
    UWORD *posctldata;
    UWORD height;
    UWORD x,y;
    UWORD num;
};
```

La struttura SimpleSprite contiene i seguenti dati:

- un puntatore alla struttura SpriteImage associata allo sprite hardware. La struttura SpriteImage inizia con un insieme di dati per il controllo della posizione dello sprite; per questo motivo il parametro di puntamento nella struttura SimpleSprite viene chiamato posctldata. Si tratta di un insieme di valori di coordinate x,y per la posizione dello sprite hardware. In genere questi valori si possono impostare a zero. Il resto della struttura SpriteImage consiste in bit di dati per la definizione dell'immagine dello sprite hardware. Per una completa definizione della struttura SpriteImage si veda la spiegazione della funzione ChangeSprite.

- L'altezza che s'intende assegnare allo sprite hardware. È il numero delle linee di scansione in modo non interlace che verranno impiegate per lo sprite. Tale valore è limitato al numero di linee di scansione in modo non interlace che costituiscono la viewport alla quale lo sprite è assegnato dalla funzione ChangeSprite. Uno sprite può quindi raggiungere anche un'altezza di 256 linee, ma la viewport deve avere almeno la stessa altezza.

- La posizione x,y dello sprite hardware, espressa nel sistema di coordinate della viewport. La funzione MoveSprite consente di cambiare questi valori.

- Il numero assegnato allo sprite. Tale numero lega questa struttura SimpleSprite a un determinato sprite hardware e viene assegnato dalla funzione GetSprite.

InitAnimate

Sintassi di chiamata della macro

InitAnimate (animKey)
AØ

Scopo della macro

Questa macro inizializza il sistema di animazione azzerando il puntatore alla prima struttura AnimOb nella lista degli oggetti di animazione.

Argomenti della macro

animKey	Indirizzo di un puntatore alla prima struttura AnimOb in una lista concatenata.

Discussione

Nella libreria Graphics ci sono due funzioni e una macro che agiscono direttamente con gli oggetti d'animazione. Si tratta delle funzioni AddAnimOb e InitGMasks e della macro InitAnimate. Si vedano anche le spiegazioni relative alle funzioni AddAnimOb e InitGMasks.

Prima di chiamare la funzione Animate si deve inizializzare il sistema d'animazione definendo un puntatore alla prima struttura AnimOb della lista degli oggetti d'animazione. Naturalmente si deve anche inizializzare la lista degli elementi grafici con una chiamata alla funzione InitGels.

Il sistema d'animazione può essere inizializzato chiamando la macro InitAnimate, oppure usando due semplici istruzioni per specificare un puntatore al primo oggetto di animazione nella relativa lista. Ovviamente è più semplice servirsi della macro InitAnimate.

InitGels

Sintassi di chiamata della funzione

```
InitGels (headVSprite, tailVSprite, gelsInfo)
          A0           A1           A2
```

Scopo della funzione

Questa funzione inizializza una lista di elementi grafici; dev'essere chiamata prima di utilizzare qualunque elemento grafico. InitGels assegna due strutture VSprite rispettivamente alla testa e alla coda della lista degli elementi grafici. Lega poi insieme queste due strutture per formare i nodi di confine della lista.

Argomenti della funzione

headVSprite	Indirizzo della struttura VSprite fittizia che dev'essere usata come "testa" della lista degli elementi grafici.
tailVSprite	Indirizzo della struttura VSprite fittizia che dev'essere usata come "coda" della lista degli elementi grafici.
gelsInfo	Indirizzo della struttura GelsInfo da inizializzare.

Discussione

Nella libreria Graphics ci sono tre funzioni che lavorano direttamente con la lista degli elementi grafici: DrawGList, InitGels e SortGList. Si vedano anche le spiegazioni relative alle funzioni DrawGList e SortGList.

La funzione InitGels inizializza una lista di elementi grafici, ovvero una lista concatenata di tutti i bob e sprite virtuali definiti come parte di uno specifico gruppo di elementi grafici. Si possono avere simultaneamente nel sistema diverse liste di elementi grafici. Ciascuna di esse viene inizializzata con una chiamata alla funzione InitGels.

La funzione InitGels richiede un puntatore al primo e all'ultimo elemento della lista. Ciascuno di questi elementi è una struttura VSprite che può

rappresentare uno sprite virtuale o un bob. Gli argomenti headVSprite e tailVSprite dovrebbero puntare a strutture VSprite fittizie. Tali strutture non contengono informazioni utili ma sono richieste per impostare la lista degli elementi grafici. Quando poi viene eseguita la funzione InitGels, alla struttura VSprite di testa (head) viene assegnata una coppia di coordinate y,x negative, mentre alla struttura VSprite di coda (tail) viene assegnata una coppia di coordinate y,x positive. Queste coordinate devono essere scelte in modo tale che la posizione y,x di ogni elemento grafico ricada sempre tra la coppia negativa e quella positiva. Ciò assicura che queste due strutture VSprite fittizie siano sempre alle due estremità della lista degli elementi grafici.

Modifica delle definizioni per gli elementi grafici

Prima di aggiungere nuovi elementi a una lista di elementi grafici, si può cambiare l'aspetto dei bob o degli sprite virtuali. Per farlo è necessario cambiare il puntatore ImageData nella struttura VSprite. In RAM è possibile disporre diversi array di dati per ciascuna immagine. Quando si vuole poi puntare a nuovi dati di un bob o di uno sprite virtuale, ci si limita a cambiare il parametro di puntamento ImageData nella struttura VSprite. Quando si cambiano i dati dell'immagine per un bob o uno sprite virtuale, ci si ricordi di chiamare la funzione InitMasks per aggiornare le relative maschere di contorno e di collisione.

Per cambiare i colori di uno sprite virtuale, il parametro di puntamento SprColor va impostato a un nuovo valore. Si possono disporre in RAM diverse word SprColor. Quando poi si vuole puntare a un nuovo insieme di definizioni di colore per uno sprite virtuale, ci si limita a cambiare il parametro di puntamento SprColor nella struttura VSprite.

Per cambiare i colori di un bob, vanno cambiati i parametri PlanePick e PlaneOnOff presenti nella struttura VSprite associata con il bob. Si può anche cambiare il parametro Depth, se si vogliono aggiungere o sottrarre bitplane dalla definizione dell'immagine del bob. Si noti che per i bob non viene impiegato il parametro SprColor della struttura VSprite; questo parametro viene utilizzato soltanto per gli sprite virtuali.

Per cambiare la posizione di un bob o di uno sprite virtuale nell'area di disegno, si cambiano le coordinate y,x nella struttura VSprite.

Per cambiare la priorità di disegno del bob si cambia la sequenza di disegno riordinando i puntatori Before e After nella struttura Bob.

Per trasformare un bob in un pennello bisogna impostare il flag SAVEBOB nella struttura Bob. Ciò informa il sistema di non procedere alla cancellazione della vecchia immagine del bob quando questo viene spostato. Una serie di immagini bob apparirà perciò contemporaneamente sullo schermo video, simulando in questo modo l'effetto che produrrebbe un pennello in movimento. Si noti che nessuno di questi cambiamenti avrà effetto fino a quando le funzioni SortGList e DrawGList non saranno state entrambe eseguite.

La struttura GelsInfo

Ecco la definizione della struttura GelsInfo:

```
struct GelsInfo {
      BYTE sprRsrvd;
      UBYTE Flags;
      struct VSprite *gelHead, *gelTail;
      WORD *nextLine;
      WORD **lastColor;
      struct collTable *collHandler;
      SHORT leftmost, rightmost, topmost, bottommost;
      APTR firstBlissObj, lastBlissObj;
};
```

I parametri della struttura GelsInfo sono i seguenti:

- il parametro sprRsrvd è un parametro a un byte, nel quale i singoli bit determinano quali sprite hardware possono essere impiegati dal sistema degli sprite virtuali. Se, per esempio, questo parametro vale 00000000, significa che tutti gli sprite hardware sono disponibili per il sistema degli sprite virtuali. Se invece il suo valore è 10000001, significa che il primo e l'ultimo sprite hardware (cioè gli sprite numero 0 e 7) non possono essere impiegati. Tale parametro può essere impostato per impedire l'assegnazione di particolari sprite hardware.

- Il parametro Flags è un parametro a un byte i cui bit vengono impostati e impiegati dal sistema.

- Il parametro gelHead è un puntatore alla struttura VSprite da collocare in testa alla lista degli elementi grafici. Si tratta di una struttura VSprite fittizia alla quale il sistema assegna le massime coordinate negative y,x.

- Il parametro gelTail è un puntatore alla struttura VSprite da collocare in coda alla lista degli elementi grafici. Si tratta di una struttura VSprite fittizia alla quale il sistema assegna le massime coordinate positive y,x.

- Il parametro nextLine è un puntatore a un array di otto word che definisce le linee di sprite disponibili. Si tratta delle linee di schermo video tra le immagini degli sprite durante la cui visualizzazione si possono cambiare i registri di colore e gli altri registri relativi agli sprite. Questo parametro è riservato al sistema.

- Il parametro lastColor è un puntatore a un array di otto puntatori ai colori assegnati per ultimi agli sprite virtuali. Questo parametro è riservato al sistema.

- Il parametro collHandler è un puntatore all'array della tavola di collisione. Tale array contiene una lista dei puntatori alle routine di gestione delle collisioni da impiegare quando quando il sistema rileva una collisione fra due elementi grafici o fra un elemento grafico e un confine. Gli elementi della tavola di collisione sono definiti dalla funzione SetCollision.

- I parametri leftmost e rightmost contengono le coordinate x, mentre i parametri topmost e bottommost contengono le coordinate y del rettangolo di delimitazione usato per limitare il disegno di un bob in una determinata bitmap. Quando il bob supera questi confini, le routine di disegno "tagliano" la parte eccedente durante l'esecuzione del disegno. Questi parametri non vengono impiegati per limitare il disegno degli sprite virtuali. Si noti che è possibile impiegare questi quattro parametri per rilevare collisioni fra elemento grafico e confine.

- I parametri firstBlissObj e lastBlissObj sono riservati al sistema.

InitGMasks

Sintassi di chiamata della funzione

InitGMasks (animOb)
AØ

Scopo della funzione

Questa funzione imposta tutte le maschere di una struttura AnimOb. Chiama la funzione InitMasks per ogni sequenza di disegno di ogni struttura AnimComp.

Argomenti della funzione

animOb	Indirizzo della struttura AnimOb.

Discussione

Nella libreria Graphics ci sono due funzioni e una macro che agiscono direttamente con gli oggetti di animazione. Esse sono rispettivamente: AddAnimOb, InitGMasks e InitAnimate.

La funzione InitGMasks fornisce un modo rapido per impostare le maschere di contorno e di collisione di tutti i componenti di un oggetto d'animazione. Ogni volta che si cambiano i dati dell'immagine di un bob o di uno sprite virtuale che servono per definire un determinato componente di animazione, si dovrebbe chiamare la funzione InitMasks. Ma se si cambiano diverse definizioni di elementi grafici in un oggetto d'animazione potrebbe risultare più efficiente la chiamata alla funzione InitGMasks per aggiornare automaticamente le maschere di contorno e di collisione per ciascun componente d'animazione dell'oggetto.

La funzione InitGMasks chiama la funzione InitMasks per ogni singolo componente di un determinato oggetto d'animazione. Ogni maschera degli sprite virtuali o dei bob che costituiscono il componente d'animazione viene impostata. Si veda anche la funzione InitMasks.

InitMasks

Sintassi di chiamata della funzione

InitMasks (vSprite)
AØ

Scopo della funzione

Questa funzione imposta le maschere di contorno e di collisione per uno sprite virtuale. Rileva i dati dell'immagine conservati nella struttura ImageData ed elabora tali maschere per lo sprite virtuale.

Argomenti della funzione

vSprite	Indirizzo della struttura VSprite.

Discussione

Nella libreria Graphics ci sono tre funzioni che agiscono direttamente con gli sprite virtuali: AddVSprite, InitMasks e RemVSprite. Si vedano anche le spiegazioni relative alle funzioni AddVSprite e RemVSprite.

La funzione InitMasks imposta le maschere di contorno e di collisione associate alla struttura VSprite. I parametri di puntamento BorderLine e CollMasks sono entrambi contenuti nella struttura VSprite.

La maschera di collisione di un elemento grafico è in genere identica alla sua maschera d'ombra. Per ciascun pixel dell'immagine di bitmap di un elemento grafico (cioè i dati di ImageData), i bit del primo bitplane subiscono un'operazione di OR logico con i corrispondenti bit di tutti gli altri bitplane. Il risultato viene inserito nell'area dati della maschera di collisione. Perciò le misure di altezza e larghezza della bitmap della maschera di collisione coincidono con le misure dei bitplane dell'elemento grafico.

Se, per esempio, un elemento grafico ha due bitplane e ogni bitplane ha le dimensioni di 16 per 32 bit, anche la maschera di collisione sarà di 16 per 32 bit. Ogni bit della maschera di collisione risulta impostato se anche uno solo dei bitplane che costituiscono l'immagine dell'elemento grafico ha un bit impostato nella stessa posizione. Se nessuno dei bitplane ha un bit impostato in quella posizione, il bit corrispondente nella maschera di collisione assume il valore 0.

La maschera di collisione viene impiegata per controllare il disegno di un elemento grafico e per definire dove ci sono bit impostati nella bitmap dell'elemento grafico, al fine di rilevare eventuali collisioni. Ciascun bit impostato nella maschera di collisione indica che nella corrispondente posizione c'è una parte dell'immagine dell'elemento grafico. Quando tale immagine si sovrappone al pixel appartenente a un altro oggetto presente sullo schermo, significa che si sta verificando una collisione.

Per rilevare le collisioni viene impiegata anche la maschera di contorno. La maschera di contorno viene generata tramite un'operazione di OR logico tra tutte le word che definiscono l'immagine dell'elemento grafico. Se per esempio l'immagine di un elemento grafico è definita dalle seguenti sei word:

```
0011000011000000
0001100110000000
0000111100000000
0001100110000000
0011000011000000
0000000000000000
```

la maschera di contorno risulterà essere:

```
0011111111000000
```

Si produce, in pratica, un bit a 1 nelle posizioni in cui almeno una delle word che definiscono l'immagine dell'elemento grafico ha un bit a 1. Questo

modo di procedere equivale più o meno a prendere l'intera immagine e condensarla in una sola linea.

La maschera di contorno ha sempre un bit a 1 sui bordi dell'immagine dell'elemento grafico. Nell'esempio, i bordi dell'immagine si trovano alle posizioni 6 e 13 (il bit 0 è quello più a destra). Il sistema impiega la maschera di contorno per determinare rapidamente quando il bob o lo sprite virtuale sta toccando il confine destro o quello sinistro dell'area di disegno. Questo sistema integra la rilevazione della collisione con l'impiego della maschera. La misura dell'area dati da assegnare alla maschera di contorno dev'essere estesa almeno quanto la larghezza dell'immagine dell'elemento grafico (espressa in word). Se, per esempio, occorrono tre word per registrare ciascuna linea dell'immagine dati del bob, la RAM assegnata alla maschera di contorno deve contenere almeno tre word.

Si noti che sia gli sprite virtuali sia i bob partecipano alla rilevazione delle collisioni software. Perciò la funzione InitMasks dovrebbe essere chiamata per ciascun elemento grafico su cui si vuole effettuare il controllo di collisione. In alternativa, è possibile chiamare la funzione InitGMasks per impostare, con una sola chiamata di funzione, la maschera di ciascun elemento grafico di un oggetto d'animazione.

MoveSprite

Sintassi di chiamata della funzione

```
MoveSprite (viewPort, simpleSprite,    x,    y)
             A0        A1              D0    D1
```

Scopo della funzione

Questa funzione sposta l'immagine di uno sprite hardware verso un nuovo punto di una viewport. Le coordinate x,y vengono misurate nel sistema di coordinate della viewport.

Argomenti della funzione

viewPort	Indirizzo della struttura ViewPort.
simpleSprite	Indirizzo della struttura SimpleSprite.

x	Nuova coordinata orizzontale dello sprite, relativa al margine sinistro della bitmap di viewport.
y	Nuova coordinata verticale dello sprite, relativa al margine superiore della bitmap di viewport.

Discussione

Ci sono quattro funzioni della libreria Graphics che agiscono specificamente con gli sprite hardware del sistema Amiga: ChangeSprite, FreeSprite, GetSprite e MoveSprite.

La funzione MoveSprite consente di spostare uno sprite hardware all'interno di una viewport. Tale movimento può produrre i desiderati effetti d'animazione. La funzione MoveSprite, come ChangeSprite, è legata a una determinata viewport.

Le funzioni GetSprite e FreeSprite, invece, non riguardano una specifica viewport, ma rendono disponibili per il sistema gli sprite hardware, che poi verranno assegnati alle viewport tramite la funzione ChangeSprite.

Si può impiegare la funzione MoveSprite per definire una serie di quadri video, al fine di ottenere effetti d'animazione. A questo scopo va specificata per lo sprite una posizione x,y spostata di pochi pixel da un quadro all'altro.

Quando la funzione MoveSprite restituisce il controllo, lo sprite si trova in una nuova posizione x,y della bitmap di viewport. Si ricordi che le coordinate specificate per la chiamata alla funzione MoveSprite sono espresse nel sistema della viewport. L'origine di questo sistema di coordinate è il punto 0,0 della bitmap di viewport, cioè l'angolo superiore sinistro della viewport, così come appare sullo schermo video.

Gli argomenti x,y sono le nuove coordinate pixel verso le quali lo sprite va mosso. La risoluzione dello spostamento è legata alla bassa risoluzione, in modo video senza interlace. Se la viewport fa parte di uno schermo in alta risoluzione e/o in interlace, il sistema può spostare lo sprite soltanto con incrementi di due pixel da alta risoluzione o due pixel da interlace a ogni chiamata di MoveSprite.

OFF_SPRITE

Sintassi di chiamata della macro

OFF_SPRITE

Scopo della macro

Questa macro azzera il bit di abilitazione del DMA degli sprite (direct memory access, accesso diretto alla memoria). Qualsiasi sprite successivamente caricato con la funzione LoadView non verrà presentato sullo schermo fino alla chiamata della macro ON_SPRITE.

Argomenti della macro

Questa macro non ha argomenti.

Discussione

Ci sono due macro che riguardano il controllo DMA degli sprite: ON_SPRITE e OFF_SPRITE, che agiscono sul bit SPREN nel registro DMACON.

Le macro ON_SPRITE e OFF_SPRITE sono molto simili alle macro ON_DISPLAY e OFF_DISPLAY, trattate nel capitolo 2. Rimandiamo alle spiegazioni a esse dedicate.

ON_SPRITE

Sintassi di chiamata della macro

ON_SPRITE

Scopo della macro

Questa macro imposta il bit di abilitazione del DMA (direct memory access, accesso diretto alla memoria). Qualsiasi sprite (hardware o virtuale) successivamente caricato con la funzione LoadView verrà presentato sullo schermo.

Argomenti della macro

Questa macro non ha argomenti

Discussione

Ci sono due macro che riguardano il controllo DMA degli sprite: ON_SPRITE e OFF_SPRITE. ON_SPRITE imposta il bit numero 5 (chiamato bit SPREN) nel registro di controllo DMA (cioè il registro denominato DMACON). Dopo che una nuova view è stata caricata con la funzione LoadView, si può impiegare la macro ON_SPRITE per consentire al sistema DMA di visualizzare gli sprite.

ON_SPRITE è l'impostazione di default del sistema. Ciò significa che, a meno che si non provveda a disattivare il DMA di sprite utilizzando la macro OFF_SPRITE, qualsiasi sprite definito viene subito visualizzato sullo schermo.

RemBob

Sintassi di chiamata della macro

RemBob (bob)
AØ

Scopo della macro

Questa macro rimuove una struttura Bob da una lista di elementi grafici e la cancella dalla definizione della bitmap.

Argomenti della macro

bob	Indirizzo della struttura Bob.

Discussione

Ci sono due funzioni e una macro della libreria Graphics che agiscono direttamente con i bob (Blitter objects, oggetti del Blitter): le funzioni AddBob e RemIBob, e la macro RemBob. Sia la funzione RemIBob che la macro RemBob servono per rimuovere un bob dalla lista degli elementi grafici. A differenza della macro RemBob, la funzione RemIBob cancella immediatamente il bob dalla bitmap. Si vedano anche le le spiegazioni relative alle funzioni AddBob e RemIBob.

La macro RemBob svolge due operazioni:

1. Informa il sistema di cancellare il bob dall'area di disegno. Se sono impostati i flag appropriati, le informazioni di sfondo relative all'area occupata dal bob verranno prelevate dai relativi buffer e ricostruite.

2. Rimuove il bob dalla lista degli elementi grafici.

Contrariamente alla funzione RemIBob, il bob non viene immediatamente rimosso. Il sistema lo cancella nel corso della successiva esecuzione della funzione DrawGList. Se è in uso la tecnica double-buffer, la versione su schermo del bob viene rimossa nel corso della prima esecuzione della funzione DrawGList, mentre la versione fuori schermo viene eliminata nel corso della seconda esecuzione della funzione DrawGList.

In ogni caso, il sistema slega la struttura Bob dalla lista degli elementi grafici. Per disegnare un bob al di sopra di quello rimosso è necessaria un'ulteriore chiamata alla funzione DrawGList.

RemIBob

Sintassi di chiamata della funzione

```
RemIBob (bob, rastPort, viewPort)
         A0   A1       A2
```

Scopo della funzione

Questa funzione rimuove una struttura Bob da una lista di elementi grafici e cancella immediatamente l'immagine corrispondente dalla bitmap.

Argomenti della funzione

bob	Indirizzo della struttura Bob.
rastPort	Indirizzo della struttura RastPort di controllo.
viewPort	Indirizzo della struttura ViewPort

Discussione

Ci sono due funzioni e una macro della libreria Graphics che agiscono direttamente con i bob: le funzioni AddBob e RemIBob, e la macro RemBob. Si vedano anche le spiegazioni relative alla funzione AddBob e alla macro RemBob.

La funzione RemIBob svolge due operazioni:

1. Informa il sistema di cancellare il bob dall'area di disegno. Se sono impostati i flag appropriati, le informazioni di sfondo relative all'area occupata dal bob vengono ricostruite.

2. Rimuove immediatamente il bob dalla lista degli elementi grafici, slegando la struttura Bob dalla lista. Per disegnare un bob al di sopra di quello appena rimosso è necessaria un'ulteriore chiamata alla funzione DrawGList.

RemVSprite

Sintassi di chiamata della funzione

RemVSprite (vSprite)
A0

Scopo della funzione

Questa funzione libera e rimuove una struttura VSprite dalla lista in uso degli elementi grafici.

Argomenti della funzione

vSprite Indirizzo della struttura VSprite.

Discussione

Nella libreria Graphics ci sono tre funzioni che agiscono direttamente con gli sprite virtuali: AddVSprite, InitMasks e RemVSprite. Si vedano anche le spiegazioni relative alle funzioni AddVSprite e InitMasks.

L'azione della funzione RemVSprite è molto semplice. Si limita a rimuovere uno sprite virtuale dalla lista degli elementi grafici. Si noti che è possibile disporre di diverse liste di elementi grafici contemporaneamente attive, ognuna delle quali è stata inizializzata con una chiamata alla funzione InitGels. Ogni lista di elementi grafici inizia e finisce con una struttura VSprite fittizia. Tali strutture VSprite di testa e di coda sono state specificate nella chiamata alla funzione InitGels.

Ogni sprite virtuale aggiunto con una chiamata alla funzione AddVSprite colloca un nuovo sprite virtuale in una lista di elementi grafici. Lo stesso sprite virtuale può essere collocato in diverse liste, che vanno inizializzate una per una con nuove chiamate alla funzione InitGels prima di procedere con la chiamata alla funzione AddVSprite. Ciascuno di tali sprite virtuali risulta associato a una struttura RastPort, a una particolare bitmap e a una particolare struttura GelsInfo.

La chiamata alla funzione RemVSprite rimuove la struttura VSprite dalla stessa lista di elementi grafici in cui era stata originariamente posta dalla chiamata alla funzione AddVSprite.

SetCollision

Sintassi di chiamata della funzione

```
SetCollision (numroutine, routine, gelsInfo)
             D0           A0       A1
```

Scopo della funzione

Questa funzione imposta un puntatore a una routine di gestione delle collisioni prevista dal programmatore. L'indirizzo della routine di gestione

viene memorizzato nel vettore numero numroutine (che può variare da 0 a 15) della tavola delle routine di gestione delle collisioni.

Argomenti della funzione

numroutine	Numero della routine di gestione delle collisioni (può variare da 0 a 15).
routine	Indirizzo della routine di gestione delle collisioni.
gelsInfo	Indirizzo della struttura GelsInfo.

Discussione

Ci sono due funzioni della libreria Graphics che riguardano direttamente le collisioni fra coppie elementi grafici e fra elemento grafico e confine: DoCollision e SetCollision. La funzione DoCollision controlla il verificarsi di collisioni su ogni elemento della lista degli elementi grafici. Se ne trova, chiama automaticamente l'appropriata routine di gestione.

La funzione SetCollision associa a un bit di collisione una routine di gestione delle collisioni. Ciò consente di collocare in RAM, in diverse posizioni, varie routine di questo genere. Si definisce poi una tavola di collisione in cui vengono conservati i puntatori alle diverse routine predisposte. La funzione SetCollision non fa altro che definire i valori della tavola di vettori che puntano alle routine di gestione delle collisioni.

La funzione SetCollision agisce con la struttura CollTable. Tale struttura consiste in un puntatore a una tavola di puntatori a routine di gestione delle collisioni (denominata collPtrs). Ciascuno di questi 16 puntatori individua una specifica routine di gestione delle collisioni nel sistema.

Quando la funzione DoCollision rileva una collisione tra due elementi grafici, informa il sistema di chiamare una o più delle previste routine di gestione delle collisioni. I parametri MeMask e HitMask assegnati a ciascun elemento grafico determinano qual è la routine da chiamare. Si veda anche l'introduzione a questo capitolo.

Quando la struttura GelsInfo viene inizializzata per la prima volta, il sistema imposta a zero tutti i puntatori alle routine di gestione delle collisioni presenti nella struttura CollTable. Per adoperare queste routine si devono perciò impostare le voci della tavola di collisione che corrispondono ai bit impostati per HitMask e MeMask.

SortGList

Sintassi di chiamata della funzione

SortGList (rastPort)
A1

Scopo della funzione

Questa funzione ordina la lista degli elementi grafici in accordo con la progressione delle coordinate y,x degli elementi che la costituiscono. Gli elementi grafici vengono poi disposti in una nuova lista in ordine crescente di coordinate y,x. La lista dev'essere ordinata tramite SortGList prima di effettuare chiamate alle funzioni DrawGList e DoCollision.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo che contiene il puntatore alla struttura GelsInfo.

Discussione

Nella libreria Graphics ci sono tre funzioni che agiscono direttamente con la lista degli elementi grafici: DrawGList, InitGels e SortGList. Si vedano anche le spiegazioni relative alle funzioni DrawGList e InitGels.

Sia gli sprite virtuali che i bob vengono inseriti in una determinata lista di elementi grafici definita con la funzione InitGels. Ogni lista di questo genere può vedere alterato il proprio ordine per diverse ragioni.

La prima è dovuta al fatto che la posizione di uno sprite virtuale o di un bob può mutare per effetto di una chiamata a una funzione d'animazione; per questo è necessario riordinare tutti gli elementi grafici nel corretto ordine di disegno. Secondo, una chiamata a una delle due routine AnimORoutine, AnimCRoutine può spostare sprite virtuali o bob mettendoli in condizioni di non corretto ordine y,x. Terzo, una chiamata a una delle 16 routine di gestione delle collisioni può spostare sprite virtuali o bob. Il verificarsi di queste due ultime categorie di eventi dipende da come sono state progettate le routine di gestione delle collisioni e quelle speciali.

Si ricordi che vi possono essere diverse liste attive di elementi grafici. Se

ci si trova in tale condizione, tutte devono essere disposte nel corretto ordine y,x. Per farlo basta cambiare l'argomento puntatore alla struttura RastPort nella chiamata alla funzione SortGList, e chiamare questa funzione per ciascuna delle strutture RastPort che utilizzano una diversa struttura GelsInfo.

La funzione SortGList dovrebbe essere chiamata dopo la conclusione delle operazioni da parte della funzione Animate. Quest'ultima, infatti, è in grado di cambiare la posizione sullo schermo degli oggetti d'animazione. Inoltre, le routine AnimORoutine e AnimCRoutine, chiamate automaticamente dalla funzione Animate, possono cambiare le coordinate y,x degli elementi grafici nella lista corrispondente.

La funzione SortGList dovrebbe essere chiamata anche dopo una chiamata a DoCollision, la quale attraverso le routine di gestione delle collisioni (qualora siano previste) può cambiare la posizione sullo schermo degli elementi grafici.

Le funzioni di gestione del testo della libreria Graphics

Introduzione

Questo capitolo definisce e illustra le funzioni di gestione dei testi della libreria Graphics e le funzioni della libreria DiskFont, che rendono possibile gestire testi e fonti-carattere nel sistema Amiga. Si suddividono in tre categorie:

- le funzioni di gestione del testo costituite da ClearEOL, ClearScreen, Text e TextLength; servono per inserire testi nelle bitmap.
- Le funzioni di gestione delle fonti-carattere residenti in RAM: AddFont, AskFont, AskSoftStyle, CloseFont, OpenFont, RemFont, SetFont e SetSoftStyle. Servono per aggiungere, gestire e rimuovere dal sistema le fonti-carattere.
- Le funzioni di gestione delle fonti-carattere residenti su disco: AvailFonts e OpenDiskFont. Queste ultime funzioni appartengono alla libreria DiskFont e sono state incluse in questo capitolo a causa della loro relazione con le funzioni di gestione delle fonti-carattere residenti in RAM.

Le liste delle fonti-carattere

Le funzioni che riguardano le fonti-carattere operano servendosi di due liste. Una è conservata in RAM e l'altra su disco. La prima lista contiene l'elenco delle fonti residenti in RAM ed è chiamata lista delle fonti di sistema. Ogni chiamata alla funzione AddFont aggiunge a questa lista una nuova fonte. Al contrario, ogni chiamata alla funzione RemFont ne rimuove una.

La lista su disco riguarda tutte le fonti presenti nel disco e conservate in file separati. Tale lista viene automaticamente aggiornata dalle strutture che definiscono e aggiungono nuove fonti al disco considerato. Lo scopo di questa lista è di legare insieme tutti i file di fonti del disco. L'insieme di tutte le fonti presenti in un disco è composto in genere da una o più sotto-categorie di famiglie. Impiegando le strutture DiskFontHeader, FontContents e FontContentsHeader, la lista delle fonti su disco lega insieme tutti i file appartenenti alla stessa famiglia. Quando poi viene caricata una fonte-carattere appartenente a una famiglia, vengono caricate anche tutte le altre della stessa famiglia. Ciò riduce il numero degli accessi al disco che il sistema deve effettuare.

Una famiglia consiste molto spesso in un insieme di fonti che hanno lo stesso nome ma corpi differenti. Per esempio, le fonti di sistema Topaz-60 e Topaz-80 appartengono alla stessa famiglia; hanno lo stesso nome ma differiscono nella dimensione (cioè nel corpo dei caratteri). Le fonti Sapphire sono un altro esempio di famiglia. Naturalmente, corpo-carattere a parte,

possono anche essere altre le ragioni che inducono a definire una famiglia di fonti.

Il sistema è in grado di separare gli elementi di una famiglia dall'insieme complessivo delle fonti sul disco seguendo i nomi corrispondenti. Il sistema operativo impiega le strutture FontContentsHeader e FontContents per rintracciare una determinata fonte, seguendo il percorso che inizia dalla directory font del disco considerato, attraverso il disco e da lì a una famiglia di fonti e infine alla specifica fonte. Le strutture FontContentsHeader e FontContents sono contenute in quello che è chiamato il "file per la definizione delle fonti", che possiede sempre l'estensione .font. Questo file contiene la descrizione di una famiglia di fonti; non contiene una fonte effettiva. Le strutture FontContentsHeader e FontContents vengono discusse insieme alla funzione OpenDiskFont.

Oltre al file per la descrizione delle fonti, ogni fonte appartenente a una famiglia possiede un unico file per la sua definizione. Si tratta del file che contiene la struttura DiskFontHeader e la definizione effettiva della fonte-carattere. La struttura DiskFontHeader contiene una sotto-struttura TextFont e altre informazioni necessarie per definire la fonte-carattere. Si tratta del file che contiene le definizioni di bitmap per ciascun carattere della fonte. La struttura DiskFontHeader impiega una sotto-struttura Node per legare tutte le fonti in una famiglia. La struttura DiskFontHeader è trattata anche nel corso della spiegazione della funzione OpenDiskFont.

La definizione e l'impiego delle fonti-carattere

Per definire una determinata fonte è necessario costruire una struttura TextFont che contenga tutti i parametri necessari per indicare le sue caratteristiche e fornire i puntatori ai dati di ciascun carattere di cui la fonte è composta. È però necessario allocare prima la memoria per la struttura TextFont e l'area dati alla quale essa punta. Si possono impiegare le funzioni di allocazione di memoria della libreria Exec per effettuare le assegnazioni destinate alla struttura TextFont e ai blocchi di dati associati. Tali aree di memoria devono trovarsi nella memoria pubblica, cioè del tipo MEMF_PUBLIC. Queste funzioni restituiscono puntatori utilizzabili per specificare la posizione in memoria della struttura TextFont e dei blocchi di dati a essa associati. Tali blocchi di dati devono contenere le informazioni di bitmap relative a ciascun carattere della fonte; si tenga presente che nella versione 1.3 del sistema ogni carattere è definito da un solo bitplane.

Una volta che la struttura TextFont e i blocchi di dati a essa associati sono predisposti in RAM, la funzione AddFont può usare il puntatore alla struttura TextFont per aggiungere la fonte-carattere alla lista di sistema. Quando AddFont restituisce il controllo, ogni task può accedere alla fonte chiamando la funzione OpenFont.

Ogni task che voglia fare uso di una specifica fonte, deve eseguire una chiamata alla funzione OpenFont. È necessario specificare la struttura TextAttr per la chiamata a OpenFont, usando opportune istruzioni di assegnazione dei

parametri di struttura in modo da inizializzare ognuno dei suoi quattro parametri. In alternativa, si può usare la funzione AskFont per creare una struttura TextAttr basata sulle impostazioni della fonte in uso in una determinata struttura RastPort.

Se tuttavia la struttura TextFont era stata creata e salvata su disco in precedenza, allora si tratta di una fonte residente su disco. In tal caso le chiamate alle funzioni AddFont e OpenFont sono rimpiazzate dalle chiamate alle funzioni AvailFonts e OpenDiskFont. La funzione AvailFonts esamina tutte le fonti sul disco e crea in memoria un insieme di strutture per contarle e caratterizzarle. Costruisce per ciascuna fonte presente sul disco una struttura AvailFonts, contenente a sua volta una struttura TextAttr che ne segnala gli attributi. Quando viene chiamata la funzione OpenDiskFont per aprire una fonte con un determinato insieme di attributi, viene consultato l'insieme delle strutture AvailFonts per controllare se esiste sul disco una fonte con quegli attributi. In caso affermativo OpenDiskFont carica il file della fonte così individuata.

Una volta che una fonte è stata caricata per un task particolare, è possibile impiegare le funzioni AskSoftStyle e SetSoftStyle per creare versioni di caratteri sottolineati, in neretto o in corsivo (oppure combinazioni tra tali versioni). Il disegno dei caratteri viene effettuato nella bitmap, eventualmente in momenti diversi della sequenza di presentazione. Se tuttavia la fonte ha quelle caratteristiche nella sua definizione originale, non è possibile impiegare AskSoftStyle e SetSoftStyle per toglierle. La stessa fonte o una sua versione modificata può essere impiegata in qualsiasi task che abbia chiamato la funzione OpenFont. Quando un task ha acquisito l'accesso a una fonte può chiamare le funzioni di testo per inserire caratteri nella bitmap in cui opera.

Ciascun task è in grado di chiamare la funzione OpenFont (o OpenDiskFont) per qualsiasi fonte sia stata inserita nel sistema con le funzioni AddFont o AvailFonts. Ciò significa che possono esserci diverse fonti contemporaneamente aperte in qualunque task. Un task può cambiare la fonte in uso nelle sue operazioni di gestione del testo impiegando la funzione SetFont.

Inoltre, ciascun task può verificare le caratteristiche di stile di qualsiasi fonte-carattere abbia aperto, usando la funzione AskSoftStyle; una volta individuate le caratteristiche dello stile, può alterarle chiamando la funzione SetSoftStyle. In seguito il task, quando ha terminato l'accesso alla fonte-carattere considerata, può sempre tornare alle precedenti caratteristiche di stile ripristinando i valori originari.

Quando un task non deve più impiegare una certa fonte-carattere, può chiamare CloseFont per liberare la memoria assegnata in RAM alla struttura TextFont e alle strutture per le fonti-carattere a essa associate. Ciascun task, prima di essere eliminato dal sistema, dovrebbe eseguire una chiamata alla funzione CloseFont per tutte le fonti che ha aperto, in modo che le risorse di memoria a esse riservate tornino disponibili per gli altri task del sistema.

Le strutture correlate al testo

Le due più importanti strutture del sistema delle fonti-carattere sono la struttura TextFont e la struttura TextAttr. La prima è la struttura fondamentale, che contiene tutte le informazioni necessarie per definire una fonte-carattere. In particolare, la struttura TextFont contiene le caratteristiche di misura e i puntatori a un insieme di dati a blocchi di bit che definiscono tutti i caratteri presenti nella fonte.

I parametri nella struttura TextFont sono discussi in modo approfondito insieme alla funzione AddFont. Si noti che il primo elemento della struttura TextFont è una sotto-struttura Node, che consente alle varie strutture TextFont di essere legate insieme nella lista di sistema delle fonti.

La struttura TextAttr contiene soltanto quattro parametri: un puntatore al nome della fonte, la sua altezza in pixel, una definizione da 1 byte dei bit di stile e un parametro di flag da 1 byte. I parametri nella struttura TextAttr sono trattati in dettaglio insieme alla funzione AskFont.

Le strutture TextFont e TextAttr sono riportate nei file INCLUDE text.h (per il linguaggio C) e text.i (per il linguaggio Assembly).

Oltre a queste due strutture, le funzioni di libreria per le fonti riguardano cinque strutture specificamente riferite alle fonti su disco. Si tratta delle strutture AvailFontsHeader, AvailFonts, FontContentsHeader, FontContents e DiskFontHeader.

Le strutture AvailFontsHeader e AvailFonts definiscono tutte le fonti presenti nel sistema, tanto quelle in RAM quanto quelle su disco. La struttura AvailFontsHeader contiene un solo elemento che rappresenta il numero complessivo delle fonti su disco e in RAM. Questa struttura viene creata in RAM al momento della chiamata alla funzione AvailFonts; viene quindi descritta insieme a tale funzione.

La struttura AvailFonts, anch'essa creata in RAM al momento della chiamata alla funzione AvailFonts, è costituita da due elementi. Il primo indica la provenienza della fonte (dal disco o dalla memoria). Il secondo è una sotto-struttura TextAttr relativa alla fonte considerata. Per ciascuna fonte esiste una struttura AvailFonts. Si veda a questo proposito la spiegazione relativa alla funzione AvailFonts.

Le strutture FontContentsHeader e FontContents sono contenute nei file per la descrizione delle fonti presenti sul disco. Tali strutture forniscono il meccanismo per contare i file su disco e per indirizzare il sistema operativo verso tali file. La struttura FontContentsHeader contiene due elementi. Il primo è la definizione del tipo di file, che identifica il file come header di fonte-carattere. Il secondo elemento è il numero di fonti-carattere presenti nella famiglia. La struttura FontContentsHeader viene descritta insieme alla funzione OpenDiskFont.

La struttura FontContents è molto simile alla struttura TextAttr. Infatti gli ultimi tre parametri sono gli stessi di quelli visti per TextAttr. Il primo parametro definisce il percorso di directory che il sistema deve seguire per raggiungere la fonte descritta nella struttura. La struttura FontContents viene trattata insieme alla funzione OpenDiskFont.

La struttura DiskFontHeader si trova nei file su disco che contengono effettivamente una definizione di file. Per ciascuna struttura FontContents esiste una corrispondente definizione di fonte-carattere. La struttura DiskFontHeader contiene una sotto-struttura Node per legare questo file di definizione di fonte in una famiglia, e una sotto-struttura TextFont per definire le caratteristiche della specifica fonte-carattere. La struttura DiskFontHeader viene trattata insieme alla funzione OpenDiskFont.

AddFont

Sintassi di chiamata della funzione

AddFont (textFont)
A1

Scopo della funzione

Questa funzione inserisce una fonte-carattere nella lista di sistema delle fonti. Quando AddFont restituisce il controllo, la fonte aggiunta risulta a disposizione di ogni task. La fonte rimane nella lista di sistema fino a quando non viene rimossa con la funzione RemFont.

Argomenti della funzione

textFont Indirizzo della struttura TextFont.

Discussione

La funzione AddFont appartiene alla categoria delle funzioni per la gestione delle fonti residenti in RAM. In questo gruppo le funzioni AddFont, CloseFont, OpenFont, RemFont e SetFont hanno tutte a che fare con la struttura TextFont.

La struttura TextFont

Questa è la definizione della struttura TextFont:

```
struct TextFont {
    struct Message tf_Message;
    UWORD tf_YSize;
    UBYTE tf_Style;
    UBYTE tf_Flags;
    UWORD tf_XSize;
    UWORD tf_Baseline;
    UWORD tf_BoldSmear;
    UWORD tf_Accessors;
    UBYTE tf_LoChar;
    UBYTE tf_HiChar;
    APTR tf_CharData;
    UWORD tf_Modulo;
    APTR tf_CharLoc;
    APTR tf_CharSpace;
    APTR tf_CharKern;
};
```

I parametri della struttura TextFont sono i seguenti:

- tf_Message. Si tratta di una sotto-struttura Message. È il primo parametro della struttura TextFont che consente a diversi task di accedere alla stessa fonte-carattere. Tale struttura Message viene impiegata come messaggio di risposta da un task quando una determinata struttura TextFont e la fonte-carattere a essa associata sono state rimosse dal task considerato. All'interno della struttura Message la sotto-struttura Node consente a un insieme di strutture TextFont di essere legate insieme nella lista di sistema delle fonti. Ogni volta che si aggiunge una fonte (usando AddFont) o si rimuove una fonte (usando RemFont), il sistema aggiorna la sua lista. È importante ricordare che il nome della fonte è quello contenuto nella sotto-struttura Node.

- tf_YSize. Questo parametro contiene l'altezza in pixel della fonte, misurata in base alla risoluzione della bitmap. Per esempio, se questo parametro vale 8, ciascun carattere è destinato a occupare otto linee di risoluzione verticale sulla bitmap destinazione. Questa dimensione viene chiamata "corpo" della fonte.

- tf_Style. I bit di questo parametro da 1 byte specificano lo stile intrinseco della fonte. Al momento, questi bit identificano cinque possibili stili: tondo, sottolineato, neretto, corsivo, esteso. Si chiamano bit di stile intrinseco perché contengono la definizione di base dello stile della fonte; questo valore non viene mai cambiato nella struttura

TextFont. I corrispondenti bit di stile del parametro AlgoStyle della struttura RastPort possono essere alterati attraverso la funzione SetSoftStyle soltanto nei casi in cui il loro valore intrinseco nel parametro tf_Style della struttura TextFont sia uguale a zero. Si vedano a questo proposito le spiegazioni delle funzioni AskSoftStyle e SetSoftStyle.

- tf_Flags. Questo parametro mantiene le impostazioni delle preferenze. Ciascuna di queste impostazioni è indicata attraverso un valore specifico del parametro Flag. Alcuni esempi d'impostazioni di preferenza sono ROMFONT, REVPATH e PROPORTIONAL. L'impostazione di preferenza ROMFONT (valore = 0) non dovrebbe essere usato a meno che non si provveda a creare una nuova ROM per proprio conto. L'impostazione di preferenza REVPATH (valore = 2) viene impiegata quando una fonte dev'essere presentata da destra verso sinistra, come nel caso dei caratteri ebraici. L'impostazione di preferenza PROPORTIONAL (valore = 32) viene usata quando i caratteri non hanno sempre la stessa larghezza XSize. Nei file INCLUDE sono descritte altre impostazioni di preferenza.

- tf_XSize. Questo parametro contiene la larghezza in pixel del carattere, misurata sulla risoluzione orizzontale della bitmap destinazione.

- tf_Baseline. Questo parametro contiene il numero di linee, dalla sommità alla linea di base (intesa come la linea più bassa dei caratteri senza discendenti), del più alto carattere della fonte. La posizione della linea di base è la stessa per tutti i caratteri della fonte. Quando un carattere viene disegnato in una bitmap, il valore del parametro cp_y nella struttura RastPort di controllo coincide con la posizione della linea di base.

- tf_BoldSmear. Questo è un parametro da due byte usato per produrre caratteri in neretto. Una sbavatura (smear) è un insieme di bit aggiunti al carattere per conferirgli l'aspetto caratteristico del neretto.

- tf_Accessors. Questo parametro contiene il numero dei task che hanno accesso alla fonte; varia con lo svolgersi di operazioni di apertura e chiusura sulla fonte in questione. Si tratta del secondo parametro che rende possibile l'accesso contemporaneo di più task alla fonte-carattere. Ogni volta che un nuovo task chiama OpenFont o OpenDiskFont, questo parametro viene incrementato. Una chiamata a CloseFont decrementa tf_Accessors che tuttavia non scende mai sotto il valore zero. Questo parametro dovrebbe essere impostato a zero prima di aggiungere una nuova fonte nel sistema ma in seguito è gestito dal sistema.

- tf_LoChar. Questo parametro contiene il valore numerico (compreso tra 1 e 255) del primo carattere descritto da questa struttura TextFont. Ogni fonte non può avere più di 255 caratteri. Se sono necessari più caratteri

bisogna usare due o più fonti, con le relative strutture TextFont. Si noti che non è obbligatorio usare il valore 1 per il primo carattere. Oltre ai numeri relativi ai caratteri effettivamente presenti nella definizione, si deve assegnare un numero a un carattere fittizio aggiuntivo. Tale carattere fittizio aggiuntivo viene usato per tutti i riferimenti del programma a un carattere il cui valore numerico non sia stato assegnato. Se per esempio si crea una fonte che possiede i caratteri A, B e C e il programma richiama un carattere diverso, nella bitmap viene disegnato il carattere fittizio.

- tf_HiChar. Questo parametro rappresenta il valore numerico (compreso tra 1 e 255) dell'ultimo carattere descritto dalla struttura TextFont. Si veda, a questo proposito, la spiegazione relativa al parametro tf_LoChar.

I prossimi cinque parametri descrivono la fonte-carattere: essi forniscono la definizione di come sono organizzati i dati per i caratteri all'interno dell'array. Si noti che per le fonti proporzionali dev'esserci un insieme di descrittori di fonte (charData, modulo, charSpace e charKern) per ciascun carattere che compone l'insieme.

- tf_CharData. Si tratta di un puntatore a un'area di memoria in cui si trovano i bit di definizione. Ciascun carattere della fonte ha una propria area rettangolare di memoria. In ciascun rettangolo di memoria vi sono bit a 0 e a 1 che rappresentano la definizione in pixel del carattere. Tale area di memoria contiene una rappresentazione della fonte organizzata in raggruppamenti di bit.

- tf_Modulo. Questo parametro contiene il numero dei byte per linea di carattere della fonte. I dati di bit per ciascuna fonte sono organizzati con i bit della linea più alta del primo carattere adiacenti ai bit della linea più alta del secondo carattere e così via. Questo parametro informa il sistema della posizione in cui si trova l'informazione di bit per la successiva linea del carattere. Se per esempio l'insieme dei caratteri definito nel raggruppamento di bit ha bisogno di 20 word per mantenere la linea più alta di tutti i caratteri che compongono l'insieme, tf_Modulo vale 40 byte. Il sistema deve aggiungere 40 al puntatore della matrice carattere per procedere da una linea del carattere alla successiva.

- tf_CharLoc. Si tratta di un puntatore alla posizione di memoria in cui si trova un array di valori accoppiati per ciascun carattere della fonte. I dati per la fonte sono organizzati in insiemi di coppie di word; la prima word è l'offset di bit nell'array a raggruppamento di bit, relativo al carattere, la seconda word è la larghezza in bit del carattere. Il sistema impiega questo array per determinare la larghezza in pixel di ciascun carattere della fonte e per localizzare il punto iniziale di una determinata definizione in bit di un carattere.

- tf_CharSpace. È un puntatore a un array di word che definiscono la spaziatura proporzionale della fonte considerata. Ciascuna word dell'array contiene la larghezza del rettangolo in cui è contenuto il carattere corrispondente. Per esempio, una I potrebbe essere definita con una larghezza di tre soli bit, ma il suo rettangolo potrebbe arrivare anche a una larghezza di sette bit, lasciando due bit di spazio su ogni lato del carattere. Se questo puntatore vale zero, il sistema adopera per ciascun carattere la larghezza specificata nel parametro XSize.

- tf_CharKern. È un puntatore a un array di word usato per definire i dati riguardanti il punto d'origine di ogni carattere. I dati di origine informano il sistema sulla posizione in cui i bit iniziano effettivamente nel rettangolo in cui il carattere è compreso. Se per esempio la larghezza da fianco a fianco del rettangolo è di 10 pixel e un carattere inizia all'altezza del secondo pixel, la word di origine per la lettera considerata assumerà il valore 2. Se questo puntatore è nullo, significa che non esistono dati di origine.

AskFont

Sintassi di chiamata della funzione

AskFont (rastPort, textAttr)
A1 AØ

Scopo della funzione

Questa funzione riempie la struttura TextAttr con i parametri di attributo per il testo presenti nella struttura RastPort relativa alla fonte in uso. Alcuni di questi parametri vengono ottenuti indirettamente dalla struttura TextFont alla quale punta la struttura RastPort.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
textAttr	Indirizzo della struttura TextAttr.

Discussione

La funzione AskFont è una delle funzioni di gestione delle fonti-carattere residenti in RAM che fanno parte della libreria Graphics. Rileva gli attributi della fonte assegnata a una data struttura RastPort in un determinato task e impiega queste informazioni per riempire la struttura TextAttr.

Ciascun task può usare diverse fonti. Inoltre ciascun task può disegnare testi in diverse bitmap. L'inserimento di tali testi in ciascuna bitmap viene controllato attraverso la struttura RastPort associata alla bitmap. Si ricordi che ogni struttura RastPort contiene un puntatore a una struttura BitMap. Inoltre, ciascuna struttura RastPort contiene sette parametri riguardanti il controllo del disegno dei testi nella bitmap associata alla struttura RastPort. Ecco i parametri di disegno del testo presenti nella struttura RastPort:

- Font. Questo parametro punta alla struttura TextFont associata alla struttura RastPort. Fornisce a qualsiasi funzione che punti alla struttura RastPort l'accesso ai parametri della struttura TextFont. In particolare permette alle funzioni AskSoftStyle e SetSoftStyle di accedere ai bit di stile intrinseci contenuti nella struttura TextFont (cioè il parametro tf_Style). Si noti che la funzione SetFont può cambiare il valore del parametro Font della struttura RastPort, rendendo così possibile impiegare diverse fonti per il disegno dei testi nella stessa bitmap.

- AlgoStyle. Si tratta di un parametro da 1 byte, impiegato in modo simile al parametro tf_Style nella struttura TextFont. È un byte singolo i cui bit individuali contengono la definizione dei bit di stile generati in modo algoritmico per la particolare fonte-carattere. Tale parametro viene cambiato dalle chiamate alla funzione SetSoftStyle. Il sistema lo imposta a 0. Per la definizione dei bit di stile si veda la spiegazione della struttura TextFont, trattata insieme alla funzione AddFont.

- TxFlags. È un parametro da 1 byte, i cui bit definiscono le preferenze per la fonte-carattere in uso. È paragonabile al parametro tf_Flags della struttura TextFont. A proposito dei bit di flag si veda la descrizione della struttura TextFont, trattata con la funzione AddFont.

- TxHeight. Questo è un parametro da una word che definisce l'altezza, espressa in numero di pixel, della fonte impiegata per disegnare testi nella bitmap associata alla struttura RastPort. Questo parametro è correlato al parametro tf_YSize della struttura TextFont, e in pratica indica il corpo della fonte.

- TxWidth. È un parametro da una word che definisce la larghezza, espressa in pixel, della fonte in uso nella bitmap associata alla struttura RastPort. Questo parametro è correlato al parametro tf_XSize della struttura TextFont.

- TxBaseline. È un parametro da una word, che definisce la posizione della linea di base della fonte impiegata per disegnare testi nella bitmap associata alla struttura RastPort. Questo parametro è correlato al parametro tf_Baseline della struttura TextFont.

- TxSpacing. È un parametro da una word che definisce la spaziatura dei caratteri, espressa in pixel, della fonte in uso nella bitmap associata alla struttura RastPort. Il parametro considerato è correlato al parametro tf_CharSpace della struttura TextFont.

La struttura TextAttr

La struttura TextAttr viene utilizzata dalle funzioni AskFont, OpenFont e OpenDiskFont. Ecco la definizione della struttura TextAttr:

```
struct TextAttr {
    STRPTR ta_Name;
    UWORD ta_YSize;
    UBYTE ta_Style;
    UBYTE ta_Flags;
};
```

Ciascuno dei parametri nella struttura TextAttr può essere assegnato con una semplice istruzione di assegnazione di parametro, oppure si può ottenere dai parametri di attributo della fonte (AlgoStyle, TxFlags...) contenuti nella struttura RastPort. La struttura TextAttr serve a due scopi. Per prima cosa fornisce una definizione a quattro parametri di una fonte-carattere che un task *richiede* in uso. Questo è il suo significato quando viene impiegata nelle chiamate alle funzioni OpenFont o OpenDiskFont. In secondo luogo fornisce una definizione a quattro parametri per le fonti che si trovano effettivamente nel sistema. Questo è il significato che assume nelle strutture AvailFonts realizzate attraverso la funzione AvailFonts. In ogni situazione il sistema è in grado di confrontare ciò che si richiede con ciò che è disponibile, per procedere con azioni appropriate. Ecco i parametri della struttura TextAttr:

- ta_Name. È un puntatore a una stringa di testo a terminazione nulla che rappresenta il nome della fonte-carattere. Se per esempio si vogliono usare le fonti di default del sistema, per questo parametro possono essere usate Topaz-60 o Topaz-80. D'altra parte, se si sono predisposte fonti personalizzate o se si desidera usare qualche altra fonte-carattere, sarà la funzione AskFont a fornire l'appropriato nome della fonte. Il parametro del nome qui specificato verrà paragonato con il nome della fonte nella sotto-struttura Node della struttura TextFont.

- ta_Size. Questo parametro indica l'altezza in pixel della fonte. La fonte specificata nel parametro del nome dovrebbe avere questa altezza, detta anche corpo.

- ta_Style. Questo parametro contiene i bit di stile desiderati oppure quelli intrinseci della fonte, a seconda del contesto in cui la struttura TextAttr viene usata.

- ta_Flags. Questo parametro contiene le impostazioni di preferenza desiderate o effettive, ancora una volta in relazione al contesto in cui la struttura TextAttr viene usata.

AskSoftStyle

Sintassi di chiamata della funzione

```
enable = AskSoftStyle (rastPort)
DØ                    A1
```

Scopo della funzione

Questa funzione restituisce una versione modificata del parametro AlgoStyle presente nella struttura RastPort. Il parametro AlgoStyle contiene i bit di stile che vengono generati in modo algoritmico dall'attività di diversi task che hanno, fino al momento considerato, avuto accesso e manipolato i bit di stile di una fonte. Il valore che viene restituito può essere impiegato come maschera per ulteriori cambiamenti di stile, effettuati usando la funzione SetSoftStyle. In altre parole il valore restituito diventa una maschera di abilitazione quando viene adoperata come argomento della funzione SetSoftStyle.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo contenente il parametro AlgoStyle che rappresenta i bit di stile generati algoritmicamente per la fonte-carattere.

Discussione

La funzione AskSoftStyle è una funzione di gestione delle fonti-carattere residenti in RAM che fa parte della libreria Graphics.

Ciascuna fonte di sistema possiede un insieme di bit di stile. I bit di stile originariamente assegnati a una fonte-carattere sono contenuti nel parametro da 1 byte tf_Style nella struttura TextFont. Si tratta dei cosiddetti bit intrinseci di stile. I bit presenti in questo parametro definiscono i vari stili nel modo seguente:

Stile	**Byte di stile**
Tondo	00000000 (valore = 0)
Sottolineato	00000001 (valore = 1)
Neretto	00000010 (valore = 2)
Corsivo	00000100 (valore = 4)
Esteso	00001000 (valore = 8)

Dato che ciascun bit di questo parametro indica uno stile specifico, è possibile combinare due o più stili della stessa fonte-carattere. Il valore 3, per esempio, indica una fonte in neretto sottolineato. Come si può osservare, i bit 5, 6, 7 e 8 sono liberi per stili addizionali, al momento non ancora definiti.

Ciascuna fonte-carattere del sistema ha diversi bit intrinseci di stile ed eventualmente un insieme di bit di stile generati in modo algoritmico. I bit intrinseci di stile sono i bit impostati nella struttura TextFont, al momento dell'inizializzazione. Si noti che una fonte può avere tutti i bit di stile impostati a zero (anzi, è la norma).

I bit di stile generati in modo algoritmico sono definiti come bit che hanno subito alterazioni da parte di un task. Un determinato task può, per esempio, ridefinire i bit di stile della fonte originaria impostando a 1 il bit meno significativo (il bit 0 viene impostato a 1). Qui i bit di stile generati in modo algoritmico stanno a rappresentare una fonte sottolineata. Ciò potrebbe avvenire cambiando il parametro AlgoStyle nella struttura RastPort per *richiedere* una fonte sottolineata. Può essere definita una struttura TextAttr corrispondente a questa situazione. Una chiamata a OpenFont verificherà attraverso la struttura AvailFonts se la citata fonte è già sottolineata. Se la fonte è stata definita come normale, il sistema provvede a modificarla in sottolineata per le successive operazioni di disegno.

Quando i bit di stile sono stati modificati da un task di programma, vengono chiamati algoritmici. In genere i bit di stile vengono alterati dalla funzione SetSoftStyle all'interno dell'algoritmo di un programma. La funzione SetSoftStyle prende il suo nome dall'azione dell'algoritmo software sull'impostazione del parametro di stile.

Siccome i bit di stile possono cambiare sotto il controllo del programma, essi hanno un comportamento dinamico all'interno del task; i bit intrinseci di stile nella struttura TextFont, restituiti dalle funzioni OpenFont o OpenDiskFont, non rappresentano i soli stili possibili nel disegno dei testi in una bitmap.

Per questo motivo, i bit di stile sono contenuti in tre diversi parametri di tre diverse strutture: il parametro AlgoStyle della struttura RastPort, il parametro ta_Style della struttura TextAttr e il parametro tf_Style della struttura TextFont.

La definizione iniziale dei bit di stile è contenuta nella struttura TextFont.

La versione dei bit di stile della struttura TextFont costituisce la versione intrinseca. Nessuno stile che sia stato definito come intrinseco (i cui bit siano cioè stati impostati) può essere cambiato; gli stili caratterizzati in questo modo verranno sempre impiegati seguendo tale definizione di carattere per la fonte. Gli stili che non sono stati definiti come intrinseci (cioè quelli i cui bit sono impostati a zero) possono invece essere generati in modo algoritmico impiegando la funzione SetSoftStyle. Ciò significa che è possibile aggiungere caratteristiche come il nero, il sottolineato o il corsivo a una fonte normale, ma che non si possono sottrarre tali caratteristiche da una fonte che è stata prevista per includerle. Si dovrebbe tuttavia sapere che il sistema, in generale, non svolge un buon lavoro nell'aggiungere il nero o il corsivo alle fonti-carattere nelle quali non sono previsti tali effetti. I caratteri risultanti a volte risultano difficili da leggere o semplicemente brutti. Si noti inoltre che il sistema non può creare una fonte estesa se non esiste uno stile intrinseco. Per restare sul sicuro, tuttavia, è meglio usare il sistema soltanto per creare il sottolineato da una fonte non sottolineata. Tutti gli altri stili dovrebbero essere creati come stili intrinseci nella definizione della fonte.

La definizione dei bit di stile generati algoritmicamente impiegata per il disegno dei testi è contenuta nel parametro AlgoStyle della struttura RastPort. Quei bit spiegano come il testo va effettivamente disegnato in un determinato momento e in una particolare bitmap. I bit di stile generati algoritmicamente, e che si trovano nella struttura RastPort, possono essere, in seguito, cambiati ancora una volta attraverso la funzione SetSoftStyle.

Una terza definizione dei bit di stile viene utilizzata per cercare tra le strutture AvailFonts una fonte che risponda a determinate caratteristiche. Tale versione dei bit di stile è contenuta nel parametro ta_Style della struttura TextAttr. Ogni volta che si vuole aprire una nuova fonte per usarla secondo la definizione iniziale presente nella relativa struttura RastPort, è necessario definire e impostare una struttura TextAttr. Ciò si ottiene definendo direttamente i parametri nella struttura TextAttr, oppure, in alternativa, chiamando la funzione AskFont. Se si adopera quest'ultimo sistema, la struttura TextAttr viene definita tramite gli appropriati parametri della struttura RastPort (AlgoStyle, TxFlags...). Si chiamano poi le funzioni OpenFont oppure OpenDiskFont per cercare, nella lista delle fonti disponibili, una che abbia le caratteristiche definite nel parametro ta_Style della struttura TextAttr.

La funzione AskSoftStyle riporta semplicemente la versione modificata del valore dei bit di stile generati algoritmicamente, contenuta in una particolare struttura RastPort. AskSoftStyle copia il parametro AlgoStyle dalla struttura RastPort e lo modifica prima di restituirlo.

Supponiamo, per esempio, che si voglia utilizzare la fonte in uso (cioè la fonte puntata nella struttura TextFont) per disegnare in una bitmap caratteri in corsivo. I bit di stile intrinseci nella struttura TextFont indicano che la fonte era originariamente definita come carattere in neretto, esteso, sottolineato. Il parametro di stile nella struttura TextFont è quindi 00001011. Dato che si vuole alterare la fonte per includervi il corsivo, si cambia il parametro AlgoStyle della struttura RastPort in 00000100. In queste condizioni AskSoftStyle restituirà il valore 00000100. Il valore restituito dalla funzione AskSoftStyle dà via libera al sistema per modificare la fonte-carattere e generare caratteri in corsivo. Questo

valore può essere usato in seguito nella funzione SetSoftStyle per alterare ulteriormente l'insieme dei bit generati in modo algoritmico. Per un approfondimento sulla manipolazione dei bit di stile si veda la spiegazione della funzione SetSoftStyle.

AvailFonts

Sintassi di chiamata della funzione

```
error = AvailFonts (buffer, number_bytes, types)
D0                  A0      D0             D1
```

Scopo della funzione

Questa funzione crea una struttura AvailFontsHeader che contiene il numero delle fonti in memoria e su disco. Crea inoltre per ciascuna di esse una struttura AvailFonts (che contiene una struttura TextAttr).

Argomenti della funzione

buffer — Indirizzo in memoria del buffer in cui inserire i dati per definire la struttura AvailFontsHeader e una serie di strutture AvailFonts.

number_bytes — Numero di byte del buffer.

types — Valori che indicano se si desidera effettuare la ricerca delle fonti in memoria (AFF_MEMORY) o su disco (AFF_DISK).

Discussione

Ci sono due funzioni che agiscono direttamente con le fonti-carattere su disco: AvailFonts e OpenDiskFont. La funzione AvailFonts consente di costruire un array di tutte le fonti su disco e in memoria per mezzo di una struttura AvailFontsHeader e di creare inoltre in memoria una serie di strutture AvailFonts. La funzione OpenDiskFont carica una fonte-carattere dal disco e la

mette a disposizione del task. Queste due funzioni sono contenute nella libreria DiskFont, ma le includiamo in questo capitolo perché sono direttamente correlate alle funzioni della libreria Graphics che operano con le fonti residenti in RAM.

La funzione AvailFonts riempie il buffer indicato con informazioni destinate a definire una struttura AvailFontsHeader e una serie di strutture AvailFonts, che contengono informazioni relative a tutte le fonti disponibili in memoria e su disco. Una volta che le citate strutture sono state appropriatamente definite, le fonti su disco devono essere caricate chiamando la funzione OpenDiskFont. Le fonti che già risiedono in memoria possono essere aperte con una chiamata alla funzione OpenFont.

Uno dei risultati della chiamata alla funzione AvailFonts è la definizione di una struttura TextAttr per ogni fonte. Per la definizione della struttura TextAttr si veda la spiegazione della funzione AskFont.

La funzione AvailFonts restituisce il controllo quando le citate strutture sono state immagazzinate in un buffer di memoria. Si noti che esisteranno voci doppie per le fonti presenti contemporaneamente in memoria e su disco; esse differiranno soltanto per il tipo (AFF_MEMORY oppure AFF_DISK). L'argomento types va impostato ad AFF_MEMORY per cercare le fonti in memoria, e ad AFF_DISK per cercarle sul disco. I due tipi AFF_MEMORY e AFF_DISK possono essere specificati anche contemporaneamente, se si vogliono aggiungere al sistema tutte le fonti su disco e in memoria. L'esistenza di una struttura AvailFonts nel buffer indica soltanto che esiste la relativa definizione di fonte. La funzione AvailFonts però non controlla la validità della definizione. Perciò una chiamata alla funzione OpenDiskFont può anche non avere successo, se la struttura TextFont che definisce la fonte in questione è costruita in modo sbagliato.

Se la variabile error restituita dalla funzione AvailFonts è diversa da zero, il suo valore indica il numero dei byte aggiuntivi richiesti dal buffer per mantenere le informazioni generate dalla funzione AvailFonts. In tal caso significa che le informazioni sulle fonti non sono state tutte incluse nel buffer perché non era abbastanza capiente. Quindi, se si ottiene dalla funzione un valore diverso da zero si deve accrescere come indicato la misura del buffer e chiamare nuovamente la funzione AvailFonts.

La funzione AvailFonts localizza le fonti all'interno del cammino di ricerca impostato dall'AmigaDOS. Si può impartire un comando ASSIGN dell'AmigaDOS per attribuire al dispositivo logico FONTS: un percorso di ricerca alternativo per individuare le fonti su disco. Per default, il sistema assegna al dispositivo logico FONTS: il percorso di ricerca sys:fonts.

La struttura AvailFontsHeader

La struttura AvailFontsHeader è molto semplice:

```
struct {
    UWORD afh_NumEntries;
};
```

Il parametro afh_NumEntries contiene il numero di strutture AvailFonts presenti nel buffer RAM. Le fonti doppie in RAM e su disco vengono contate una sola volta.

La struttura AvailFonts

Anche la struttura AvailFonts è molto semplice:

```
struct AvailFonts {
    UWORD af_Type;
    struct TextAttr af_Attr;
};
```

Il parametro af_Type contiene il tipo di fonte; può essere tanto una fonte già in RAM (AFF_MEMORY), quanto una fonte ancora su disco (AFF_DISK).

Il parametro af_Attr rappresenta una sotto-struttura TextAttr che descrive gli attributi della fonte. La struttura TextAttr contiene il nome della fonte, la sua altezza, lo stile e i parametri di flag. La struttura TextAttr viene descritta insieme alla funzione AskFont.

ClearEOL

Sintassi di chiamata della funzione

```
ClearEOL (rastPort)
         A1
```

Scopo della funzione

Questa funzione cancella un'area rettangolare (definita dall'altezza della fonte di testo in uso) dalla posizione "pixel corrente" verso il confine destro della bitmap. La posizione pixel corrente all'interno della bitmap viene definita dal parametro cp_x della struttura RastPort.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.

Discussione

Ci sono quattro funzioni della libreria Graphics che riguardano la posizione del testo in una bitmap: ClearEOL, ClearScreen, Text e TextLength.

La funzione ClearEOL cancella un'area rettangolare iniziando dalla posizione pixel corrente e procedendo verso il fianco destro della bitmap. L'altezza dell'area rettangolare è pari all'altezza della fonte-carattere in uso, indicata dal parametro TxHeight della struttura RastPort di controllo. La posizione verticale dell'area rettangolare è fissata dalla coordinata verticale della penna (il parametro cp_y), contenuta sempre nella struttura RastPort e corrisponde alla linea di base dei caratteri come indicato in TxBaseline. Ciò garantisce che qualsiasi output di testo nell'area rettangolare cancellata non uscirà dai suoi confini verticali. L'operazione di cancellazione consiste nell'impostare il colore dell'area in questione al colore 0 (cioè il colore definito dal registro colore 0) oppure, qualora la variabile DrawMode fosse JAM2, al colore di BgPen.

ClearScreen

Sintassi di chiamata della funzione

ClearScreen (rastPort)
A1

Scopo della funzione

Questa funzione cancella un'area di bitmap per inserire caratteri di testo. I caratteri possono essere inseriti iniziando dalla posizione pixel corrente fino al confine della bitmap.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.

Discussione

La funzione ClearScreen fa parte delle funzioni di gestione del testo della libreria Graphics. ClearScreen cancella un'area rettangolare della bitmap, partendo dalla posizione della penna e procedendo verso destra, utilizzando la funzione ClearEOL. Poi azzera tutta la parte di bitmap che si trova al di sotto dell'area rettangolare cancellata. La posizione della penna nella bitmap viene definita dai parametri cp_x e cp_y della struttura RastPort. La cancellazione consiste nell'impostare il colore dell'area da azzerare al valore 0 (cioè il colore definito dal registro colore 0) oppure, se la variabile DrawMode è impostata a JAM2, al colore di BgPen.

CloseFont

Sintassi di chiamata della funzione

CloseFont (textFont)
A1

Scopo della funzione

Questa funzione chiude una fonte, aperta con le funzioni OpenFont oppure OpenDiskFont. In tal modo le fonti che non sono più necessarie cessano di occupare memoria. Dopo la chiusura, il task non può più accedere a quella particolare fonte-carattere a meno che non chiami di nuovo OpenFont oppure OpenDiskFont.

Argomenti della funzione

textFont	Indirizzo della struttura TextFont.

Discussione

La funzione CloseFont, della libreria Graphics, fa parte delle funzioni per la gestione delle fonti-carattere residenti in RAM. In ogni momento, il sistema può avere parecchi task concorrenti. Ciascuno di essi può avere aperto diverse

fonti, le quali, insieme a tutte le altre informazioni richieste da un task, possono imporre al sistema problemi di memoria. Per limitare questo problema, ogni task dovrebbe chiudere appena possibile le fonti impiegando la funzione CloseFont.

Si noti che non esiste una funzione specifica per chiudere le fonti residenti sul disco. Questo dipende dal fatto che quando una fonte residente sul disco viene aperta, diventa residente in RAM. Entrambi i tipi di fonti vengono perciò chiusi attraverso la funzione CloseFont.

Si noti inoltre che la fonte rimane nella lista di sistema quando CloseFont restituisce il controllo. Se si desidera rimuoverla completamente dalla memoria, è necessario chiamare la funzione RemFont.

OpenDiskFont

Sintassi di chiamata della funzione

```
textFont = OpenDiskFont (textAttr)
DØ                      AØ
```

Scopo della funzione

Questa funzione cerca nel disco la fonte-carattere indicata nella struttura TextAttr. Carica poi in memoria la struttura TextFont e i dati della fonte a essa associati, restituendo l'indirizzo della struttura TextFont. Tale indirizzo può essere impiegato in chiamate successive alle funzioni SetFont e CloseFont. La funzione OpenDiskFont restituisce il valore zero se su disco non trova la fonte richiesta.

Argomenti della funzione

textAttr	Indirizzo della struttura TextAttr.

Discussione

Ci sono due funzioni che agiscono direttamente con le fonti-carattere su disco: AvailFonts e OpenDiskFont. Queste due funzioni sono contenute nella libreria DiskFont, ma le includiamo in questo capitolo perché sono direttamen-

te correlate alle funzioni della libreria Graphics che operano con le fonti residenti in RAM.

Il procedimento di accesso a una fonte viene chiamato "apertura della fonte"; la funzione OpenDiskFont apre le fonti-carattere residenti su disco e le carica in memoria. È importante che ogni chiamata alla funzione OpenDiskFont sia seguita prima o poi da una chiamata alla funzione CloseFont. Se la fonte è già in memoria, OpenDiskFont accede alla memoria senza ricaricare la copia che si trova sul disco. Se la fonte non viene rintracciata, la funzione OpenDiskFont restituisce il valore zero.

Si noti che la fonte caricata da OpenDiskFont è determinata unicamente dal contenuto della struttura TextAttr indicata nella chiamata alla funzione. La struttura TextAttr è costituita da quattro parametri: il nome della fonte, la sua altezza, lo stile e i flag. A proposito della struttura TextAttr si vedano anche l'introduzione al capitolo e la spiegazione della funzione AskFont.

La funzione cerca nel disco una struttura TextFont che abbia le caratteristiche specificate nella struttura TextAttr indicata. OpenDiskFont carica quindi in memoria tale struttura TextFont e gli array per la definizione dei caratteri a essa associati.

Se la chiamata alla funzione si conclude con successo, viene restituito un puntatore alla struttura TextFont. Tale puntatore può poi essere impiegato dal task che ha emesso la chiamata per indicare la fonte-carattere alle funzioni SetFont, CloseFont e RemFont.

Se un task vuole impiegare una certa fonte-carattere, deve effettuare una chiamata alla funzione OpenDiskFont oppure OpenFont. Ogni fonte può essere assegnata simultaneamente a diversi task. Ognuno, quando diventa attivo, può accedere alla fonte e disegnare caratteri di testo nelle bitmap su cui sta lavorando.

Ogni file di fonte-carattere su disco è realizzato come modulo caricabile ed eseguibile. Per questa ragione l'AmigaDOS può allocare memoria e caricare una fonte proprio come se si trattasse di un programma. L'AmigaDOS può anche liberare la memoria occupata dalla fonte quando questa viene rimossa dal sistema. Ciò significa che non ci si deve preoccupare di allocare e liberare memoria per le fonti su disco.

Per accedere a una fonte-carattere su disco si seguono i seguenti passi:

1. Si apre la libreria DiskFont con una chiamata alla funzione OpenLibrary, come viene spiegato nell'introduzione del libro.

2. Se necessario, si esegue una chiamata alla funzione AvailFonts per sapere quali sono le fonti presenti su disco e in RAM. La funzione AvailFonts crea in RAM le strutture AvailFontsHeader e AvailFonts. Per ogni fonte in RAM e su disco esiste una struttura AvailFonts (che contiene una struttura TextAttr). Si veda a questo proposito la spiegazione della funzione AvailFonts.

3. Si apre la fonte su disco chiamando la funzione OpenDiskFont e indicando come argomento una specifica struttura TextAttr. È possibile definire la struttura TextAttr con un insieme di istruzioni o impiegando

la funzione AskFont. La struttura TextAttr deve contenere le indicazioni per rintracciare una determinata fonte all'interno di una famiglia (naturalmente ci possono essere anche famiglie composte da una sola fonte). La funzione AskFont tenta di trovare l'appropriata struttura TextAttr in una delle strutture AvailFonts create con la chiamata alla funzione AvailFonts.

Ogni fonte del sistema appartiene a una famiglia. Per esempio Topaz-60 e Topaz-80 appartengono alla famiglia di default del sistema, Topaz. Un altro esempio è la famiglia delle fonti Sapphire; ogni fonte di questa famiglia ha un'altezza diversa. Le famiglie consentono a fonti correlate di essere raggruppate sul disco.

Una directory di fonti contiene in genere due nomi per ciascuna famiglia. Una tipica coppia di voci per una famiglia di fonti è:

```
sapphire.font
sapphire(dir)
```

Il file sapphire.font contiene una descrizione del contenuto della particolare famiglia di fonti all'interno di una struttura FontContentsHeader, e un insieme di strutture FontContents. La struttura FontContentsHeader indica quante strutture FontContents sono necessarie per definire tutte le fonti appartenenti alla famiglia.

Le struttura TextFont e altre informazioni richieste per definire tutte le fonti di una particolare famiglia sono contenute nei file definiti nella directory Sapphire. Le strutture FontContents dicono al sistema come fare per raggiungere e caricare ciascuno di questi file di fonti.

La struttura FontContentsHeader

La definizione della struttura FontContentsHeader è:

```
struct FontContentsHeader {
    UWORD fch_FileID;
    UWORD fch_Num_Entries;
};
```

Il parametro fch_FileID contiene un identificatore numerico indicante che il file che contiene la struttura FontContentsHeader è un file correlato a una fonte. Ora come ora, il suo valore è sempre il numero esadecimale 0xF00. Tale valore informa il sistema di trattare il file in questione come file di fonte.

La variabile fch_Num_Entries contiene il numero delle fonti contenute nella famiglia. Ciascuna avrà la propria struttura FontContents, il proprio file di fonte su disco, la propria struttura TextFont.

La struttura FontContents

La definizione della struttura FontContents è:

```
struct FontContents {
    char fc_FileName[MAXFONTPATH];
    UWORD fc_YSize;
    UBYTE fc_Style;
    UBYTE fc_Flags;
};
```

Il significato dei parametri della struttura FontContents è il seguente:

- il parametro fc_FileName[MAXFONTPATH] contiene il percorso di ricerca che l'AmigaDOS deve seguire per localizzare il file contenente le strutture DiskFontHeader e TextFont relative alla fonte considerata. Il percorso di ricerca potrebbe per esempio essere FONTS:sapphire/14. Questo informa il sistema di ricercare una subdirectory denominata sapphire nella directory logica FONTS: e di cercare il file 14 nella subdirectory sapphire. L'argomento MAXFONTPATH consente percorsi di ricerca lunghi fino a 256 caratteri. In tal modo è possibile specificare un nome di percorso directory/subdirectory anche molto complicato.

- Il parametro fc_YSize contiene l'altezza della fonte espressa in pixel. Appare anche nella struttura TextAttr.

- Il parametro fc_Style contiene lo stile della fonte. Appare anche nella struttura TextAttr.

- Il parametro fc_Flags è il parametro dei flag per la fonte. Appare anche nella struttura TextAttr.

La struttura DiskFontHeader

Ciascun file di fonte caricato dal disco è associato a una struttura DiskFontHeader che consente a tutti gli elementi di una famiglia di essere collegati insieme. La definizione della struttura DiskFontHeader è la seguente:

```
struct DiskFontHeader {
    struct Node dfh_DF;
    UWORD dfh_FileID;
    UWORD dfh_Revision;
    LONG dfh_Segment;
    char dfh_Name[MAXFONTNAME];
    struct TextFont dfh_TF;
};
```

Il significato dei parametri della struttura DiskFontHeader è il seguente:

- il parametro dfh_DF rappresenta una struttura Node usata per collegare fra loro tutte le fonti sul disco.
- Il parametro dfh_FileID contiene un valore (0x0F80) che consente al sistema di riconoscere che questo è il file di definizione di una fonte.
- Il parametro dfh_Revision identifica la revisione della fonte.
- Il parametro dfh_Segment contiene l'indirizzo di segmento per la fonte, quando questa viene caricata. I file per la definizione della fonte vengono caricati come se si trattasse di file eseguibili; comunque, il sistema non tenta mai di "mandare in esecuzione" un file di fonte.
- Il parametro dfh_Name[MAXFONTNAME] contiene una stringa di testo a terminazione nulla che rappresenta il nome della fonte. Ogni nome di fonte ha l'estensione .font. Per esempio, un nome valido è sapphire.font. La costante MAXFONTNAME vale 32 e indica il numero massimo di caratteri da inserire per il nome. Sei di essi sono dedicati ai caratteri ".font\0".
- Il parametro dfh_TF costituisce la struttura TextFont in questione.

OpenFont

Sintassi di chiamata della funzione

```
textFont = OpenFont (textAttr)
D0                  A0
```

Scopo della funzione

Questa funzione cerca nella lista delle fonti di sistema una struttura TextFont che corrisponda agli attributi specificati nella struttura TextAttr. Restituisce l'indirizzo della struttura TextFont più vicina a tali caratteristiche. Se viene trovata la fonte con il nome indicato ma non sono disponibili la grandezza o lo stile richiesti nella struttura TextAttr, viene restituito un puntatore alla struttura TextFont con gli attributi più simili. La funzione OpenFont restituisce il valore zero se non è reperibile la fonte indicata (o una simile).

Argomenti della funzione

textAttr Indirizzo della struttura TextAttr.

Discussione

La funzione OpenFont appartiene al gruppo di funzioni della libreria Graphics che gestiscono le fonti residenti in RAM.

Il sistema può contenere diverse fonti nello stesso momento. Alcune possono provenire da disco, mentre altre possono già trovarsi in RAM. Il procedimento di accesso a una fonte-carattere si chiama apertura della fonte. Il sistema ha due funzioni per aprire le fonti: la funzione OpenFont, destinata ad aprire fonti residenti in RAM e la funzione OpenDiskFont, per l'apertura delle fonti residenti su disco.

La fonte aperta tramite OpenFont o OpenDiskFont viene determinata unicamente dal contenuto della struttura TextAttr. A questo proposito, si vedano l'introduzione al capitolo e la spiegazione della funzione AskFont.

Le funzioni OpenFont e OpenDiskFont si basano sulle caratteristiche specificate nella struttura TextAttr per cercare nel sistema (in RAM o sul disco) una particolare struttura TextFont. Se non è rintracciabile una fonte che corrisponda esattamente a queste caratteristiche, OpenFont cercherà la fonte che più vi si avvicina. Ciò significa che se esistono fonti con il nome richiesto ma diverse come misura e stile, viene restituito un puntatore alla struttura TextFont che meglio corrisponde a quella cercata. Se la chiamata a OpenFont ha successo, la fonte è a disposizione del task che ha effettuato la chiamata.

Qualunque task può chiamare OpenFont o OpenDiskFont. In tal modo una fonte può essere assegnata simultaneamente a diversi task. Quando il task diventa attivo può accedere alla fonte e disegnare caratteri di testo nelle proprie bitmap; ogni task può anche avere diverse fonti aperte contemporaneamente.

Il puntatore alla struttura TextFont restituito dalla chiamata alla funzione OpenFont può essere impiegato in successive chiamate alle funzioni SetFont e CloseFont. È importante far seguire a ciascuna chiamata di OpenFont una chiamata a CloseFont. Ciò assicura il rilascio della memoria una volta che il task ha finito di utilizzare una fonte.

RemFont

Sintassi di chiamata della funzione

```
error = RemFont (textFont)
DØ              A1
```

Scopo della funzione

Questa funzione rimuove una fonte-carattere dalla lista di sistema.

Argomenti della funzione

textFont	Indirizzo della struttura TextFont.

Discussione

La funzione RemFont fa parte delle funzioni della libreria Graphics per la gestione delle fonti-carattere residenti in RAM.

Quando un task ha finito di utilizzare una determinata fonte, dovrebbe chiamare la funzione CloseFont per liberare le risorse di memoria a essa assegnate.

Inoltre, se nessun task fa più uso di una fonte, si dovrebbe chiamare la funzione RemFont per rimuoverla dalla lista delle fonti di sistema. Si osservi che, sebbene la funzione RemFont impedisca ad altri task di aprire la fonte rimossa, un task che la stia utilizzando può continuare a impiegarla, mantenendo semplicemente un puntatore alla struttura TextFont, che rimane in RAM insieme con gli altri dati della fonte. Quando anche quel task chiama CloseFont, la fonte viene rimossa dalla memoria.

SetFont

Sintassi di chiamata della funzione

```
error = SetFont (rastPort, textFont)
DØ              A1       AØ
```

Scopo della funzione

Questa funzione reimposta i parametri della fonte definiti nella struttura RastPort, sostituendoli con i valori specificati nella struttura TextFont. Sostituisce anche il puntatore alla struttura TextFont presente nella struttura RastPort con il valore specificato come secondo argomento di chiamata della funzione. Se l'argomento textFont vale 0, tutti i parametri correlati alla fonte nella struttura RastPort vengono azzerati.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
textFont	Indirizzo della struttura TextFont.

Discussione

La funzione SetFont è una delle otto funzioni della libreria Graphics che agiscono direttamente con le fonti residenti in RAM.

Il programma può avere a che fare contemporaneamente con diverse bitmap. Ciascuna di esse è associata a una struttura RastPort che controlla il disegno di grafica e testo. Cambiando, tramite SetFont, il puntatore alla struttura TextFont nella struttura RastPort, si possono inserire in ogni bitmap caratteri provenienti da diverse fonti-carattere.

SetFont rileva un insieme di attributi da una struttura TextFont e li inserisce nella struttura RastPort (a questo proposito si veda anche la spiegazione della funzione AskFont). Da quel momento in poi, finché non avviene un altro cambiamento, il disegno dei testi nella bitmap avrà le caratteristiche indicate da quella struttura TextFont.

Si noti che la fonte deve prima essere aperta attraverso la funzione OpenFont oppure OpenDiskFont.

SetSoftStyle

Sintassi di chiamata della funzione

```
newStyle = SetSoftStyle (rastPort, style, enable)
DØ                       A1       DØ    D1
```

Scopo della funzione

Questa funzione altera i bit di stile contenuti nel parametro AlgoStyle della struttura RastPort. L'argomento enable agisce come una maschera per variare i bit: possono essere alterati soltanto i bit di stile impostati a 1 nell'argomento enable. La funzione SetSoftStyle restituisce il nuovo stile impostato nella variabile newStyle.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
style	Variabile a 1 byte i cui bit rappresentano i nuovi bit di stile richiesti per la fonte.
enable	Maschera che indica i bit di stile destinati a essere alterati; è una variabile restituita da una precedente chiamata alla funzione AskSoftStyle.

Discussione

La funzione SetSoftStyle fa parte delle funzioni della libreria Graphics per la gestione delle fonti-carattere residenti in RAM. Si veda anche la spiegazione relativa alla funzione AskSoftStyle.

Si noti che, se i bit di stile intrinseci della struttura TextFont impediscono cambiamenti, le richieste di variazioni di stile saranno ignorate. Se per esempio è impostato il bit UNDERLINED (cioè il bit 0), non è possibile togliere la sottolineatura alla fonte-carattere. Il valore restituito dalla funzione costituisce la combinazione di due elementi: la modifica software indicata nell'argomento style e i bit di stile intrinseci nella struttura TextFont. Il sistema è a conoscenza

della condizione dei bit intrinseci grazie all'argomento enable, restituito dalla precedente chiamata alla funzione AskSoftStyle.

Ogni fonte possiede un insieme di bit di stile (si veda la spiegazione della funzione AskSoftStyle). SetSoftStyle cambia il valore dei bit generati in modo algoritmico in una struttura RastPort associata a un particolare task. Si ricordi che i bit generati in modo algoritmico per una specifica fonte-carattere sono contenuti nel parametro AlgoStyle della struttura RastPort. La funzione AskSoftStyle copia questo parametro e ne restituisce una versione modificata nella variabile enable.

La funzione SetSoftStyle può cambiare il parametro AlgoStyle della struttura RastPort. Ci sono tuttavia alcune restrizioni. Possono essere alterati soltanto i bit impostati a 1 nella variabile enable, così come viene restituita dalla funzione AskSoftStyle. Si supponga, per esempio, che una struttura TextFont contenga una fonte-carattere in neretto, sottolineato, corsivo ed esteso. In tal caso, AskSoftStyle restituisce forzatamente il valore 00000000 qualunque sia il valore del parametro AlgoStyle della struttura RastPort, e SetSoftStyle non può cambiare alcun bit del parametro AlgoStyle.

Supponiamo ora che la struttura TextFont contenga una fonte normale. In tal caso, AskSoftStyle può restituire valori diversi da zero (a secondo del valore presenti in AlgoStyle). Se AskSoftStyle restituisce il valore 3 (cioè 00000011), indicando che è possibile modificare questa fonte normale inserendo il sottolineato e il neretto, SetSoftStyle può cambiare il parametro AlgoStyle a 3 (cioè 00000011), 2 (00000010) oppure 1 (00000001), richiedendo rispettivamente il neretto sottolineato, il neretto o il sottolineato.

Possono essere alterati soltanto i bit impostati a zero nel parametro tf_Style dell'iniziale struttura TextFont. Se la fonte è normale (cioè ha il parametro tf_Style pari a 00000000), SetSoftStyle può cambiare qualunque bit di stile. Se la fonte è invece sottolineata, estesa, in corsivo e neretto (cioè ha il parametro tf_Style pari a 00001111), SetSoftStyle non può cambiare alcun bit nel parametro AlgoStyle della struttura RastPort. Tutte queste restrizioni sono regolate dall'argomento di maschera enable nella chiamata alla funzione SetSoftStyle.

Text

Sintassi di chiamata della funzione

```
error = Text (rastPort, stringPointer, count)
D0          A1         A0          D0: 0-15
```

Scopo della funzione

Questa funzione scrive caratteri di testo nella bitmap indicata, iniziando dalla posizione pixel corrente.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
stringPointer	Indirizzo della stringa di caratteri da inserire nella bitmap.
count	Numero di caratteri della stringa; se vale zero, non ci sono caratteri da inviare in output.

Discussione

Ci sono quattro funzioni della libreria Graphics che riguardano l'inserimento di testi in una bitmap: ClearEOL, ClearScreen, Text e TextLength. Grazie a esse i task possono inserire caratteri di testo nelle bitmap. Ogni bitmap è associata a una struttura RastPort che viene impiegata per controllare gli inserimenti di testo: in genere, i parametri per il controllo del disegno della struttura RastPort vengono impiegati per la grafica, ma alcuni servono anche per il disegno dei testi.

I modi di disegno per i testi

Uno dei parametri della struttura RastPort che controlla il disegno di testi è DrawMode. Se il parametro DrawMode è impostato a JAM1 i testi vengono disegnati nel colore della penna di primo piano, indicato dal parametro FgPen della struttura RastPort. Dove esiste un bit a 1 nella matrice che costituisce il carattere di testo, là il colore FgPen si sovrappone al bit corrispondente nella bitmap. Se il parametro DrawMode è impostato a JAM2 il colore di FgPen viene adoperato come colore per i pixel dei caratteri e il colore di BgPen viene usato come colore di sfondo per i pixel circostanti. Ogni carattere è contornato da un rettangolo di pixel. I pixel che compongono tale rettangolo vanno a sovrapporsi ai pixel della bitmap destinazione.

Se il parametro DrawMode è impostato a COMPLEMENT, i caratteri vengono disegnati in uno dei seguenti modi:

1. Per ogni bit del carattere che abbia valore 1, i corrispondenti pixel nella bitmap vengono invertiti.

2. Per ogni bit del carattere che abbia valore zero, i corrispondenti pixel nella bitmap rimangono inalterati.

Se è impostato il flag INVERSVID della struttura RastPort, tutti i bit del carattere vengono invertiti (i bit a 0 diventano a 1 e viceversa) prima che il carattere venga disegnato nella bitmap. Se in quel momento il modo di disegno è JAM2, i colori del carattere vengono invertiti.

Con la funzione Text viene presentato soltanto il testo sulla riga in cui si trova il cursore. Per tale motivo è necessario progettare attentamente le chiamate alla funzione Text. Se i caratteri della stringa oltrepassano i confini della bitmap, il testo eccedente viene tagliato. In tal caso, i parametri di posizione della penna (cioè cp_x e cp_y) vengono impostati al confine di bitmap, e non rappresentano più la posizione che avrebbe la penna se tutti i caratteri di testo fossero stati inseriti nella bitmap.

La funzione Text e i caratteri generati in modo algoritmico

La funzione Text dev'essere in grado di agire con due generi di caratteri: quelli che provengono direttamente dalla definizione della fonte senza modifiche (cioè i caratteri intrinseci della fonte) e i caratteri generati in modo algoritmico. Se per esempio la fonte originale è di tipo normale e si vogliono caratteri sottolineati, è necessario generarli impiegando la funzione SetSoftStyle.

D'altra parte, se si vogliono caratteri sottolineati e la definizione di fonte originaria è già per caratteri di questo tipo, si possono usare direttamente i caratteri della fonte senza modifiche. In questo caso, la misura interna dei caratteri e i dati di spaziatura nella struttura TextFont sono uguali per tutti i caratteri visualizzati attraverso la funzione Text.

I caratteri generati in modo algoritmico, invece, possono dare origine a qualche problema, in quanto la funzione Text calcola sempre la posizione e la larghezza del carattere basandosi sulla larghezza e lo spazio intercarattere definiti rispettivamente nei parametri tf_Size e tf_charSpace della struttura TextFont.

Dopo aver disegnato tutti i caratteri, la funzione Text mette automaticamente la penna di disegno nella posizione calcolata per il carattere successivo, basandosi su questi due parametri. Una simile procedura può causare la sovrapposizione dei caratteri generati in modo algoritmico e disegnati individualmente. Per esempio, caratteri corsivi generati algoritmicamente possono inclinarsi fino a entrare nel rettangolo del carattere successivo. Questo problema può essere superato prendendo le seguenti precauzioni:

1. Per aumentare la spaziatura tra i caratteri si inserisce l'opportuno numero di pixel nel parametro tx_Spacing della struttura RastPort. Conviene fare alcune prove per determinare la quantità più opportuna.

2. Si riuniscono i caratteri in una stringa di testo prima di chiamare la funzione Text. Per farlo si impiegano le funzioni di gestione delle stringhe del linguaggio C.

3. Si chiama la funzione Text per disegnare lo stile corretto con la giusta spaziatura tra caratteri.

TextLength

Sintassi di chiamata della funzione

```
numPixels = TextLength (rastPort, stringPointer, numchars)
DØ                     A1         AØ            DØ: Ø-15
```

Scopo della funzione

Questa funzione determina la lunghezza in pixel dei dati di testo da presentare in una bitmap. La lunghezza è basata sugli attributi della fonte indicati nella relativa struttura RastPort. TextLength restituisce la lunghezza in pixel della stringa di caratteri nella variabile numPixels.

Argomenti della funzione

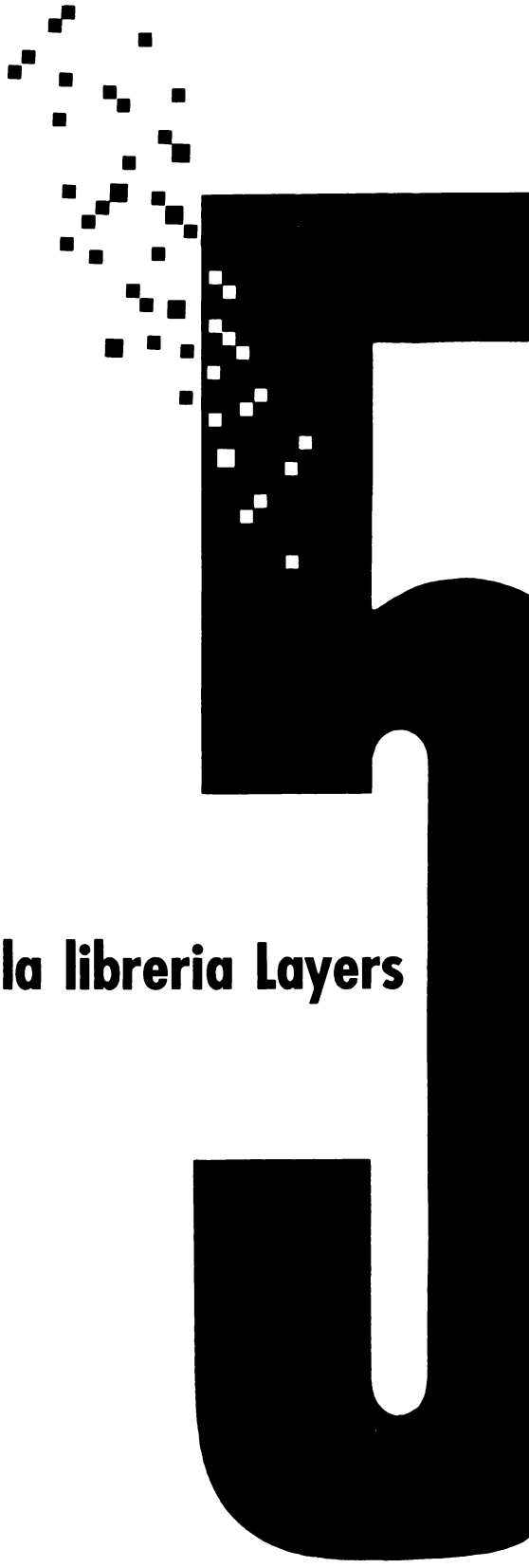
rastPort	Indirizzo della struttura RastPort di controllo.
stringPointer	Indirizzo della stringa di testo di cui dev'essere determinata la lunghezza.
numchars	Numero dei caratteri nella stringa di testo; il valore zero significa che la stringa non contiene caratteri.

Discussione

Ci sono quattro funzioni della libreria Graphics che riguardano la rappresentazione del testo in una bitmap: ClearEOL, ClearScreen, Text e TextLength. Il valore restituito dalla funzione TextLength è il numero dei pixel che il testo occupa nella direzione x, anche se l'origine della stringa di testo è

negativa (si veda il parametro tf_CharKern, illustrato con la funzione AddFont). Per determinare quale posizione di pixel risulterà per la stringa considerata dopo una chiamata alla funzione Text, si aggiunga il contenuto della variabile numPixels al valore del parametro cp_x della struttura RastPort. In tal modo è possibile determinare se una stringa sarà disegnata senza essere tagliata al limite destro della bitmap. Una volta che questo è stato accertato, si può chiamare la funzione Text per disegnare la stringa. Si noti che la variabile cp_y della struttura RastPort resta immutata dopo l'inserimento dei caratteri di testo.

Per capire meglio come i parametri della struttura RastPort influenzano la larghezza e la spaziatura dei caratteri inseriti nella bitmap, si veda anche l'introduzione a questo capitolo.



Le funzioni della libreria Layers

Introduzione

Questo capitolo tratta le funzioni della libreria Layers. Sono usate per creare e gestire i layer (strati) sullo schermo video dell'Amiga, producendo molti effetti interessanti. Infatti la capacità di gestire gli strati, rende unico il comportamento grafico dell'Amiga rispetto agli altri computer. Il miglior esempio è il comportamento delle finestre nel sistema Intuition, che sono tutte layer e quindi gestite attraverso le funzioni della libreria Layers.

Per capire le operazioni dei layer e le funzioni della libreria Layers, ci si deve impadronire prima di tutto della filosofia che caratterizza le operazioni di schermo e le capacità multi-layer dell'Amiga. La prima osservazione da fare è che lo schermo non è di per sé multi-layer. Gli strati multipli non esistono *effettivamente sullo schermo*, che al livello più basso è pur sempre una bitmap; i layer esistono invece in memoria come entità singole e indipendenti.

Che cosa c'è allora effettivamente sullo schermo? La semplice risposta è che lo schermo video è un insieme di rettangoli costituiti da pixel; ogni rettangolo può provenire da un layer diverso in memoria. Ciascun layer può essere associato a una propria bitmap, oppure diversi layer possono usare la stessa bitmap. Sullo schermo, in ogni momento, è presente un amalgama di rettangoli che provengono da molte diverse bitmap di layer in memoria.

Per rendere più chiari questi concetti, basta osservare come lavora il sistema di schermi e finestre di Intuition. Ci soffermiamo sull'esempio di Intuition e sugli insegnamenti che se ne possono trarre, perché Intuition offre lo stesso tipo di effetti video che si possono ottenere nei propri programmi. Per semplicità, nell'esempio prendiamo in considerazione soltanto schermi personalizzati di Intuition, e supponiamo che un solo task grafico abbia il controllo dello schermo video Intuition.

Riprendiamo i concetti esposti nel capitolo 2, ricordando che tutto ciò che l'utente vede sul video in un dato istante costituisce una view. Può trattarsi della view creata da Intuition per default, o di una view creata da un task, ma comunque in ogni istante è attiva sempre e solo una view; due o più view non possono condividere lo schermo. Prendiamo il caso della view gestita per default da Intuition e analizziamola.

Intuition permette di aprire diversi schermi di vario tipo, di sovrapporli fra loro e muoverli verticalmente, mostrando così contemporaneamente view parziali di più schermi nella view attiva. Non è difficile rendersi conto che questi schermi sono delle viewport, cioè delle strisce orizzontali della view dotate di caratteristiche proprie, come la palette dei colori e la risoluzione. Basta osservare che gli schermi, come le viewport, devono necessariamente occupare l'intera larghezza dello schermo video, mentre verticalmente possono occupare qualsiasi posizione e qualsiasi numero di linee di scansione. Un altro particolare che ci ricorda le viewport è che la linea di demarcazione che in Intuition separa gli schermi l'uno dall'altro, quando appaiono sovrapposti, è costituita da un certo numero di linee di schermo nere (cioè linee durante la cui scansione il pennello elettronico mostra il colore 0).

Intuition, con questa organizzazione a una view e diverse viewport, svolge alcune operazioni di gestione che non vengono compiute automaticamente dalla libreria Graphics. Una in particolare è la procedura che simula la sovrapposizione a più livelli di profondità degli schermi. In pratica, ogni viewport è definita da una bitmap grande quanto lo schermo, ma in ogni istante ne è visibile solo una parte, o anche nessuna, a seconda di come l'utente ha disposto gli schermi tramite il mouse. Quando l'utente afferra uno schermo e lo muove verso il basso, Intuition modifica l'aspetto della view riducendo la parte visibile della viewport spostata, e aumentando quella della viewport sottostante. Compiendo queste operazioni, Intuition simula efficacemente la sovrapposizione degli schermi sfruttando le viewport, che per loro natura non sono sovrapponibili, ma solo affiancabili.

Su ogni schermo di Intuition è possibile aprire un numero qualsiasi di finestre, sovrapponendole come si desidera. Ogni finestra è un layer. Intuition è in grado di spostare la finestra, di ridimensionarla, di cambiarle il livello di profondità rispetto alle altre, di controllare un insieme di gadget a essa associati, e svolge queste funzioni servendosi per lo più delle funzioni della libreria Layers. Quando due finestre si sovrappongono, quella appartenente al layer più profondo risulta parzialmente nascosta all'osservatore.

In ogni dato momento, poi, lo schermo di Intuition mostra un insieme di complesse informazioni grafiche provenienti da diversi layer in memoria. Il programma può azzerare qualsiasi pixel, o individuare come quel pixel è stato portato là e da dove proviene. Ogni pixel è prodotto usando una determinata bitmap di layer che ne stabilisce il colore; ciò significa che, in ogni momento, ciascun pixel è associato con una e una sola bitmap di layer.

Se lo schermo video è statico e non viene alterato, si può individuare lo specifico layer dal quale dipende ogni pixel che compone la schermata. Per avere un'idea più chiara di come lavorano i layer, è utile ricondurre ogni parte dello schermo video alla bitmap di layer in cui è definita. Il primo passo consiste nel realizzare uno schema delle assegnazioni pixel-layer. Si può poi mettere insieme questo schema con la posizione di ogni bitmap di layer in memoria. Esaminando quindi le informazioni di bitplane si può scoprire qual è l'origine in memoria di ogni pixel. In pratica, quest'analisi può essere svolta facilmente attraverso la funzione WhichLayer.

S'immagini di alterare lo schermo spostando una finestra in una nuova posizione. Nel farlo si espongono nuove aree di altre finestre sottostanti. Ora, se si è creata una mappa di layer per i pixel, relativa all'intero schermo, si può osservare che la mappa è cambiata per le parti di schermo alterate. La finestra spostata ha coperto qualcosa e scoperto qualcos'altro. Siccome la finestra è rettangolare, le aree in cui si sono verificati i cambiamenti sono pure rettangolari.

Nello studio delle funzioni della libreria Layers si tenga presente questo quadro generale perché servirà come aiuto alla comprensione di come operano queste funzioni. Questo capitolo risponderà ad alcune domande fondamentali: da dove provengono le informazioni di pixel di ciascuna posizione di schermo e che cosa succede loro quando lo schermo subisce un cambiamento?

layer e la gestione multitasking

Esiste un altro importante aspetto del comportamento dei layer. Riguarda il caso in cui diversi task accedono in scrittura allo stesso layer. La libreria Layers fornisce diverse funzioni che permettono di controllare in modo efficiente questa evenienza. Se il programma è progettato in modo che più di un task possa disegnare nella stessa bitmap di layer, ogni task potrebbe in un determinato momento cercare d'impedire agli altri d'intervenire nel layer considerato. È questo lo scopo delle funzioni di gestione delle interazioni tra task presenti nella libreria Layers. Esse consentono a un task di bloccare le operazioni di disegno di altri task su una specifica bitmap di layer. Questo blocco è necessario perché l'esatto percorso di esecuzione seguito da un determinato task non è sempre prevedibile; a volte un task, nel momento in cui acquisisce il controllo della macchina, esegue una sequenza di disegno, a volte no. Perciò, il sistema ha bisogno di un meccanismo esterno al task per impedirgli di disegnare in particolari bitmap di layer nel momento in cui prende il controllo dell'elaborazione.

Ogni volta che un task acquisisce il controllo della CPU e tenta di eseguire un'istruzione di disegno su un layer bloccato, il sistema lo manda in attesa. Solo quando il task che ha effettuato il blocco acquisisce nuovamente il controllo della CPU e provvede a sbloccare il layer, diventa possibile per il secondo task riottenere il controllo e procedere con le istruzioni di disegno in quella bitmap di layer.

Questi concetti diverranno più chiari leggendo le spiegazioni relative alle funzioni LockLayer, LockLayers, LockLayerInfo, UnlockLayer, UnlockLayers e UnlockLayerInfo.

tipi di layer

La libreria Layers opera con quattro tipi di layer: i layer backdrop (di ultimo piano), i layer simple-refresh (a refresh semplice), i layer smart-refresh (a refresh avanzato), e i layer superbitmap. Un layer backdrop può essere combinato con ciascuno degli altri tre tipi; ciò significa che si può avere un layer backdrop che è anche a refresh semplice, a refresh avanzato oppure superbitmap. I layer a refresh semplice, a refresh avanzato e superbitmap sono, invece, mutuamente esclusivi; ogni layer può appartenere soltanto a uno di questi tre tipi.

Studiando la definizione e il modo di operare di questi tipi di layer, si inizia a comprendere come le bitmap di layer vengono gestite per produrre la cangiante composizione a strati dello schermo video dell'Amiga.

I layer a refresh semplice

Il layer a refresh semplice (simple-refresh) è il solo tipo di layer che non impiega una bitmap addizionale per conservarvi le parti oscurate da ripristinare quando tornano alla luce. In altre parole, quando un'area nascosta dello schermo appartenente a un layer a refresh semplice viene nuovamente esposta, è il task che deve ridisegnare la porzione oscurata dello schermo.

Ciò significa che il task deve eseguire le funzioni di disegno della libreria Graphics per riprodurre l'area in questione. Se le procedure di disegno del task agiscono su ampie aree della bitmap di layer, incluse quella al di fuori dell'area che era stata nascosta, la funzione BeginUpdate della libreria Layers limita automaticamente il disegno ai confini rettangolari dell'area oscurata e tornata alla luce. Questa procedura evita di disegnare inutilmente sulle porzioni già visibili del layer a refresh semplice, con una sensibile riduzione dei tempi necessari per effettuare il ripristino. La gestione del processo di delimitazione dell'area oscurata per il successivo ripristino è effettuata dal software sistema, purché sia informato che si tratta di un layer a refresh semplice.

I layer a refresh avanzato

I layer a refresh avanzato (smart-refresh) sono dotati di una o più bitmap addizionali per le porzioni oscurate, in modo da mantenere sempre una copia delle informazioni contenute nella parti oscurate da altri layer. Ciò in genere interessa un insieme di piccole bitmap sparse attraverso lo spazio di memoria disponibile.

Seguendo i cambiamenti effettuati sullo schermo video dall'utente o dal programma, varie parti di diversi layer vengono nascoste ed esposte. Per ogni area appena esposta, la relativa piccola bitmap di salvataggio viene automaticamente sostituita nella memoria video per ripristinare la parte di layer tornata visibile.

Si noti che se il task disegna sulle parti nascoste della bitmap, queste conterranno i nuovi disegni quando verranno ripristinate. Ciò significa che il task può continuare a disegnare nel layer anche quando è parzialmente o del tutto oscurato, sicuro che nulla di quanto disegna andrà perso.

I layer superbitmap

I layer superbitmap hanno una sola grande bitmap per le parti oscurate, a differenza del layer a refresh avanzato che impiega una o più piccole bitmap. Un vantaggio del layer superbitmap è che consente di effettuare al suo interno lo scroll del contenuto della più grande bitmap (appunto detta superbitmap). Né il layer a refresh semplice né quello a refresh avanzato permettono di effettuare lo scroll.

La superbitmap può essere anche più grande della bitmap del layer. In ogni momento la superbitmap può contenere alcune o tutte le informazioni presenti nella bitmap di layer. Generalmente, la superbitmap contiene un

esatto duplicato di tutti i bit della bitmap di layer, più altri bit addizionali che definiscono i pixel esterni alla bitmap di layer; ciò significa che, in ogni momento, i pixel della superbitmap costituiscono un super-insieme dei pixel della bitmap di layer; quest'ultima agisce come una finestra nella superbitmap.

In ogni momento, perciò, la superbitmap contiene due categorie d'informazioni. La prima è costituita dalle informazioni corrispondenti alla bitmap di layer presente sullo schermo. Questo sotto-insieme d'informazioni è la copia esatta delle informazioni contenute nella bitmap. La posizione x,y dell'origine della bitmap all'interno della superbitmap definisce la parte di superbitmap visibile nel layer. Per definire la posizione e le dimensioni di questa "finestra" sulla superbitmap si utilizzano le funzioni ScrollLayer e SizeLayer.

La seconda categoria d'informazioni di superbitmap è costituita dai dati di bitmap relativi ai pixel che non appartengono alla bitmap di layer sullo schermo. In alcuni casi, sono le parti di superbitmap uscite dallo schermo per effetto della funzione ScrollLayer. In altri casi sono soltanto sezioni di superbitmap non ancora presentate sullo schermo. La misura di questa seconda categoria di pixel di superbitmap varia a seconda dei parametri della funzione SizeLayer.

È importante capire che la superbitmap esiste in modo completamente indipendente dalla bitmap di layer. Ecco perché si può fare in modo che i task disegnino, in ogni momento, in tutte le aree dei bitplane appartenenti alla superbitmap. In tal senso, un layer superbitmap è un autentico esempio di double-buffer per lo schermo video in cui il secondo buffer, quello nascosto, è più grande del primo, il cui contenuto è visualizzato sullo schermo.

I layer backdrop

I layer di ultimo piano (backdrop) appaiono sempre dietro a tutti gli altri. Si noti che la natura dei layer backdrop impedisce di ridimensionarli, di spostarli o di variarne la profondità. Si ricordi inoltre che il layer di tipo backdrop può essere combinato con ciascuno degli altri tre tipi, per creare layer backdrop a refresh semplice, layer backdrop a refresh avanzato e infine layer backdrop a superbitmap. Si noti infine che si può creare e annullare un layer backdrop semplicemente agendo sul flag backdrop della struttura Layer. In un certo senso, così facendo si possono ridimensionare, spostare e riordinare in profondità anche i layer backdrop.

Le strutture della libreria Layers

La libreria Layers opera in particolare con tre strutture: la struttura Layer, la struttura Layer_Info e la struttura ClipRect. Interagisce inoltre con la consueta struttura RastPort di controllo.

La struttura Layer

La struttura Layer viene impiegata per definire le caratteristiche dei layer nel sistema. Sostanzialmente contiene informazioni per definire la posizione di un layer nel suo ordine di stack. Contiene inoltre informazioni che consentono al sistema di definire e controllare i rettangoli di delimitazione (clipping-rectangle) associati al layer, e di bloccare il layer quando diversi task sono in competizione per accedere alla sua bitmap. La struttura Layer contiene diversi parametri chiave che conviene mettere in evidenza. Il primo è un insieme di due puntatori a strutture Layer (front e back, rispettivamente il puntatore al layer di livello superiore a quello di livello inferiore) che permettono a ciascuna struttura Layer di essere mantenuta in una lista concatenata.

Si noti inoltre che la struttura Layer contiene un puntatore alla struttura RastPort di controllo e un puntatore opzionale a una struttura SuperBitMap che definisce le caratteristiche di un layer superbitmap. Il quarto parametro rilevante è il parametro Flags, che può assumere quattro valori diversi per definire il tipo di layer.

La definizione della struttura Layer è inclusa nella trattazione della funzione BehindLayer e nei file INCLUDE denominati clip.h per il linguaggio C e clip.i per il linguaggio Assembly.

La struttura Layer_Info

Lo scopo della struttura Layer_Info è fornire un meccanismo per controllare le operazioni di disegno su un gruppo correlato di layer. In particolare, la struttura Layer_Info fornisce ai task il meccanismo per bloccare e sbloccare l'intero insieme di layer associati alla struttura. Inoltre, la stessa struttura Layer_Info è bloccabile, e quindi contiene parametri anche per questa evenienza.

Quando si crea un layer, bisogna sempre indicare a quale struttura Layer_Info dev'essere associato. In questo modo, i layer che appartengono allo stesso insieme si possono bloccare anche globalmente, con un'unica chiamata di funzione.

La definizione della struttura Layer_Info è inclusa nella trattazione della funzione NewLayerInfo e nei file INCLUDE denominati layers.h per il linguaggio C e layers.i per il linguaggio Assembly.

La struttura ClipRect

La struttura ClipRect ha lo scopo di mantenere una lista di rettangoli di delimitazione associati a un layer. Tali rettangoli definiscono le porzioni di layer che devono essere aggiornate quando eventuali parti nascoste tornano a essere visibili sullo schermo. I più importanti parametri della struttura ClipRect sono il puntatore alla struttura Layer, i punti di confine del rettangolo di delimitazione e i due puntatori a strutture ClipRect. Questi due ultimi parametri puntano alle strutture ClipRect precedente e seguente in una lista

concatenata di rettangoli di delimitazione mantenuta dalla struttura ClipRect considerata.

La definizione della struttura ClipRect è inclusa nella trattazione della funzione UpfrontLayer e nei file INCLUDE denominati clip.h per il linguaggio C e clip.i per il linguaggio Assembly.

La struttura RastPort

Infine, è importante ricordare che le funzioni per la creazione di layer (CreateUpfrontLayer e CreateBehindLayer) assegnano sempre una struttura di controllo RastPort al layer appena creato. Si può individuare l'indirizzo della struttura RastPort di un task accedendo al parametro rp della struttura Layer. Questo indirizzo è fondamentale se si desidera disegnare all'interno del layer, dal momento che tutte le funzioni di disegno lo richiedono espressamente. Sulla struttura RastPort si veda anche il capitolo 2.

La creazione e la gestione dei layer

La Figura 5.1 mostra due bitmap di dimensioni diverse. La prima dispone di quattro layer; la seconda di tre.

Nella prima bitmap i layer sono disposti nel seguente ordine:

1. Il layer 1 è stato inserito usando la funzione CreateUpfrontLayer.
2. Il layer 2 è stato inserito usando la funzione CreateBehindLayer.
3. Il layer 3 è stato inserito usando la funzione CreateUpfrontLayer.
4. Il layer 4 è stato inserito usando la funzione CreateBehindLayer.

Ciò produce una bitmap con quattro layer ordinati secondo la profondità come si vede nella Figura 5.1, in cui i confini invisibili dei layer sono definiti da linee più sottili. Tale ordine di creazione dei layer produce anche diversi rettangoli di delimitazione: il rettangolo di delimitazione 1 viene creato quando il layer 2 si trova dietro al layer 1; il rettangolo di delimitazione 2 viene creato quando il layer 3 si trova davanti al layer 1; i rettangoli di delimitazione 3 e 4 vengono creati quando il layer 4 si trova dietro ai layer 2 e 3.

Nella seconda bitmap i layer sono disposti nel seguente ordine:

1. Il layer 1 è stato inserito usando la funzione CreateUpfrontLayer.
2. Il layer 2 è stato inserito usando la funzione CreateUpfrontLayer.

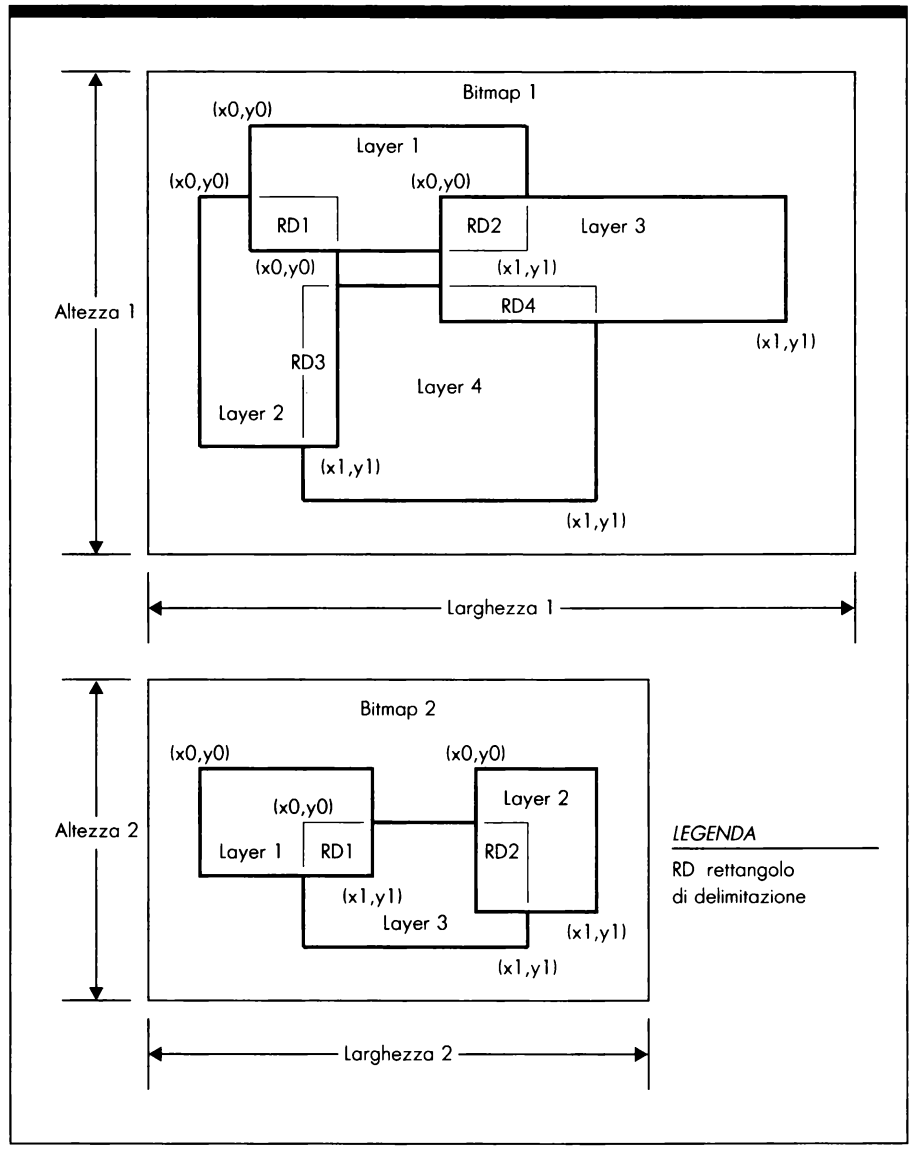


Figura 5.1: *Bitmap, layer e rettangoli di delimitazione: creazione dei layer*

3. Il layer 3 è stato inserito usando la funzione CreateBehindLayer.

Ancora una volta l'ordine di inserimento dei layer produce uno specifico insieme di rettangoli di delimitazione. Entrambi i rettangoli di delimitazione 1 e 2 vengono creati quando il layer 3 viene posto dietro ai layer 1 e 2.

Quando queste definizioni di layer sono complete, si possono impiegare le funzioni di disegno della libreria Graphics per inserire informazioni grafiche nei

layer; si possono anche adoperare le funzioni della libreria Layers per alterare il loro ordine di presentazione (l'ordine di profondità) e ridimensionarli. Se i layer sono dichiarati come layer superbitmap è anche possibile utilizzarli come finestre in grado di spostarsi su un'immagine più vasta.

È importante comprendere che il numero, la posizione, la misura e altre caratteristiche dei rettangoli di delimitazione si alterano seguendo la manipolazione dei layer effettuata attraverso le funzioni della libreria Layers; la libreria Layers registra automaticamente queste alterazioni. Quando i layer considerati vengono cambiati, le routine della libreria Layers provvedono automaticamente a ricostruire una corretta presentazione video.

BeginUpdate

Sintassi di chiamata della funzione

BeginUpdate (layer)
AØ

Scopo della funzione

Questa funzione predispone il sistema a riparare un layer a refresh semplice. Sostituisce il puntatore alla struttura DamageList con il puntatore alla struttura ClipRect della struttura Layer, scambiando così la lista dei rettangoli di delimitazione con la lista dei danni; in questo modo, quando un task va a disegnare nel layer, il disegno avviene solo nelle porzioni danneggiate, anziché solo all'interno dei rettangoli di delimitazione.

Argomenti della funzione

layer	Indirizzo della struttura Layer che rappresenta il layer a refresh semplice.

Discussione

Le funzioni BeginUpdate e EndUpdate consentono di controllare l'aggiornamento grafico dei layer a refresh semplice. Si veda anche la spiegazione della funzione EndUpdate.

La funzione BeginUpdate si usa quando si vogliono effettuare cambiamenti nei pixel delle parti nascoste di un layer a refresh semplice. Questi cambiamenti comprendono qualsiasi tipo di disegno ottenuto con le funzioni grafiche della libreria Graphics.

La funzione BeginUpdate si usa soltanto per i layer a refresh semplice, gli unici a non possedere una bitmap che conservi le parti nascoste della loro bitmap su schermo. Ne consegue che quando uno di questi layer viene oscurato durante operazioni di spostamento, ridimensionamento, scorrimento o sovrapposizione, le informazioni grafiche nell'area nascosta vanno perse. Al contrario, le aree oscurate di un layer a superbitmap o a refresh avanzato vengono salvate in uno o più buffer.

Se uno dei task vuole disegnare in un layer a refresh semplice mentre è parzialmente o interamente nascosto, la parte oscurata non riceve informazioni di disegno, dato che non esiste alcun buffer in grado di ricevere le nuove definizioni dei pixel. Ecco l'occasione in cui viene usata BeginUpdate.

Prima di ridisegnare le parti nascoste di un layer a refresh semplice durante un aggiornamento grafico di un layer, è necessario identificarle. Questo è il lavoro della lista dei danni. Una lista dei danni è semplicemente una lista di rettangoli di delimitazione che comprende tutte le parti nascoste (danneggiate) di un layer a refresh semplice.

La procedura generale per ridisegnare soltanto le parti oscurate di un layer a refresh semplice danneggiato, impiegando la funzione BeginUpdate, è la seguente:

1. Si chiama la funzione BeginUpdate. BeginUpdate salva automaticamente l'indirizzo contenuto nel puntatore alla struttura ClipRect e lo rimpiazza con l'indirizzo della struttura DamageList relativa al layer. Il sistema tiene automaticamente aggiornata la struttura DamageList quando viene manipolato un layer a refresh semplice, tenendo quindi conto di tutte le sue sezioni danneggiate (nascoste). Una volta che il sistema possiede un puntatore alla struttura DamageList, le funzioni grafiche adoperano la lista dei danni per definire l'insieme dei rettangoli di delimitazione nei quali possono disegnare.

2. Si eseguono le funzioni di disegno della libreria Graphics. Si possono, per esempio, adoperare le funzioni AreaDraw, AreaMove e AreaEnd per definire i nuovi contenuti grafici delle aree selezionate del layer a refresh semplice. Anche se le funzioni di disegno vengono specificate per l'intera superficie del layer, come effetto della lista dei danni, le funzioni grafiche disegnano soltanto nelle sezioni danneggiate; ciò significa che le operazioni di disegno risultano delimitate. Se si tratta di diverse piccole aree, il processo di disegno può essere anche molto rapido. Nessun tempo viene sprecato per disegnare in sezioni del layer che non sono state danneggiate.

3. Si chiama la funzione EndUpdate per dichiarare concluso il processo di disegno. EndUpdate restituisce ai puntatori alle strutture DamageList e ClipRect i loro valori originali. A questo punto altre parti del layer possono essere ridisegnate nel solito modo.

Per la descrizione della struttura Layer, si veda la spiegazione della funzione CreateBehindLayer.

BehindLayer

Sintassi di chiamata della funzione

```
success = BehindLayer (dummy, layer)
D0                     A0     A1
```

Scopo della funzione

Questa funzione dispone il layer dietro a tutti gli altri. Ciò viene ottenuto scambiando i bit con quelli degli altri layer, interni ed esterni alla bitmap di schermo. Se i layer sono a refresh semplice, la funzione BehindLayer raccoglie le loro liste dei danni e imposta i flag LAYERREFRESH di quei layer nei quali alcune sezioni nascoste tornano alla luce sullo schermo. I task che li controllano, verificando che il flag è impostato, devono provvedere ad effettuare il refresh "manuale" delle parti tornate in vista. Se il layer specificato è di tipo backdrop, la funzione BehindLayer lo colloca dietro a tutti gli altri layer backdrop; in caso contrario dispone il layer considerato davanti al layer backdrop superiore e dietro a tutti gli altri layer normali. Questa funzione restituisce un valore booleano: TRUE se si conclude con successo; FALSE in caso contrario.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer.

Discussione

Le funzioni BehindLayer, UpfrontLayer e MoveLayerInFrontOf riguardano la posizione relativa dei layer nello stack, che imposta la priorità video per lo schermo. Queste funzioni consentono di riordinare i layer, cosa particolarmente utile in un ambiente a finestre come Intuition o in programmi applicativi che vogliano emulare le finestre e gli schermi caratteristici di Intuition. Ogni volta che si desidera alterare l'ordine di stack dei layer, si deve ricorrere a una di queste tre funzioni.

È importante notare che la funzione BehindLayer prende in considerazione tutti i layer backdrop presenti nel sistema. Si noti inoltre che il sistema chiama automaticamente le funzioni LockLayers e LockLayerInfo durante le operazioni di riordino dello stack. Ciò impedisce a tutti gli altri layer del sistema di effettuare cambiamenti mentre si è impegnati a spostare quel particolare layer verso la posizione più arretrata.

Per la descrizione della struttura Layer si veda la spiegazione della funzione CreateBehindLayer.

CreateBehindLayer

Sintassi di chiamata della funzione

```
layer = CreateBehindLayer (layer_Info, bitMap, xØ, yØ, x1, y1, flags, [bitMap2])
DØ                         AØ          A1      DØ  D1  D2  D3  D4     [A2]
```

Scopo della funzione

Questa funzione crea un nuovo layer del tipo indicato nell'argomento flags. Se si tratta di un layer superbitmap, questa funzione adopera l'argomento bitMap2 come indirizzo della struttura BitMap che definisce la superbitmap da associare al layer, e copia il contenuto della superbitmap nella bitmap di layer. Se il layer considerato è di tipo backdrop, viene posto dietro a tutti gli altri, inclusi altri layer backdrop. Se non si tratta di un layer backdrop, viene posto dietro a tutti gli altri layer non backdrop. La funzione CreateBehindLayer restituisce l'indirizzo della struttura Layer associata al layer che ha creato.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.
bitMap	Indirizzo della struttura BitMap nella cui bitmap deve apparire il nuovo layer.
xØ, yØ	Coordinate dell'angolo superiore sinistro del layer.
x1, y1	Coordinate dell'angolo inferiore destro del layer.
flags	Insieme di bit che indicano diversi tipi di layer. Indicando un layer di tipo LAYERSUPER in questo campo, è necessario impostare anche il flag LAYERSMART.
bitMap2	[opzionale] Indirizzo di una seconda struttura BitMap qualora la variabile flags indichi che si tratta di un layer superbitmap.

Discussione

Ci sono tre funzioni che consentono di creare ed eliminare layer. Le funzioni CreateBehindLayer e CreateUpfrontLayer forniscono il solo meccanismo possibile per la creazione di layer. Queste due funzioni creano un nuovo layer e lo dispongono dietro a tutti gli altri (CreateBehindLayer) oppure davanti (CreateUpfrontLayer). La funzione DeleteLayer fornisce il solo modo possibile per cancellare un layer da una lista di layer.

In genere si creano e s'inseriscono insiemi di layer. Tutti i layer di un dato insieme possono apparire sulla stessa bitmap. Ciò significa che tutti i layer vengono creati con lo stesso argomento BitMap tramite la funzione CreateBehindLayer (oppure CreateUpfrontLayer). La sola differenza tra questi layer è la posizione dell'angolo superiore sinistro (x0,y0) e la posizione dell'angolo inferiore destro (x1,y1). Il risultato è che tutti i layer dell'insieme appaiono nella stessa bitmap. È un concetto importante per l'uso delle funzioni della libreria Layers.

Nel creare diversi layer che condividono tutti la stessa bitmap, si può produrre l'effetto di stratificazione dei layer menzionato nell'introduzione al capitolo. Ciascun nuovo layer si sovrappone parzialmente o si pone al di sotto di altri layer già definiti. Più layer si aggiungono a una particolare bitmap, più numerosi sono i casi di layer sovrapposti. Quando la bitmap viene visualizzata sullo schermo, ciascun sotto-rettangolo visibile nella stratificazione completa dello schermo può appartenere a un layer diverso.

Inoltre alcuni di questi layer (o anche tutti) possono condividere la stessa struttura Layer_Info, se nella chiamata alla funzione CreateBehindLayer

oppure CreateUpfrontLayer viene usato lo stesso argomento puntatore a Layer_Info. In tal caso, tutti i layer associati a una determinata struttura Layer_Info sarebbero trattati come un'entità unica dalle altre funzioni della libreria Layers. Si noti che non è necessario che i due gruppi (layer che condividono la stessa bitmap e layer che condividono la stessa struttura Layer_Info) siano composti dagli stessi elementi.

La struttura Layer

La definizione della struttura Layer è la seguente:

```
struct Layer {
        struct Layer *front, *back;
        struct ClipRect *ClipRect;
        struct RastPort *rp;
        struct Rectangle bounds;
        UBYTE reserved[4];
        UWORD priority;
        UWORD Flags;
        struct BitMap *SuperBitMap;
        struct ClipRect *SuperClipRect;
        APTR Window;
        SHORT Scroll_X, Scroll_Y;
        struct ClipRect *cr, *cr2, *crnew;
        struct ClipRect *SuperSaveClipRects;
        struct ClipRect *_cliprects;
        struct Layer_Info *LayerInfo;
        struct SignalSemaphore Lock;
        UBYTE reserved3[8];
        struct Region *ClipRegion;
        struct Region *saveClipRects;
        UBYTE reserved2[22];
        struct Region *DamageList;
};
```

Ecco il significato di alcuni parametri della struttura Layer:

- i parametri front e back puntano alle strutture Layer che definiscono i layer disposti rispettivamente davanti e dietro al layer in questione.
- Il parametro ClipRect punta alla struttura ClipRect associata al layer.
- Il parametro rp punta alla struttura RastPort di controllo associata al layer.

- Il parametro bounds è una struttura Rectangle nella quale sono contenuti un insieme di confini per i rettangoli di delimitazione da impiegare con la struttura ClipRect associata al layer.

- Il parametro Flags contiene 16 bit che determinano il tipo di layer e indicano se il layer ha delle parti oscurate.

- Il parametro SuperBitMap punta a una struttura BitMap per una superbitmap, qualora questo layer sia di tipo superbitmap.

- Il parametro SuperClipRect punta a una struttura ClipRect associata a una superbitmap, qualora questo layer sia di tipo superbitmap.

- Il parametro Window punta a una struttura Window di Intuition.

- I parametri Scroll_X e Scroll_Y contengono le distanze di scroll in pixel per le direzioni x e y; servono per l'operazione di scroll relativa al layer di superbitmap.

- I parametri cr, cr2 e crnew puntano ciascuno a una struttura ClipRect che il sistema impiega per gestire i rettangoli di delimitazione associati al layer.

- Il parametro SuperSaveClipRects punta a una struttura ClipRect che rappresenta un insieme di rettangoli di delimitazione pre-assegnati dal sistema per un layer superbitmap.

- Il parametro _cliprects punta a una struttura ClipRect. Viene usato soltanto dal sistema durante il refresh.

- Il parametro LayerInfo punta alla struttura Layer_Info della cui lista il layer fa parte.

- Il parametro Lock è una struttura SignalSemaphore che definisce il semaforo di segnalazione impiegato per interdire l'accesso indiscriminato a questo layer.

- Il parametro DamageList punta a una struttura Region; è usato soltanto per i layer a refresh semplice e delimita le aree sulle quali occorre effettuare il refresh.

CreateUpfrontLayer

Sintassi di chiamata della funzione

```
layer = CreateUpfrontLayer (layer_Info, bitMap, xØ, yØ, x1, y1, flags [bitMap2])
DØ                          AØ          A1      DØ  D1  D2  D3  D4    [A2]
```

Scopo della funzione

Questa funzione crea un nuovo layer del tipo indicato nell'argomento flags e lo colloca davanti a tutti. Se si tratta di un layer a superbitmap, questa funzione adopera l'argomento bitMap2 come indirizzo della struttura BitMap che definisce la superbitmap da associargli, e copia il contenuto della superbitmap nella bitmap di layer. Se il layer è di tipo backdrop, viene posto davanti a tutti i layer dello stesso tipo eventualmente presenti, ma dietro a tutti quelli normali. La funzione CreateUpfrontLayer restituisce l'indirizzo della struttura Layer associata al layer che ha creato.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.
bitMap	Indirizzo della struttura BitMap nella cui bitmap deve apparire il layer.
xØ, yØ	Coordinate dell'angolo superiore sinistro del layer.
x1, y1	Coordinate dell'angolo inferiore destro del layer.
flags	Insieme di bit che indicano diversi tipi di layer. Se viene specificato il flag LAYERSUPER, dev'essere impostato anche il flag LAYERSMART.
bitMap2	[opzionale] Indirizzo di una seconda struttura BitMap qualora la variabile flags indichi che si tratta di un layer superbitmap.

Discussione

Ci sono tre funzioni che consentono di creare ed eliminare layer. Le funzioni CreateBehindLayer e CreateUpfrontLayer forniscono il solo meccanismo utilizzabile per creare layer. Queste due funzioni servono per creare un nuovo layer e disporlo dietro a tutti gli altri (CreateBehindLayer) oppure davanti (CreateUpfrontLayer). La funzione DeleteLayer fornisce il solo modo utilizzabile per cancellare un layer da una lista di layer. Per una spiegazione più approfondita, si veda la funzione CreateBehindLayer.

DeleteLayer

Sintassi di chiamata della funzione

```
DeleteLayer (dummy, layer)
            A0      A1
```

Scopo della funzione

Questa funzione cancella il layer specificato da una lista di layer mantenuta dal sistema. Nel farlo, la funzione DeleteLayer libera la memoria associata alla sua bitmap di layer e alla relativa struttura Layer.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer.

Discussione

La funzione DeleteLayer fornisce il solo modo possibile per cancellare un layer da una lista di layer mantenuta da una struttura Layer_Info. Ricostituisce inoltre la presentazione video degli altri layer che possono essere stati precedentemente nascosti dal layer considerato, ricostruendo le parti tornate in vista. Se si tratta di un layer superbitmap, ci si deve accertare che la

definizione di superbitmap sia aggiornata. Quando la funzione DeleteLayer restituisce il controllo, la superbitmap del layer continua a essere disponibile nel sistema; non viene rimossa. È perciò possibile usare nel task, in un secondo momento, le funzioni CreateBehindLayer e CreateUpfrontLayer per ricostruire il layer, adoperando la definizione aggiornata di superbitmap.

DisposeLayerInfo

Sintassi di chiamata della funzione

DisposeLayerInfo (layer_Info)
AØ

Scopo della funzione

Questa funzione libera la memoria precedentemente assegnata a una struttura Layer_Info attraverso la funzione NewLayerInfo.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DisposeLayerInfo, FattenLayerInfo, InitLayers, LockLayerInfo, NewLayerInfo, ThinLayerInfo e UnlockLayerInfo. Si vedano anche le spiegazioni relative a queste funzioni.

La funzione NewLayerInfo consente di allocare memoria per una nuova struttura Layer_Info, crearla e impostarla. La struttura Layer_Info riunisce layer correlati in un'unica lista, così che varie operazioni possano essere svolte su tutti i layer agendo soltanto sulla struttura Layer_Info che definisce la lista. Quando si prevede di non dover utilizzare più alcun layer della struttura, questa può essere rimossa dal sistema. La funzione DisposeLayerInfo consente di liberare la memoria precedentemente assegnata alla struttura Layer_Info dalla funzione NewLayerInfo.

EndUpdate

Sintassi di chiamata della funzione

EndUpdate (layer, flag)
AØ DØ

Scopo della funzione

Questa funzione, che dev'essere eseguita dopo ogni chiamata a BeginUpdate, ripristina l'indirizzo della lista dei danni nel parametro DamageList della struttura Layer. I task chiamano BeginUpdate per chiedere al sistema di predisporre il layer affinché le funzioni di disegno operino soltanto nelle parti danneggiate tornate alla luce. Questa procedura è necessaria con i layer a refresh semplice, per i quali il refresh delle parti tornate alla luce dev'essere svolto direttamente dal task. Quando il refresh è terminato, il task deve chiamare la funzione EndUpdate per ripristinare le normali condizioni di funzionamento del layer. In particolare, questo ripristina l'originale regione di delimitazione associata al layer. Da quel momento in poi, il disegno sarà ristretto alle aree comprese nella regione di delimitazione.

Argomenti della funzione

layer — Indirizzo della struttura Layer che rappresenta il layer a refresh semplice.

flag — In questo argomento, viene indicata la costante TRUE se l'aggiornamento è stato completato (la lista dei danni viene cancellata), oppure la costante FALSE se l'aggiornamento non è stato completato (la lista dei danni viene conservata).

Discussione

Le funzioni BeginUpdate ed EndUpdate consentono di controllare l'aggiornamento del disegno di un layer a refresh semplice. La funzione BeginUpdate permette a un task di cambiare (o ricostruire) i contenuti grafici

nelle porzioni nascoste di un layer a refresh semplice, impiegando le funzioni di disegno della libreria Graphics. La funzione EndUpdate serve per dichiarare la conclusione del processo di aggiornamento delle parti danneggiate di un layer a refresh semplice.

La funzione EndUpdate viene usata soltanto per i layer a refresh semplice. Scambia i puntatori alle struttura ClipRect e DamageList impiegati nell'aggiornamento delle sezioni danneggiate del layer. Si veda anche la spiegazione relativa alla funzione BeginUpdate.

La funzione EndUpdate dovrebbe essere chiamata dopo che il task grafico ha ridisegnato completamente le parti nascoste di un layer a refresh semplice. Si dovrebbe indicare il valore zero nell'argomento flag se si sta effettuando un aggiornamento parziale delle aree oscurate. Si possono inoltre impiegare le funzioni di gestione delle regioni della libreria Graphics per definire una regione di delimitazione più piccola rispetto all'area totale definita dalla struttura DamageList. Ciò riduce ulteriormente l'area effettiva di disegno conducendo perciò a un aggiornamento parziale. Questa scelta è opportuna soltanto se si desidera accelerare il processo di refresh, e non ha alcuna importanza l'aspetto intermedio che vengono ad assumere alcune sezioni del layer a refresh semplice.

FattenLayerInfo

Sintassi di chiamata della funzione

FattenLayerInfo (layer_Info)
A0

Scopo della funzione

Questa funzione alloca memoria addizionale per la struttura Layer_Info quando viene usata la funzione InitLayers, non più attuale con le versioni del software sistema superiori alla 1.0.

Argomenti della funzione

layer_Info Indirizzo della struttura Layer_Info.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DisposeLayerInfo, FattenLayerInfo, InitLayers, LockLayerInfo, NewLayerInfo, ThinLayerInfo e UnlockLayerInfo.

Le funzioni FattenLayerInfo e ThinLayerInfo sono predisposte per la compatibilità verso il basso con la versione 1.0 del software sistema. Esse consentono di adattare i programmi 1.0 che impiegano l'obsoleta funzione InitLayers alle successive versioni del software sistema. Queste due funzioni allocano e liberano memoria aggiuntiva necessaria per le informazioni addizionali che la struttura Layer_Info contiene a partire dalla versione 1.1.

Se si sta eseguendo la compilazione in una versione del software sistema superiore alla 1.0 e si vuole utilizzare la funzione InitLayers per inizializzare la struttura Layer_Info, si deve far seguire la chiamata a InitLayers dalla chiamata alla funzione FattenLayerInfo. Ciò allocherà la memoria aggiuntiva necessaria alla nuova versione della struttura Layer_Info. In alternativa è possibile usare le funzioni NewLayerInfo e DisposeLayerInfo, evitando in questo modo di impiegare le funzioni AllocMem, InitLayers, FattenLayerInfo, ThinLayerInfo e FreeMem.

InitLayers

Sintassi di chiamata della funzione

InitLayers (layer_Info)
AØ

Scopo della funzione

Questa funzione inizializza la versione 1.0 della struttura Layer_Info, ed è quindi una funzione obsoleta, mantenuta unicamente per garantire la compatibilità verso il basso del software sistema.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DisposeLayerInfo, FattenLayerInfo, InitLayers, LockLayerInfo, NewLayerInfo, ThinLayerInfo e UnlockLayerInfo. InitLayers e NewLayerInfo consentono entrambe d'impostare una nuova struttura Layer_Info. NewLayerInfo alloca e inizializza la versione aggiornata della struttura Layer_Info, mentre la funzione InitLayers inizializza la struttura Layer_Info versione 1.0 che il task deve aver già allocato per proprio conto.

Per usare InitLayers con la versione software dell'Amiga superiore alla 1.0, si deve far seguire la chiamata a InitLayers da una chiamata a FattenLayerInfo. Ciò alloca la memoria addizionale necessaria per la versione attuale della struttura Layer_Info. Quando si vuole liberare tutta la memoria assegnata alla struttura Layer_Info, si deve chiamare la funzione ThinLayerInfo (che provvede a liberare la memoria aggiuntiva allocata attraverso la funzione FattenLayerInfo) e poi la funzione FreeMem della libreria Exec.

InstallClipRegion

Sintassi di chiamata della funzione

```
oldClipRegion = InstallClipRegion (layer, region)
D0                                 A0     A1
```

Scopo della funzione

Questa funzione installa una regione di delimitazione in un layer. Tutti i disegni che il task esegue nel layer vengono limitati all'interno della regione. Se la chiamata ha successo, InstallClipRegion restituisce l'indirizzo della precedente struttura Region installata nel layer; se nessuna regione era stata mai installata prima nel layer, viene restituito il valore 0.

Argomenti della funzione

layer	Indirizzo della struttura Layer che definisce il layer al quale si vuole associare la regione di delimitazione.

region Indirizzo della struttura Region che rappresenta la regione di delimitazione da associare al layer.

Discussione

InstallClipRegion è la sola funzione che riguarda i rapporti tra layer e regioni. Consente di modificare l'area di una bitmap di layer che servirà come area di disegno, assegnando una nuova regione di delimitazione al layer.

Se il sistema eccede le capacità di memoria mentre è in corso il calcolo dei rettangoli di delimitazione richiesti dalla combinazione layer-regione, nel parametro flags della struttura Layer viene impostato il flag LAYERS_CLIPRECTS_LOST. In tal caso la lista dei rettangoli di delimitazione che rappresenta la nuova combinazione regione-layer può andare perduta. Tuttavia, non appena la memoria ridiventa sufficiente e la libreria Layers viene nuovamente chiamata, la lista dei rettangoli di delimitazione viene ricostruita.

È importante notare che, se si era precedentemente installata nel layer una regione di delimitazione non nulla, prima di chiamare la funzione DeleteLayer della libreria Graphics o la funzione CloseWindow della libreria Intuition, si deve effettuare la seguente chiamata:

```
InstallClipRegion (layer, NULL)
```

LockLayer

Sintassi di chiamata della funzione

```
LockLayer (dummy, layer)
          A0      A1
```

Scopo della funzione

Questa funzione permette a un task di bloccare un layer, per essere l'unico in grado di alterare la bitmap o i rettangoli di delimitazione associati a quel layer. Se un altro task sta disegnando nel layer, la funzione LockLayer manda il task in questione in attesa fino a quando il layer non viene sbloccato.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer.

Discussione

Ci sono sette funzioni che consentono di gestire singoli layer in un gruppo di layer controllati da una specifica struttura Layer_Info: DeleteLayer, LockLayer, MoveLayer, ScrollLayer, SizeLayer, UnlockLayer e WhichLayer. Si vedano anche le spiegazioni relative a queste funzioni.

Se il task intende effettuare alcuni cambiamenti in un particolare layer, come spostarlo o ridimensionarlo, deve inibire agli altri task l'alterazione del layer. È questo lo scopo della funzione LockLayer; LockLayer impedisce agli altri task di cambiare il disegno o la definizione dei rettangoli di delimitazione di una bitmap di layer. Tale condizione di blocco continua fino a che il task non emette una chiamata alla funzione UnlockLayer. Se si sta bloccando un solo layer con un'unica chiamata a LockLayer, non è necessario far precedere la chiamata da una chiamata alla funzione LockLayerInfo. Anzi, a questo scopo è possibile impiegare anche la più rapida funzione LockLayerRom della libreria Graphics.

Lo scambio di controllo fra i task non viene influenzato dall'esecuzione della funzione LockLayer, dal momento che il controllo d'accesso ai layer viene gestito attraverso i semafori di segnalazione. Tuttavia, un nuovo task che prenda il controllo della macchina e tenti di disegnare nel layer bloccato viene automaticamente sospeso, fino a quando il task che ha originato il blocco non rende nuovamente disponibile il layer.

Se si vuole bloccare contemporaneamente più di un layer, ci sono due modi di procedere.

Primo, si possono effettuare diverse chiamate a LockLayer. In tal caso, però, ciascuna chiamata dev'essere preceduta da una chiamata alla funzione LockLayerInfo e seguita da una chiamata alla funzione UnlockLayerInfo; ciò per evitare il blocco del sistema nell'eventualità che più task cerchino di accedere contemporaneamente agli stessi layer.

Secondo, si possono simultaneamente bloccare tutti i layer rappresentati da una struttura Layer_Info, chiamando la funzione LockLayers. Naturalmente tale chiamata va fatta seguire da una corrispondente chiamata alla funzione UnlockLayers quando i layer possono essere sbloccati. Seguendo tale procedura non è necessario chiamare LockLayerInfo e UnlockLayerInfo.

LockLayerInfo

Sintassi di chiamata della funzione

LockLayerInfo (layer_Info)
AØ

Scopo della funzione

Questa funzione blocca una struttura Layer_Info, cosicché solo il task chiamante possa modificarla. LockLayerInfo è l'inverso della funzione UnlockLayerInfo.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info da bloccare.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DeleteLayer, DisposeLayerInfo, FattenLayerInfo, LockLayerInfo, NewLayerInfo, ThinLayerInfo e UnlockLayerInfo. Si vedano anche le spiegazioni relative a queste funzioni.

La funzione LockLayerInfo consente di bloccare i dati in una struttura Layer_Info, in modo che nessun altro task oltre a quello in esecuzione possa effettuarvi cambiamenti. La funzione UnlockLayerInfo libera la struttura Layer_Info precedentemente bloccata, così che gli altri task possano nuovamente utilizzarla.

Se un task si è procurato l'accesso esclusivo a una struttura Layer_Info con una chiamata alla funzione LockLayerInfo, e non ha ancora provveduto a sbloccarla, gli altri task che tentassero a loro volta di bloccare la struttura verrebbero sospesi.

Nella versione 1.3 del sistema tutte le routine della libreria Layers eseguono una chiamata alla funzione LockLayerInfo al loro ingresso e una alla funzione UnlockLayerInfo al termine. Una chiamata esplicita alla coppia di funzioni LockLayerInfo e UnlockLayerInfo è quindi necessaria soltanto nel caso in cui il programmatore intenda modificare i layer in una maniera non prevista da questa routine.

LockLayers

Sintassi di chiamata della funzione

LockLayers (layer_Info)
AØ

Scopo della funzione

Questa funzione blocca tutti i layer associati all'indicata struttura Layer_Info, cosicché soltanto il task in esecuzione possa alterare le loro bitmap o i loro rettangoli di delimitazione. I layer correlati ad altre strutture Layer_Info non vengono ovviamente bloccati da questa chiamata.

Argomenti della funzione

layer_Info Indirizzo della struttura Layer_Info.

Discussione

Ci sono due funzioni che permettono di gestire simultaneamente i layer associati a una struttura Layer_Info: LockLayers e UnlockLayers. Si possono usarle per bloccare un gruppo di layer correlati, in modo che non subiscano cambiamenti da parte di altri task. Quando non c'è più necessità di accedere ai layer in modo esclusivo, si deve chiamare la funzione UnlockLayers per liberarli tutti.

Dal momento che la struttura Layer_Info concatena un gruppo di strutture Layer, la funzione LockLayers può riferirsi a essa per bloccare in una volta sola tutti i relativi layer. Il miglior esempio dell'azione di questa funzione si può osservare quando Intuition attiva il contorno rettangolare arancione che rappresenta i confini di una finestra durante uno spostamento o un ridimensionamento. Nella fase di spostamento della finestra tutti i layer sono bloccati grazie a una chiamata alla funzione LockLayers. Quando poi l'utente deposita la cornice arancione in una nuova posizione, Intuition chiama la funzione UnlockLayers per liberare tutti i layer che appaiono sullo schermo.

MoveLayer

Sintassi di chiamata della funzione

MoveLayer (dummy, layer, delta-x, delta-y)
AØ A1 DØ D1

Scopo della funzione

Questa funzione sposta un layer non backdrop in una nuova posizione nella bitmap di layer. Se tale operazione riporta in vista settori di layer a refresh semplice, MoveLayer raccoglie le aree danneggiate e imposta i flag REFRESH nel parametro Flags delle relative strutture Layer, in modo che i task che li controllano vengano avvisati della necessità di procedere al refresh dei rispettivi layer.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer che rappresenta un layer non backdrop.
delta-x	Spostamento (in pixel) della bitmap di layer in direzione orizzontale.
delta-y	Spostamento (in pixel) della bitmap di layer in direzione verticale.

Discussione

Ci sono sette funzioni che consentono di gestire singoli layer in un gruppo di layer controllati da una specifica struttura Layer_Info: DeleteLayer, LockLayer, MoveLayer, ScrollLayer, SizeLayer, UnlockLayer e WhichLayer. Si vedano anche le spiegazioni relative a queste funzioni.

Si utilizza MoveLayer ogni volta che si vuole spostare un layer sullo schermo. Questa funzione agisce sulla lista dei layer gestita dalla struttura Layer_Info, e quindi la blocca automaticamente nel corso della sua esecuzione.

MoveLayerInFrontOf

Sintassi di chiamata della funzione

```
success = MoveLayerInFrontOf (layerToMove, targetLayer)
DØ                            AØ           A1
```

Scopo della funzione

Questa funzione sposta un layer non backdrop in una nuova posizione nell'ordine di stack per i layer. Se il layer da spostare è a refresh semplice, MoveLayerInFrontOf raccoglie automaticamente le eventuali aree danneggiate in una lista dei danni e imposta il flag REFRESH nel parametro Flags del layer danneggiato. La funzione restituisce un valore booleano. Se ha successo, il valore è TRUE; in caso contrario, FALSE.

Argomenti della funzione

layerToMove — Indirizzo della struttura Layer che rappresenta il layer non backdrop da spostare.

targetLayer — Indirizzo della struttura Layer che rappresenta il layer non backdrop di fronte al quale dev'essere collocato il layer considerato.

Discussione

Ci sono tre funzioni che consentono di riordinare i layer sullo schermo dell'Amiga: BehindLayer, UpfrontLayer e MoveLayerInFrontOf. Questa possibilità è particolarmente utile in un ambiente a finestre come Intuition e in un programma applicativo che voglia emulare la sua tipica architettura a schermi e finestre.

La funzione MoveLayerInFrontOf consente di spostare un layer davanti a un altro. L'operazione permette un preciso controllo sull'ordine di stack per i layer nel sistema. Si noti che si può adoperare questa funzione anche per spostare un layer backdrop, azzerando il flag BACKDROP nel parametro Flags della relativa struttura Layer.

Per la descrizione della struttura Layer si veda la spiegazione della funzione CreateBehindLayer.

NewLayerInfo

Sintassi di chiamata della funzione

```
layerInfo = NewLayerInfo ()
DØ
```

Scopo della funzione

Questa funzione alloca memoria per una struttura Layer_Info e la inizializza. Ogni volta che si crea un layer, occorre associarlo a una struttura Layer_Info. Fatta quest'operazione, le altre funzioni della libreria Layers possono operare sulla lista dei layer associata alla struttura Layer_Info. NewLayerInfo ha anche l'effetto di sbloccare i layer, rendendo possibile intervenire sulle loro definizioni di bitmap.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DisposeLayerInfo, FattenLayerInfo, InitLayers, LockLayerInfo, NewLayerInfo, ThinLayerInfo e UnlockLayerInfo. Si vedano anche le spiegazioni relative a queste funzioni.

La struttura Layer_Info

La struttura Layer_Info è definita come segue:

```
struct Layer_Info {
        struct Layer *top_layer;
        struct Layer *check_lp;
        struct Layer *obs;
        struct MinList FreeClipRects;
        struct SignalSemaphore Lock;
        struct List gs_Head;
        LONG longreserved;
        UWORD Flags;
        BYTE fatten_count;
        BYTE LockLayersCount;
        UWORD LayerInfo_extra_size;
        WORD *blitbuff;
        struct LayerInfo_extra *LayerInfo_extra;
};
```

Ecco una breve spiegazione di alcuni dei parametri contenuti nella struttura Layer_Info:

- il parametro top_layer punta al layer di superficie tra i layer rappresentati dalla struttura.
- I parametri check_lp e obs sono riservati al sistema.
- Il parametro Lock è una sotto-struttura SignalSemaphore che rappresenta il semaforo di segnalazione impiegato dal sistema per regolare l'accesso alla struttura.

I rimanenti parametri sono per lo più riservati al sistema.

ScrollLayer

Sintassi di chiamata della funzione

```
ScrollLayer (dummy, layer, delta-x, delta-y)
             A0     A1     D0       D1
```

Scopo della funzione

Questa funzione effettua lo scroll di un layer superbitmap. Copia i bit tra la superbitmap e la bitmap di layer sullo schermo (che costituisce la parte visibile del layer) per cambiare la parte di layer che viene presentata.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer per un layer superbitmap.
delta-x	Spostamento della bitmap di layer (in pixel) in direzione orizzontale.
delta-y	Spostamento della bitmap di layer (in pixel) in direzione verticale.

Discussione

Ci sono sette funzioni che consentono di gestire singoli layer in un gruppo di layer controllati da una specifica struttura Layer_Info: DeleteLayer, LockLayer, MoveLayer, ScrollLayer, SizeLayer, UnlockLayer e WhichLayer. Si vedano anche le spiegazioni relative a queste funzioni.

La funzione ScrollLayer lavora in genere con i layer superbitmap. Si ricordi che un layer di questo tipo possiede un'unica bitmap di salvataggio. Tale bitmap, detta superbitmap del layer, conserva lo stato sia dei pixel non visibili che di quelli visibili all'interno del layer. La superbitmap può essere (e generalmente lo è) più grande della bitmap di layer su schermo.

La funzione ScrollLayer cambia la parte di superbitmap mostrata sullo schermo. In generale ci sono due modi per cambiare l'aspetto di un layer sullo schermo video. Il primo modo consiste nell'alterare la bitmap di layer attraverso le funzioni di disegno della libreria Graphics. L'altro consiste nell'usare una bitmap più grande del layer (una superbitmap) e presentare in ogni istante soltanto una sezione limitata della superbitmap.

Il principale vantaggio di un layer superbitmap è la facilità con cui si possono fare scorrere le informazioni del layer sullo schermo. Il principale svantaggio consiste nella necessità di predisporre un'area aggiuntiva di memoria per contenere i bitplane della superbitmap.

Per esempio, un layer da 320 x 256 pixel a 32 colori, richiede 10240 byte per bitplane, cioè 51200 byte per la sola bitmap visibile. Se si tratta di un layer

superbitmap, la parte esterna allo schermo richiede un'ulteriore quantità di memoria che va a sommarsi ai 51200 byte necessari per la bitmap visibile.

La funzione ScrollLayer può essere utilizzata con ogni tipo di layer al fine di simulare un effetto simile a quello ottenibile da Intuition tramite le finestre GIMMEZEROZERO, ma senza l'ingombro di un layer aggiuntivo (si veda anche il capitolo 6).

Ad esempio, usando 5 come delta-x e 10 come delta-y, il punto (0,0) verrà tracciato in realtà nella posizione (5,10).

SizeLayer

Sintassi di chiamata della funzione

SizeLayer (dummy, layer, delta-x, delta-y)
A0 A1 D0 D1

Scopo della funzione

Questa funzione aumenta o diminuisce le dimensioni di una bitmap di layer.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer relativa a un layer non backdrop.
delta-x	Aumento della bitmap di layer (in pixel) in direzione orizzontale.
delta-y	Aumento della bitmap di layer (in pixel) in direzione verticale.

Discussione

Ci sono sette funzioni che consentono di gestire singoli layer in un gruppo di layer controllati da una struttura Layer_Info: DeleteLayer, LockLayer, MoveLayer, ScrollLayer, SizeLayer, UnlockLayer e WhichLayer. Si vedano anche le spiegazioni relative a queste funzioni.

La funzione SizeLayer cambia le dimensioni di un layer modificando le coordinate dell'angolo inferiore destro della bitmap di layer, e lasciando inalterate le coordinate dell'angolo superiore sinistro. Se le coordinate dell'angolo inferiore destro aumentano, la bitmap di layer viene ingrandita, in caso contrario viene rimpicciolita.

Se il layer è di tipo superbitmap, la funzione SizeLayer copia i pixel dalla superbitmap, seguendo le richieste di variazione di misura nella bitmap di layer. La funzione raccoglie inoltre le damage list dei layer a refresh semplice che restino eventualmente danneggiati dall'operazione di ridimensionamento. Queste liste possono poi essere usate dai task per ridisegnare le sezioni a seconda delle necessità. Si noti che il sistema blocca automaticamente la struttura Layer_Info mentre sta ridimensionando un layer.

SwapBitsRastPortClipRect

Sintassi di chiamata della funzione

```
SwapBitsRastPortClipRect (rastPort, clipRect)
                          A0        A1
```

Scopo della funzione

Questa funzione scambia i bit tra una bitmap di layer (la bitmap relativa alla struttura BitMap puntata dalla struttura RastPort indicata come primo argomento) e una bitmap fuori schermo (la bitmap relativa alla struttura BitMap puntata dalla struttura ClipRect indicata come secondo argomento). Questa procedura evita la necessità di creare un nuovo layer per la bitmap fuori schermo e conduce a un aggiornamento molto rapido dello schermo video.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort.
clipRect	Indirizzo di una speciale struttura ClipRect progettata appositamente per questa funzione.

Discussione

La funzione SwapBitsRastPortClipRect permette di scambiare un insieme di pixel tra una bitmap che appare sullo schermo e un insieme di bitmap non visibili. Questo meccanismo fornisce un modo molto rapido per far apparire sulla bitmap di schermo un disegno che si sovrapponga temporaneamente a quelli già realizzati senza dover impiegare un layer.

Si ricorre alla funzione SwapBitsRastPortClipRect quando si vogliono inserire informazioni aggiuntive sullo schermo video conservando quelle che vengono coperte, e si vuole evitare di creare un nuovo layer. Può anche essere utilizzata per rendere più veloci le operazioni di presentazione video.

È questa funzione che genera le velocissime operazioni di menu che gli utenti di Intuition ben conoscono.

Vediamo come opera la funzione. Supponiamo di avere diverse finestre aperte in uno schermo di Intuition; ciascuna di esse costituisce parte di un layer. Si supponga ora di voler portare sullo schermo un menu.

Una prima possibilità è quella di creare un layer impiegando la funzione CreateUpfrontLayer, e di disegnarvi il menu adoperando le funzioni di disegno della libreria Graphics e quelle di Intuition per la creazione e la gestione dei menu. Naturalmente dev'essere stata predisposta abbastanza memoria per la nuova bitmap di layer e per le associate strutture Layer e Layer_Info. Si devono anche creare tutti i rettangoli di delimitazione per le finestre che verranno nascoste quando il nuovo menu sarà visualizzato.

Una seconda possibilità è quella di usare la funzione SwapBitsRastPortClipRect per visualizzare direttamente il menu sullo schermo video. Innanzitutto si disegna il menu in un'area bitmap fuori schermo. Si bloccano poi tutti i layer tramite la funzione LockLayers, per impedire agli altri task di modificarli e di disegnare quindi sul menu.

Si scambiano infine i bit della bitmap su schermo (la bitmap relativa alla struttura BitMap indicata nella struttura RastPort) con i bit della bitmap fuori schermo (la bitmap relativa alla struttura BitMap indicata in una speciale struttura ClipRect), impiegando la funzione SwapBitsRastPortClipRect. Fatto ciò, sullo schermo video appare il menu. Per chiuderlo, si usa ancora la funzione SwapBitsRastPortClipRect per scambiare nuovamente le bitmap e ricostruire la presentazione originaria dello schermo video.

Si noti che questa procedura di disegno è molto più rapida della precedente. Inoltre lascia inalterate le normali strutture ClipRect e la maggior parte delle altre strutture correlate alla presentazione a finestre, riducendo le

responsabilità del task.

Si noti infine che tutti i layer di schermo devono essere bloccati mentre il menu viene mostrato: qualsiasi task stia disegnando nei layer viene quindi posto temporaneamente in attesa mentre il menu viene presentato, rallentando di conseguenza le operazioni dei task di disegno. Con il primo metodo, invece, dal momento che si crea il menu in una bitmap di layer separata non c'è bisogno di bloccare alcun task mentre il menu viene presentato.

ThinLayerInfo

Sintassi di chiamata della funzione

ThinLayerInfo (layer_Info)
AØ

Scopo della funzione

Questa funzione libera la memoria aggiuntiva allocata per la struttura Layer_Info dalla funzione FattenLayerInfo. Consente alla funzione InitLayers di essere usata nei programmi nati per la versione 1.0 del sistema. In pratica, la funzione Layer_Info prevista dalla versione 1.0 del software sistema occupava meno memoria di quanta ne occupi l'attuale struttura Layer_Info, introdotta con la versione 1.1 del software sistema. Pertanto, se un programmatore desidera aggiornare un programma sorgente nato per la versione 1.0, che evidentemente usava la funzione InitLayers, deve aggiungere le chiamate di funzione FattenLayerInfo e ThinLayerInfo, che rispettivamente provvedono ad allocare e liberare la memoria aggiuntiva richiesta dall'attuale struttura Layer_Info.

Argomenti della funzione

layer_Info Indirizzo della struttura Layer_Info.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DisposeLayerInfo, FattenLayerInfo, InitLayers, LockLayerInfo, NewLayerInfo, ThinLayerIn-

fo e UnlockLayerInfo. La struttura Layer_Info è presentata insieme alla spiegazione della funzione NewLayerInfo.

Le funzioni FattenLayerInfo e ThinLayerInfo sono predisposte per la compatibilità verso il basso con la versione del software sistema 1.0. Esse consentono di adattare i programmi 1.0 che impiegano la funzione InitLayers alla versione 1.1 della struttura Layer_Info (che non è più cambiata nelle versioni successive del software sistema ed è quindi quella attuale). Queste due funzioni assegnano e liberano la memoria necessaria per le informazioni addizionali destinate alla struttura Layer_Info quando la funzione InitLayers viene adoperata in un programma nato per la versione 1.0 del software sistema.

Se si adopera la funzione InitLayers per inizializzare la struttura Layer_Info, si deve far seguire una chiamata a FattenLayerInfo per allocare la memoria aggiuntiva usata dalla struttura Layer_Info versione 1.1. Quando si vuole liberare la memoria aggiuntiva allocata dalla funzione FattenLayerInfo, si chiama la funzione ThinLayerInfo, seguita poi dalla chiamata alla funzione FreeMem per liberare la memoria occupata dalla struttura Layer_Info versione 1.0, memoria che è stata originariamente allocata attraverso la funzione AllocMem. Le chiamate ad AllocMem e FreeMem sono necessarie perché la funzione InitLayers inizializza ma non alloca la memoria destinata alla struttura Layer_Info.

UnlockLayer

Sintassi di chiamata della funzione

UnlockLayer (layer)
A0

Scopo della funzione

Questa funzione sblocca un layer e permette agli altri task di modificare la sua bitmap. La funzione UnlockLayer va chiamata quando il task che detiene il controllo esclusivo di un layer (ottenuto con una chiamata a LockLayer) ha finito di disegnare o di effettuare modifiche nei rettangoli di delimitazione associati alla relativa bitmap.

Argomenti della funzione

layer	Indirizzo della struttura Layer.

Discussione

Ci sono sette funzioni che consentono di gestire singoli layer in un gruppo di layer controllati da una struttura Layer_Info: DeleteLayer, LockLayer, MoveLayer, ScrollLayer, SizeLayer, UnlockLayer e WhichLayer. Si vedano anche le spiegazioni relative a queste funzioni.

Si ricordi che la funzione LockLayer blocca un layer per consentire a un solo task di effettuarvi modifiche. Quando si vuole liberare il layer, per consentire ad altri task di modificarne la bitmap o la definizione dei rettangoli di delimitazione, si chiama la funzione UnlockLayer.

Dopo la chiamata a UnlockLayer, che bilancia una precedente chiamata alla funzione LockLayer, il primo task messo in attesa perché aveva tentato di disegnare nel layer in questione (oppure di modificarne la definizione dei rettangoli di delimitazione) riottiene il controllo.

UnlockLayerInfo

Sintassi di chiamata della funzione

UnlockLayerInfo (layer_Info)
AØ

Scopo della funzione

Questa funzione sblocca una struttura Layer_Info precedentemente bloccata con una chiamata alla funzione LockLayerInfo. Quando UnlockLayerInfo restituisce il controllo, gli altri task possono modificare la struttura Layer_Info.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.

Discussione

Ci sono sette funzioni che riguardano la struttura Layer_Info: DisposeLayerInfo, FattenLayerInfo, InitLayers, LockLayerInfo, NewLayerInfo, ThinLayerInfo e UnlockLayerInfo. Si vedano anche le spiegazioni relative a queste funzioni.

Ogni volta che si chiama LockLayerInfo, dovrebbe seguire prima o poi una chiamata alla funzione UnlockLayerInfo. Tutte le funzioni della libreria Layers chiamano automaticamente la funzione LockLayerInfo. Provvedono anche a chiamare UnlockLayerInfo appena prima di restituire il controllo. Per la definizione della struttura Layer_Info si veda la spiegazione della funzione NewLayerInfo.

Ogni volta che si vuole modificare una struttura Layer in modi non previsti dalle funzioni della libreria Layers si dovrebbe far precedere e seguire ai cambiamenti la coppia di chiamate alle funzioni LockLayerInfo e UnlockLayerInfo.

UnlockLayers

Sintassi di chiamata della funzione

UnlockLayers (layer_Info)
AØ

Scopo della funzione

Questa funzione sblocca, per prima cosa, tutti i layer associati alla struttura Layer_Info indicata. Ciò consente agli altri task di modificare le bitmap di layer e le definizioni dei rettangoli di delimitazione relative ai layer sbloccati. UnlockLayers sblocca poi la stessa struttura Layer_Info.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.

Discussione

Le funzioni LockLayers e UnlockLayers consentono di gestire simultaneamente tutti i layer associati a una struttura Layer_Info. Si può chiamare la funzione LockLayers per bloccare un gruppo di layer correlati, in modo che soltanto il task che emette la chiamata possa effettuare cambiamenti. Si chiama poi la funzione UnlockLayers per sbloccare tutti i layer contemporaneamente.

Ogni chiamata alla funzione LockLayers dovrebbe essere seguita da una corrispondente chiamata alla funzione UnlockLayers. Dopo l'esecuzione della funzione UnlockLayers, i layer tornano accessibili a tutti i task.

UpfrontLayer

Sintassi di chiamata della funzione

```
success = UpfrontLayer (dummy, layer)
D0                      A0     A1
```

Scopo della funzione

Questa funzione dispone il layer indicato davanti a tutti gli altri. UpfrontLayer restituisce un valore booleano: se ha successo TRUE, altrimenti FALSE.

Argomenti della funzione

dummy	Inutilizzato.
layer	Indirizzo della struttura Layer.

Discussione

Ci sono tre funzioni che consentono di riordinare i layer sullo schermo video dell'Amiga: BehindLayer, UpfrontLayer e MoveLayerInFrontOf. Questa possibilità è particolarmente utile in un ambiente a finestre come quello di Intuition e in un programma applicativo che voglia emulare gli schermi e le

finestre tipiche di Intuition. Ogni volta che si vuole cambiare l'ordine di stack dei layer è necessario ricorrere a una di queste tre funzioni.

Quando un layer viene portato davanti a tutti gli altri, tutte le sue parti precedentemente nascoste diventano visibili. Se il layer è del tipo a refresh avanzato (smart-refresh) o superbitmap, è il sistema che si preoccupa di eseguire il refresh delle parti tornate in luce. Se invece il layer è a refresh semplice, le informazioni che occupavano le parti nascoste sono perdute; per la ricostruzione si devono impiegare le funzioni BeginUpdate e EndUpdate (che riguardano la damage list).

In generale, le funzioni della libreria Layers suddividono l'area di disegno per tutti i tipi di layer in vari rettangoli di delimitazione. Quando il task grafico chiama le funzioni di disegno, esse disegnano soltanto nelle aree definite dai rettangoli di delimitazione.

In particolare, se si aggiorna un layer a refresh avanzato o a superbitmap, tale layer può risultare parzialmente nascosto durante l'aggiornamento. In questo caso, comunque, la libreria Layers controlla che le funzioni grafiche operino anche sulle parti nascoste.

Impiegando i rettangoli di delimitazione, le funzioni grafiche della libreria Graphics possono individuare com'è suddivisa l'area di disegno e dove devono aver luogo gli aggiornamenti. Una volta che si sia definito il tipo di layer in cui disegnare, il sistema gestisce automaticamente tali operazioni. Ricordiamo che la situazione per un layer a refresh semplice è piuttosto diversa (si veda, a questo proposito, la spiegazione della funzione BeginUpdate).

Se il layer indicato dalla funzione UpfrontLayer è a refresh semplice, la funzione raccoglie la sua damage list (un insieme di rettangoli di delimitazione) e imposta i bit nel parametro Flags della struttura Layer del layer coinvolto.

Se il layer indicato è di tipo backdrop, la funzione può disporlo davanti a tutti gli altri soltanto azzerando il flag LAYERBACKDROP nel parametro Flags della struttura Layer con la seguente istruzione:

```
layer->Flags &= ~LAYERBACKDROP;
```

Mentre per riattivare il flag si deve impiegare la seguente istruzione:

```
layer->Flags |= LAYERBACKDROP;
```

L'insieme dei rettangoli di delimitazione che costituisce il gruppo di aree del layer in cui effettuare il disegno viene posto in una lista di rettangoli di delimitazione. La lista dei rettangoli di delimitazione è rappresentata e gestita dalla struttura ClipRect.

La struttura ClipRect

Ciascuna struttura Layer contiene un puntatore a una struttura ClipRect. Tale struttura viene impiegata per gestire un insieme di rettangoli di delimitazione, da utilizzare per gli aggiornamenti grafici del layer. La struttura ClipRect è definita come segue:

```
struct ClipRect {
    struct ClipRect *Next;
    struct ClipRect *Prev;
    struct Layer *lobs;
    struct Bitmap *BitMap;
    struct Rectangle bounds;
    struct ClipRect *_p1, *_p2;
    LONG reserved;
    LONG Flags;
};
```

Il parametro Next punta alla successiva struttura ClipRect e il parametro Prev punta alla precedente. Tali strutture sono parte di una lista concatenata che definisce le aree in cui sono destinati a verificarsi gli aggiornamenti grafici relativi a un certo layer. Il parametro BitMap punta alla struttura BitMap alla quale questa struttura ClipRect è associata. Il parametro bounds è il nome di una struttura Rectangle che definisce i confini del rettangolo di delimitazione associato alla struttura ClipRect. I rimanenti parametri sono riservati al sistema.

Per la descrizione della struttura Layer si veda la spiegazione della funzione CreateBehindLayer. Per la descrizione della struttura Layer_Info si veda la spiegazione della funzione NewLayerInfo.

WhichLayer

Sintassi di chiamata della funzione

```
layer = WhichLayer (layer_Info,     x,     y)
D0                  A1              D0     D1
```

Scopo della funzione

Questa funzione identifica la bitmap di layer nella quale è contenuto il pixel di coordinate x,y. WhichLayer esamina i layer uno dopo l'altro, partendo dal layer superiore di un insieme correlato. Se identifica il layer che contiene il pixel, la funzione restituisce l'indirizzo della corrispondente struttura Layer. Se il layer invece non viene rintracciato, la funzione restituisce il valore 0.

Argomenti della funzione

layer_Info	Indirizzo della struttura Layer_Info.
x	Coordinata x del pixel cercato.
y	Coordinata y del pixel cercato.

Discussione

Ci sono sette funzioni che consentono di gestire singoli layer in un gruppo di layer controllati da una struttura Layer_Info: DeleteLayer, LockLayer, MoveLayer, ScrollLayer, SizeLayer, UnlockLayer e WhichLayer. Si vedano anche le spiegazioni relative a queste funzioni.

Se l'area complessiva dello schermo è occupata da diversi layer, può capitare di dover identificare il layer nella cui bitmap è definito un particolare pixel dello schermo.

Intuition si comporta proprio in questo modo per sapere sempre dove si trova il mouse. Quando il mouse viene spostato in una particolare posizione dello schermo e viene premuto il pulsante sinistro, Intuition deve localizzare la finestra sulla quale si trova il puntatore, e per farlo chiama la funzione WhichLayer indicando le coordinate del puntatore sullo schermo. La funzione restituisce l'indirizzo della struttura Layer che definisce lo stato del pixel localizzato da quelle coordinate, e Intuition rende attiva la finestra corrispondente.

Le funzioni della libreria Intuition

Introduzione

Intuition è l'interfaccia utente dell'Amiga. È costituita da una libreria contenente 60 funzioni già predisposte per progettare con facilità programmi applicativi che interagiscano con l'utente attraverso finestre, menu, schermi e gadget. Le funzioni di Intuition sono studiate per rendere quanto più semplice possibile la manipolazione di finestre, schermi, requester (box di dialogo), gadget e menu.

I requester e i gadget sono il principale metodo di comunicazione tra il programma e l'utente. Ci sono due tipi di gadget: i gadget di programma e i gadget di sistema. I gadget di sistema sono forniti automaticamente nelle finestre e negli schermi, sempre che s'impostino gli appropriati flag nelle strutture Screen e Window. Si noti che non è possibile usare le funzioni della libreria Intuition per rimuovere i gadget di sistema. I gadget di sistema sono di quattro tipi:

- gadget di dimensionamento, che appare nell'angolo inferiore destro della finestra o dello schermo.
- Gadget di spostamento, che appare lungo la barra titolo dello schermo o della finestra; viene anche definito barra di spostamento.
- Gadget di profondità (che consente cambiare la profondità dello schermo o della finestra). Ve ne sono due nell'angolo superiore destro.
- Gadget di chiusura, che appare nell'angolo superiore sinistro.

A eccezione del gadget di chiusura, il programma non ha bisogno di osservare ed elaborare messaggi provenienti da questi gadget. È possibile "attaccare" questi gadget di sistema alle finestre e agli schermi, e lasciare a Intuition il compito di rispondere alle richieste dell'utente.

I requester sono box di dialogo che appaiono sullo schermo per scambiare informazioni con l'utente. Possono essere presentati in finestre, tanto dal sistema quanto dai programmi. Sono anche disponibili requester che l'utente può richiedere esplicitamente. I requester generano una richiesta che l'utente deve soddisfare prima di proseguire con l'input nella finestra attiva.

Oltre ai gadget e ai requester, è possibile predisporre gli alert (allarmi). Gli alert sono messaggi d'errore che appaiono automaticamente sullo schermo quando si verifica una condizione critica. Appartengono a due classi: gli alert di sistema e gli alert dei programmi applicativi. Un esempio di alert di sistema è il messaggio "out of memory". Si possono progettare alert per i programmi in modo da avvertire l'utente quando si verifica una condizione pericolosa per l'elaborazione. Un alert appare in rosso e nero con un testo all'interno e i bordi lampeggianti.

I programmi possono adoperare due metodi di input/output con le finestre.

L'input ai programmi giunge attraverso il dispositivo Console e attraverso le message port IDCMP (Intuition Direct Communications Message Ports) per la trasmissione diretta di messaggi. L'output del programma può avvenire attraverso il dispositivo Console o direttamente attraverso le funzioni della libreria Graphics.

Gli schermi e le finestre di Intuition

Gli schermi e le finestre sono entrambi rettangolari. Uno schermo è costituito da una viewport e copre l'intera superficie orizzontale del video dell'Amiga. Non è necessario che copra anche l'intera superficie verticale dal momento che verticalmente le viewport non impongono alcuna limitazione.

Le finestre sono costituite da layer, e quindi possono essere disposte ovunque e avere qualunque dimensione entro i limiti dello schermo. Sia le finestre sia gli schermi possono essere sovrapposti e ordinati in profondità impiegando gli appositi gadget.

Gli schermi aperti dal programma possono usare tutti i vari modi video offerti dall'Amiga: bassa risoluzione, alta risoluzione, senza interlace e con interlace, single-playfield, dual-playfield e Hold And Modify. Le combinazioni di colore e le restrizioni che si applicano a questi modi video quando si adoperano le funzioni della libreria Graphics sono valide anche per le funzioni di Intuition.

Ogni programma può aprire uno o più terminali virtuali. Un *terminale virtuale* è semplicemente una finestra su uno schermo; l'utente accede a ciascun terminale virtuale attraverso la corrispondente finestra. L'input e l'output del programma avvengono attraverso le finestre. Un programma può aprire tutte le finestre che gli sono necessarie per soddisfare le sue necessità di input/output.

Il programma può ignorare i movimenti e le modifiche che l'utente effettua sulle finestre, dal momento che è Intuition a gestire automaticamente questo genere di interazioni. Se per esempio l'utente seleziona il gadget di ridimensionamento, è Intuition a seguire i movimenti del puntatore e a ridimensionare di conseguenza la finestra.

Dal punto di vista del generico programma che ha aperto una finestra di I/O per adoperarla come terminale virtuale, la presenza di finestre aperte da altri task è del tutto indifferente. Dal punto di vista del programma, tutti gli input dell'utente effettuati con il mouse e con la tastiera sono diretti soltanto alla sua finestra, anche se in realtà questo avviene solo quando la sua finestra di I/O è attiva.

Un concetto da chiarire è quello di *elemento video contenitore*. Nell'ambiente di Intuition, un elemento video contenitore è una sezione dello schermo in cui è contenuta un'altra porzione della presentazione video. Per esempio, uno schermo è un elemento video contenitore, perché può contenere le finestre. A sua volta, una finestra è un elemento video contenitore perché può far apparire al suo interno un requester.

Ogni elemento aggiunto alla presentazione video, viene necessariamente

collocato in un altro elemento. Ciò vale per la presentazione video, ma soprattutto per le strutture di dati destinate a definire gli elementi video. In particolare, quando un elemento video viene inserito in un elemento contenitore già esistente, la sua posizione è quasi sempre riferita al sistema di coordinate dell'elemento contenitore; in genere, questi sistemi di coordinate collocano il punto 0,0 nell'angolo superiore sinistro dell'elemento contenitore.

Lo spostamento (offset) rispetto all'origine del sistema di coordinate in cui ci si trova è generalmente indicato dai parametri LeftEdge e TopEdge della struttura dell'elemento video. Questi due parametri sono presenti in tutte le strutture Screen, NewScreen, Window, NewWindow, Menu, MenuItem, Requester, Gadget, Border e Image. Perciò, quando si sta per inserire un nuovo elemento video di Intuition, si tenga sempre presente l'elemento contenitore e il suo punto di riferimento per disporre adeguatamente l'elemento video che viene aggiunto.

Gli schermi devono essere necessariamente larghi quanto l'intero schermo video, e disposti orizzontalmente a partire dall'estremità sinistra della view. Verticalmente, invece, il punto di coordinata y più basso dello schermo non può andare al di sopra del limite inferiore del video dell'Amiga. Le finestre, al contrario, possono essere posizionate e e sovrapposte senza alcuna restrizione.

Gli schermi possono essere spostati verticalmente ma non orizzontalmente, mentre le finestre possono essere spostate sullo schermo in tutte le direzioni.

In un dato momento può essere attiva una sola finestra, che viene definita attiva perché è l'unica a ricevere gli input dall'utente. Per quanto riguarda invece l'output che i programmi offrono all'utente attraverso le loro finestre, non esiste il concetto di finestra attiva: l'output può avvenire su qualsiasi finestra, selezionata o meno, e perfino sulle finestre oscurate. Così, mentre è sempre uno solo il task che riceve in un dato istante l'input dall'utente, non c'è limite al numero di task che possono inviare i loro output alle rispettive finestre. Si possono perciò avere diversi task, ciascuno con la propria priorità, che operano su dati forniti in precedenza dall'utente. A questo proposito si veda anche la spiegazione della funzione SetTaskPri della libreria Exec.

Comunque, anche se l'utente continua a inoltrare input alla stessa finestra, e quindi allo stesso task, se questo è in comunicazione con altri task l'input può arrivare anche a loro attraverso il meccanismo dei messaggi.

Tutte le finestre sono associate a schermi. Tutte le finestre associate a un determinato schermo si adeguano alle sue caratteristiche video, tra cui il modo video e i colori.

Seguendo gli spostamenti che l'utente imprime al mouse, sullo schermo video si sposta un puntatore di selezione che per default ha la forma di una freccia rossa orientata in altro a sinistra. Tale puntatore consente all'utente di effettuare selezioni, di cambiare la finestra attiva, e in generale di manipolare gli schermi e le finestre. La sua forma di default può essere facilmente modificata tramite il tool Preferences, oppure attraverso le funzioni SetPointer e ClearPointer. L'utente può spostare il puntatore di selezione attraverso lo schermo sia con il mouse sia con la tastiera.

I tipi di schermo

Il sistema Intuition prevede due tipi di schermo: standard o personalizzato (custom). Lo schermo Workbench è il solo schermo standard, e viene aperto per default da Intuition. Ogni altro componente della presentazione di Intuition (finestre, gadget, requester...) risulta definito in riferimento allo schermo cui è associato. In particolare, tutte le finestre si adeguano alle caratteristiche (modo video, colori...) degli schermi in cui sono state aperte. Perciò, se un programma ha bisogno di finestre che abbiano caratteristiche video diverse da quelle dello schermo Workbench, è necessario creare un nuovo schermo dotato delle caratteristiche desiderate e poi aprire al suo interno le nuove finestre.

Gli schermi standard differiscono da quelli personalizzati per tre aspetti fondamentali. Primo, gli schermi standard (con l'importante eccezione costituita dallo schermo Workbench) si chiudono e spariscono quando tutte le finestre che vi operano vengono chiuse.

Secondo, uno schermo standard di Intuition, sempre con l'eccezione dello schermo Workbench (si veda, a questo proposito, la funzione OpenWorkBench), viene aperto automaticamente nel momento in cui si apre una delle sue finestre tramite la funzione OpenWindow. Ciò rappresenta un modo indiretto per aprire uno schermo standard. Gli schermi personalizzati vengono invece attivati con una procedura diversa; si veda, a questo proposito, la spiegazione della funzione OpenScreen.

Terzo, non è possibile alterare le caratteristiche video degli schermi standard. I parametri che stabiliscono queste caratteristiche sono predefiniti nelle strutture di schermo standard del sistema, e sono mantenuti da Intuition. Ciò consente a tutti i programmi che adoperano schermi standard di presentare un'interfaccia utente standardizzata. Ancora una volta, la sola eccezione è costituita dallo schermo Workbench, che può essere alterato dall'utente attraverso il programma Preferences o cambiato dal programmatore ridefinendo i parametri della struttura Preferences.

Il più importante schermo standard è lo schermo Workbench, uno schermo in alta risoluzione in genere senza interlace (640 x 256 pixel nello standard PAL). I creatori di programmi applicativi che visualizzano l'output prevalentemente in caratteri si rendono conto ben presto che lo schermo Workbench è il più comodo per aprirvi finestre. Prima di tutto non si chiude automaticamente quando vengono chiuse tutte le finestre che vi operano (al contrario degli altri schermi standard). Tuttavia si noti che è possibile chiudere lo schermo Workbench servendosi della funzione CloseWorkBench. Effettuando questa chiamata viene anche liberata la memoria per esso allocata.

Inoltre, con lo schermo Workbench non ci si deve preoccupare di assegnare memoria per bitmap e strutture come sarebbe necessario usando un diverso schermo standard.

Il programma *Workbench*, che è residente in ROM e viene attivato tramite il comando LOADWB del CLI, permette di usare il sistema di gestione dei file dell'Amiga in modo molto semplice, e sfrutta l'omonimo schermo. In ambiente *Workbench* è possibile associare icone ai file e ai programmi, ed è il *Workbench* stesso a mandare in esecuzione le applicazioni o a caricare i file di dati quando l'utente ne seleziona le icone.

Che il programma *Workbench* sia stato attivato o no, i programmi hanno sempre accesso alla libreria delle funzioni correlate al *Workbench*, cioè alle funzioni della libreria Icon tramite le quali è possibile creare e manipolare icone e oggetti. Queste funzioni sono trattate nel capitolo 7.

Per soddisfare particolari esigenze, i programmi possono anche servirsi di schermi personalizzati. È possibile creare due categorie di schermi personalizzati. I primi sono controllati da Intuition, e non richiedono ai programmi l'uso di alcuna funzione appartenente alla libreria Graphics. Il secondo tipo richiede invece queste funzioni per scrivere direttamente nella bitmap di schermo.

Se si decide di adoperare uno schermo personalizzato del secondo tipo, è necessario definire dettagliatamente l'intera bitmap e la relativa struttura RastPort. È possibile inoltre produrre animazioni a colori, scroll di schermo, disegni di linee, aree costituite da retini (matrici grafiche), e molti altri aspetti particolari della presentazione. È consentito combinare un tale schermo personalizzato con gli schermi di Intuition, le finestre e le altre entità grafiche. Le interazioni prodotte dalla presenza di queste definizioni video fuori standard possono però portare a effetti imprevedibili. Perciò, a meno che non si possieda un'ampia esperienza di programmazione, ci si dovrebbe limitare a schermi standard o a schermi personalizzati gestiti da Intuition.

Per progettare schermi personalizzati si può sfruttare tutto ciò che Intuition mette a disposizione (testo, gadget, requester, menu, immagini grafiche...), oppure predisporre tutto per conto proprio. Ovviamente si può sempre scegliere di sfruttare in parte le caratteristiche di Intuition e in parte task di propria progettazione.

I programmi sono in grado d'introdurre testo in ogni schermo e in ogni finestra. A questo scopo, è possibile creare una propria fonte-carattere, oppure usare quella di sistema, Topaz. Topaz è una fonte a spaziatura fissa (cioè non proporzionale) disponibile in due corpi (Topaz-60 e Topaz-80). La fonte Topaz-60 possiede caratteri alti 9 linee di schermo e produce 64 caratteri per riga in alta risoluzione e 32 in bassa risoluzione. La fonte Topaz-80 possiede caratteri alti 8 linee di schermo e produce 80 caratteri per linea in alta risoluzione e 40 in bassa risoluzione.

I tipi di finestra

Intuition mette a disposizione un'ampia varietà di finestre. Innanzitutto, c'è la finestra senza bordi, che viene generata impostando il flag BORDERLESS nella struttura NewWindow, la struttura che occorre definire per aprire una finestra. In tutti gli altri tipi di finestra Intuition produce un bordino di contorno. Sebbene una finestra senza bordi non abbia confini visibili, non è comunque priva di confini. I contorni invisibili sono determinati dalla posizione delle scritte e dei vari gadget presenti sulle aree di confine della finestra. Tuttavia, se la finestra viene creata senza testi o gadget che ne delimitino i bordi, non si distinguerà dallo schermo video di fondo. Questa soluzione risulta abitualmente fonte di confusione ma può essere adottata per creare effetti particolari.

Per esempio, assegnando a una finestra di questo tipo la misura del video

e dichiarandone l'uso in modalità backdrop, diventa possibile disegnarvi quasi con la stessa libertà di cui si godrebbe scrivendo direttamente nella memoria video di uno schermo personalizzato. In questo modo, inoltre, non si corre il rischio d'interferire su un menu o su altre finestre eventualmente presenti sullo schermo nel momento in cui si disegna.

Una seconda categoria di finestra è costituita dal tipo gimmezerozero, che si ottiene attivando il flag GIMMEZEROZERO nella struttura NewWindow. Essa consiste praticamente in due finestre contemporanee: una interna e una esterna. La finestra esterna contiene il titolo, i gadget ed eventualmente il contorno. La finestra interna contiene tutte le altre informazioni grafiche. Quando si crea una finestra gimmezerozero, vengono automaticamente assegnate due bitmap separate per la finestra interna e per quella esterna. Il maggior vantaggio della finestra gimmezerozero è la libertà di disegnare nella finestra interna senza doversi preoccupare di danneggiare il titolo, i gadget o i contorni.

La terza categoria di finestra è costituita dal tipo backdrop, che si genera attivando il flag BACKDROP nella struttura NewWindow. Una finestra backdrop appare, in ogni caso, dietro a tutte le altre finestre di uno schermo Intuition. La sua caratteristica più utile è che resta sempre sul fondo, anche quando tutte le altre finestre si sovrappongono e vengono riordinate in profondità con gli appositi gadget.

La finestra backdrop differisce dalle altre per due particolarità:

1. Si trova sempre dietro a tutte le altre finestre dello schermo e anche dietro a ogni altra backdrop aperta in precedenza. Non è possibile, sullo stesso schermo, ordinare in profondità le finestre per disporne una di altro tipo dietro alla finestra backdrop.

2. L'unico gadget che può essere abbinato alla finestra backdrop è quello di chiusura.

La quarta categoria di finestra è quella superbitmap, specificata impostando il flag SUPERBITMAP nella struttura NewWindow. Una finestra superbitmap ha una propria bitmap che può essere più estesa della bitmap dello schermo a cui appartiene, ed è appunto da questa caratteristica che deriva il suo nome. La superbitmap contiene tutte le informazioni di bitplane necessarie per definire la finestra video, tanto nelle parti visibili quanto in quelle invisibili. Per questa ragione, la superbitmap è in grado di fornire tutte le informazioni che servono per ricostruire la finestra, nel caso che una sua parte sia stata momentaneamente coperta a causa di un'operazione dell'utente.

I titoli di schermi e finestre

Ogni schermo è associato a due titoli: quello di default e quello in uso. Quello di default è definito nella struttura NewScreen e appare quando lo schermo viene aperto per la prima volta. Il titolo in uso è quello associato alla finestra attiva dello schermo; le sue caratteristiche (colori, stili...) dipendono

dalle scelte fatte impostando la definizione della finestra.

Ogni finestra di uno schermo (standard o personalizzato) può avere una sua intestazione personale, che appare nella barra titolo della finestra, e un'intestazione di schermo, che può apparire nella barra titolo dello schermo quando la finestra è attiva. La funzione SetWindowTitles di Intuition permette di specificare, cambiare o cancellare entrambi questi titoli. Anche la funzione ShowTitle entra in gioco nelle interazioni tra titolo di finestra e titolo di schermo.

Gli sprite in ambiente Intuition

Generalmente né gli sprite semplici né quelli virtuali hanno un comportamento soddisfacente nel sistema Intuition. Le funzioni della libreria Graphics controllano gli sprite indipendentemente dalla rappresentazione video di Intuition. Ciò solleva diversi problemi.

- Gli sprite non possono essere assegnati a un particolare schermo: appaiono sempre davanti a ogni schermo di Intuition.
- Quando uno schermo viene spostato, gli sprite non lo seguono automaticamente. Se si desidera che questo avvenga è necessario chiamare espressamente le funzioni di animazione MoveSprite o Animate, della libreria Graphics per compensare il movimento dello schermo.
- Gli sprite hardware possono spostarsi al di fuori delle finestre e degli schermi.
- Lo sprite 0 non è disponibile in quanto costituisce il puntatore standard di Intuition.

Le strutture del sistema Intuition

Ci sono diverse strutture definite e usate internamente dal software del sistema Intuition. Alcune di esse sono statiche; una volta che il task le ha impostate in genere non vengono più alterate per tutta l'esecuzione del task. Le strutture NewWindow e NewScreen costituiscono esempi di strutture statiche. Altre hanno un comportamento dinamico, cioè vengono continuamente aggiornate con nuovi valori che variano seguendo i movimenti del mouse. Le strutture Window e Screen sono esempi di strutture dinamiche, cioè di strutture all'interno delle quali alcuni parametri continuano a variare a mano a mano che l'utente interagisce con la relativa finestra o con il relativo schermo.

Le strutture di Intuition contengono le seguenti categorie d'informazioni:

- un insieme di parametri di puntamento a sotto-strutture. Citiamo come esempio il parametro FirstGadget della struttura NewWindow; tale parametro punta al primo gadget della lista concatenata dei gadget che possono essere presentati nella finestra.

- Un insieme di parametri che definiscono posizione e dimensioni dell'elemento video. Per esempio, le strutture NewWindow, NewScreen, Window e Screen contengono tutte i parametri LeftEdge, TopEdge, Width e Height. Tali parametri definiscono la misura di uno specifico elemento video.

- Uno o più parametri Flag. I bit in questi parametri Flag controllano varie opzioni associate alla struttura a cui si riferiscono e a eventuali strutture a essa correlate.

- Un insieme di parametri di controllo per il ridimensionamento degli elementi video. Ogni elemento video che l'utente è in grado di ingrandire o ridurre possiede questo tipo di parametri, che servono per imporre dei limiti al ridimensionamento. L'esempio più evidente è la struttura Window che contiene i parametri MinWidth, MinHeight, MaxWidth e MaxHeight. Questi parametri determinano i valori minimi e massimi entro i quali è possibile variare dinamicamente le dimensioni della finestra.

- Un insieme di parametri generici, richiesti per definire completamente l'elemento video. Per esempio, la struttura Window comprende i parametri MouseX e MouseY per indicare la posizione del puntatore sullo schermo.

La struttura NewWindow

La struttura NewWindow contiene i parametri necessari per definire una nuova finestra. Ecco com'è definita:

```
struct NewWindow {
    SHORT LeftEdge, TopEdge;
    SHORT Width, Height;
    UBYTE DetailPen, BlockPen;
    ULONG IDCMPFlags;
    ULONG Flags;
    struct Gadget *FirstGadget;
    struct Image *CheckMark;
    UBYTE *Title;
    struct Screen *Screen;
    struct BitMap *BitMap;
```

```
    SHORT MinWidth, MinHeight;
    SHORT MaxWidth, MaxHeight;
    USHORT Type;
};
```

Il significato dei parametri presenti nella struttura NewWindow è il seguente:

- il parametro LeftEdge contiene la coordinata x iniziale dell'angolo superiore sinistro della finestra.
- Il parametro TopEdge contiene la coordinata y iniziale dell'angolo superiore sinistro della finestra.
- Il parametro Width contiene la larghezza della finestra.
- Il parametro Height contiene l'altezza della finestra.
- Il parametro DetailPen contiene il numero del registro di colore impiegato per i particolari della finestra, come i gadget o il testo della barra titolo.
- Il parametro BlockPen contiene il numero del registro di colore impiegato per il riempimento dei blocchi. Lo stesso colore viene utilizzato per la barra titolo della finestra.
- Il parametro IDCMPFlags contiene un insieme di flag IDCMP. Per la descrizione di questi flag si veda la spiegazione della funzione ModifyIDCMP.
- Il parametro Flags contiene diversi flag relativi ai gadget e ad altre caratteristiche della finestra.
- Il parametro FirstGadget punta al primo elemento di una lista concatenata di strutture Gadget che definiscono i gadget della finestra.
- Il parametro CheckMark punta a una struttura Image per un'immagine personalizzata del segno di attivato nella presentazione delle voci dei menu.
- Il parametro Title punta alla stringa di testo a terminazione nulla che appare nella barra titolo della finestra.
- Il parametro Screen punta a una struttura Screen per schermi personalizzati. La bitmap della finestra costituisce una sotto-sezione di questa bitmap di schermo.

- Il parametro BitMap punta a una struttura BitMap per le finestre alle quali è associata una superbitmap.

- I parametri MinWidth, MinHeight e MaxWidth, MaxHeight contengono l'insieme dei valori minimi e massimi per le dimensioni della finestra.

- Il parametro Type determina il tipo di schermo per la finestra; può essere WORKBENCHSCREEN o CUSTOMSCREEN.

La struttura Window

La struttura Window contiene le informazioni di gestione di una finestra. Questa struttura viene creata attraverso una chiamata alla funzione OpenWindow, che richiede come argomento l'indirizzo della struttura NewWindow precedentemente definita in RAM. Quando OpenWindow restituisce il controllo, in memoria risulta presente una struttura Window opportunamente allocata e inizializzata secondo le indicazioni predisposte dal task nella struttura NewWindow. In seguito, quando la finestra viene spostata o modificata dall'utente, i parametri della struttura Window vengono aggiornati per rifletterne i cambiamenti; ciò include i parametri di dimensionamento, il numero dei requester in essa visualizzati e qualsiasi altro elemento che ne definisca lo stato.

Ecco com'è definita la struttura Window:

```
struct Window {
        struct Window *NextWindow;
        SHORT LeftEdge, TopEdge;
        SHORT Width, Height;
        SHORT MouseY, MouseX;
        SHORT MinWidth, MinHeight;
        SHORT MaxWidth, MaxHeight;
        ULONG Flags;
        struct Menu *MenuStrip;
        UBYTE *Title;
        struct Requester *FirstRequest;
        struct Requester *DMRequest;
        SHORT ReqCount;
        struct Screen *WScreen;
        struct RastPort *RPort;
        BYTE BorderLeft, BorderTop, BorderRight, BorderBottom;
        struct RastPort *BorderRPort;
        struct Gadget *FirstGadget;
        struct Window *Parent, *Descendent;
        USHORT *Pointer;
        BYTE PtrHeight;
        BYTE PtrWidth;
        BYTE XOffset, YOffset;
```

```
    ULONG IDCMPFlags;
    struct MsgPort *UserPort, *WindowPort;
    struct IntuiMessage *MessageKey;
    UBYTE DetailPen, BlockPen;
    struct Image *CheckMark;
    UBYTE *ScreenTitle;
    SHORT GZZMouseX;
    SHORT GZZMouseY;
    SHORT GZZWidth;
    SHORT GZZHeight;
    UBYTE *ExtData;
    BYTE *UserData;
    struct Layer *Wlayer;
    struct TextFont *IFont;
};
```

Il significato dei parametri presenti nella struttura Window è il seguente:

- il parametro NextWindow è un puntatore a una struttura Window che rappresenta la finestra successiva nella lista concatenata delle finestre appartenenti allo schermo.

- Il parametro LeftEdge contiene la coordinata x del bordo sinistro della finestra. Il sistema altera questo parametro ogni volta che la finestra viene spostata orizzontalmente.

- Il parametro TopEdge contiene la coordinata y del bordo superiore della finestra. Il sistema altera questo parametro ogni volta che la finestra viene spostata verticalmente.

- Il parametro Width contiene la larghezza (in pixel) della finestra. Il sistema altera questo parametro ogni volta che la finestra viene ridimensionata orizzontalmente.

- Il parametro Height contiene l'altezza (in pixel) della finestra. Il sistema altera questo parametro ogni volta che la finestra viene ridimensionata verticalmente.

- I parametri MouseY e MouseX contengono le coordinate y e x del puntatore del mouse, calcolate rispetto all'angolo superiore sinistro della finestra. Il sistema altera questi parametri, seguendo i movimenti che l'utente imprime al mouse all'interno della finestra.

- I parametri MinWidth e MinHeight contengono la minima larghezza e altezza (in pixel) che la finestra può raggiungere durante i ridimensionamenti.

- I parametri MaxWidth e MaxHeight contengono la massima larghezza e altezza (in pixel) che la finestra può raggiungere durante i ridimensionamenti.

- Il parametro Flags contiene un insieme di flag per la finestra. Si veda anche il file INCLUDE intuition.h.

- Il parametro MenuStrip è un puntatore alla struttura Menu che rappresenta il primo menu di una lista concatenata di menu associata alla finestra.

- Il parametro Title è un puntatore alla stringa di testo a terminazione nulla che rappresenta il titolo della finestra, ovvero la scritta che appare nella barra titolo della finestra nel momento in cui viene aperta.

- Il parametro FirstRequest è un puntatore a una struttura Requester che rappresenta il primo requester in una lista di requester relativi alla finestra.

- Il parametro DMRequest è un puntatore a una struttura Requester che rappresenta un DMRequester per la finestra. Si vedano, a questo proposito, le funzioni ClearDMRequest e SetDMRequest.

- Il parametro ReqCount è un contatore, che tiene conto del numero di requester attivi che bloccano l'input della finestra.

- Il parametro WScreen è un puntatore alla struttura Screen che rappresenta lo schermo all'interno del quale appare la finestra. Se la finestra è stata aperta in uno schermo personalizzato, si dispone già dell'indirizzo della sua struttura Screen, ma se la finestra è stata aperta in uno schermo standard (per esempio quello Workbench), tale parametro fornisce l'indirizzo della struttura Screen a esso associata. Si veda a questo proposito la funzione GetScreenData.

- Il parametro RPort è un puntatore alla struttura RastPort che contiene le informazioni per il controllo del disegno all'interno della finestra.

- I parametri BorderLeft, BorderTop, BorderRight e BorderBottom contengono le coordinate dei bordi sinistro, alto, destro e basso della finestra.

- Il parametro BorderRPort è un puntatore alla struttura RastPort che contiene le informazioni richieste per disegnare i bordi della finestra.

- Il parametro FirstGadget è un puntatore a una struttura Gadget che rappresenta il primo gadget della lista concatenata di gadget della finestra.

- I parametri Parent e Descendant sono puntatori ad altre strutture Window e sono utilizzati dal sistema al momento dell'apertura e della chiusura della finestra.

- Il parametro Pointer è un puntatore ai dati che definiscono lo sprite che Intuition deve impiegare per il puntatore del mouse quando la finestra viene attivata.

- I parametri PtrHeight e PtrWidth contengono altezza e larghezza (in pixel) del puntatore del mouse. Sono impiegati soltanto per puntatori definiti dall'utente. Il sistema li cambia quando viene alterata la definizione del puntatore tramite il programma Preferences.

- I parametri XOffset e YOffset contengono la posizione del "punto di selezione" (hot spot) del puntatore rispetto all'angolo superiore sinistro dello sprite che lo definisce.

- Il parametro IDCMPFlags contiene un insieme di flag IDCMP a disposizione del programma che detiene la finestra. Si veda la spiegazione relativa alla funzione ModifyIDCMP.

- Il parametro UserPort è un puntatore a una struttura MsgPort che rappresenta la message port (definita anche "user port") che Intuition usa per comunicare con il task che ha definito la finestra. I messaggi IDCMP inviati da Intuition alla finestra, e quindi al task che ne detiene il possesso, vengono accodati a questa particolare message port.

- Il parametro WindowPort è un puntatore a una struttura MsgPort che rappresenta la reply port di Intuition per la finestra. Viene chiamata window port, ed è la message port nella quale Intuition si aspetta di ricevere in risposta i messaggi che ha inviato alla user port.

- Il parametro MessageKey è un puntatore a una struttura IntuiMessage che viene impiegata per scambiare messaggi tra la user port e la window port.

- Il parametro DetailPen contiene il numero del registro colore della penna di disegno utilizzata dalle funzioni grafiche della libreria Graphics e adoperata per presentare i dettagli grafici della finestra.

- Il parametro BlockPen contiene il numero del registro di colore della penna di disegno utilizzata dalle funzioni grafiche della libreria Graphics e adoperata per effettuare il riempimento di aree nella finestra.

- Il parametro CheckMark è un puntatore alla struttura Image utilizzata per definire l'immagine del segno di attivo accanto alle voci di menu selezionate.

- Il parametro ScreenTitle è un puntatore alla stringa di testo a terminazione nulla che rappresenta il titolo di schermo della finestra. È il titolo che appare nella barra titolo di schermo quando la finestra diventa attiva.

- I parametri GZZMouseX e GZZMouseY contengono le coordinate x e y del puntatore (in pixel) relative all'origine interna in una finestra di tipo gimmezerozero.

- I parametri GZZWidth e GZZHeight contengono altezza e larghezza (in pixel) del rettangolo interno in una finestra di tipo gimmezerozero.

- Il parametro ExtData è un puntatore alla prima delle due aree dati definite dal programmatore per la finestra.

- Il parametro UserData è un puntatore alla seconda delle due aree dati definite dal programmatore per la finestra.

- Il parametro WLayer è la copia di un puntatore alla struttura Layer relativa alla finestra.

- Il parametro IFont è un puntatore alla struttura TextFont aperta al momento della creazione della finestra. Nel caso che la fonte venisse cambiata da una o più chiamate alla funzione SetFont, il sistema può comunque rintracciare qui la fonte originale da liberare con CloseFont al momento della chiusura della finestra.

La struttura NewScreen

La struttura NewScreen serve per aprire un nuovo schermo nel sistema. Dev'essere allocata e inizializzata dal task, il quale ne indica poi l'indirizzo come argomento nella chiamata alla funzione OpenScreen. La struttura NewScreen definisce tutte le caratteristiche video del nuovo schermo.

La struttura NewScreen è definita come segue:

```
struct NewScreen {
    SHORT LeftEdge, TopEdge;
    SHORT Width, Height;
    UBYTE DetailPen, BlockPen;
    USHORT ViewModes;
    USHORT Type;
    struct TextAttr *Font;
    UBYTE *DefaultTitle;
    struct Gadget *Gadgets;
    struct BitMap *CustomBitMap;
};
```

Il significato dei parametri presenti nella struttura NewScreen è il seguente:

- il parametro LeftEdge indica a OpenScreen la posizione x che il lato sinistro dello schermo deve assumere quando viene aperto.
  Attualmente, Intuition impone che questo parametro valga zero, ma in future versioni potrebbe essere possibile muovere gli schermi anche orizzontalmente.

- Il parametro TopEdge indica a OpenScreen la posizione y che deve assumere il lato superiore dello schermo quando viene aperto.

- Il parametro Width indica la larghezza in pixel dello schermo. In generale vale 320 per il modo a bassa risoluzione e 640 per quello in alta risoluzione.

- Il parametro Height indica l'altezza in pixel dello schermo. Nel sistema PAL la misura massima senza interlace è 256, mentre diventa 512 con l'interlace.
  Nella determinazione di questo valore, si tenga conto che sommandolo al valore memorizzato nel parametro TopEdge si deve sempre ottenere un risultato maggiore o uguale all'altezza massima in pixel dello schermo nel modo video impostato (256 senza interlace e 512 con interlace). Intuition, infatti, è in grado di sovrapporre gli schermi, ma non è ancora capace di visualizzare uno schermo che disposto sopra un altro ne lasci intravedere non solo la parte superiore ma anche quella inferiore. In pratica, gli schermi possono assumere soltanto posizioni tali da arrivare fino al fondo del video.

- Il parametro Depth contiene il numero di bitplane della bitmap di schermo. Questo valore può variare tra 1 e 6 per il modo video single-playfield, e tra 2 e 6 per il modo video dual-playfield.

- Il parametro DetailPen contiene il numero del registro di colore impiegato per i particolari dello schermo, come i gadget o il testo della barra titolo.

- Il parametro BlockPen contiene il numero del registro di colore impiegato per il riempimento delle aree, cioè per lo sfondo. Si tratta anche del colore utilizzato per la barra titolo dello schermo.

- Il parametro Type indica il tipo di schermo. Lo si deve impostare a CUSTOMSCREEN.

- Il parametro Font punta alla struttura TextAttr associata a questo schermo. Va impostato a NULL se si vogliono impiegare le fonti di default di Intuition (Topaz-60 e Topaz-80).

- Il parametro DefaultTitle punta alla stringa di testo a terminazione nulla che si vuole visualizzare nella barra titolo dello schermo.

- Il parametro Gadgets punta a una struttura Gadget che definisce il primo gadget di una lista concatenata di gadget relativi allo schermo.

- Il parametro CustomBitMap punta a una struttura BitMap usata per definire la memoria video dello schermo, se si desidera utilizzarne una specifica piuttosto che una allocata automaticamente da Intuition. In questo caso dev'essere impostato anche il flag CUSTOMBITMAP nel parametro Type.

La struttura Screen

La struttura Screen contiene informazioni sulle condizioni di uno schermo personalizzato. Viene creata dalla funzione OpenScreen sulla base dei dati contenuti nella struttura NewScreen definita dal task e indicata come argomento della funzione.

Ecco com'è definita la struttura Screen:

```
struct Screen {
    struct Screen *NextScreen;
    struct Window *FirstWindow;
    SHORT LeftEdge, TopEdge;
    SHORT Width, Height;
    SHORT MouseY, MouseX;
    USHORT Flags;
    UBYTE *Title;
    UBYTE *DefaultTitle;
    BYTE BarHeight, BarVBorder, BarHBorder;
    BYTE MenuVBorder, MenuHBorder;
    BYTE WBorTop, WBorLeft, WBorRight, WBorBottom;
    struct TextAttr *Font;
    struct ViewPort *ViewPort;
    struct RastPort *RastPort;
    struct BitMap *BitMap;
    struct Layer_Info *LayerInfo;
    struct Gadget *FirstGadget;
    UBYTE DetailPen, BlockPen;
    USHORT SaveColor?Ø;
    struct Layer *BarLayer;
    UBYTE *ExtData;
    BYTE *UserData;
};
```

Il significato dei parametri presenti nella struttura Screen è il seguente:

- il parametro NextScreen è un puntatore a una struttura Screen che rappresenta lo schermo successivo in una lista concatenata di schermi personalizzati.

- Il parametro FirstWindow è un puntatore alla struttura Window che rappresenta la prima finestra dello schermo.

- Il parametro LeftEdge contiene la coordinata x del bordo sinistro dello schermo (dopo uno spostamento o un ridimensionamento). Il sistema mantiene questo parametro costantemente aggiornato e il programma può rilevarne il contenuto per vedere dove l'utente ha collocato lo schermo.

- Il parametro TopEdge contiene la coordinata y del bordo superiore dello schermo. Il sistema mantiene questo parametro costantemente aggiornato e il programma può rilevarne il contenuto per vedere dove l'utente ha collocato lo schermo.

- Il parametro Width contiene la larghezza (in pixel) dello schermo.

- Il parametro Height contiene l'altezza corrente (espressa in numero di pixel) dello schermo definito dalla struttura Screen. Nel sistema PAL può assumere un valore fino a 256 per un video senza interlace e fino a 512 per un video con interlace.

- I parametri MouseY e MouseX contengono le coordinate y e x del puntatore del mouse, rispetto all'angolo superiore sinistro dello schermo. Il sistema mantiene questi parametri costantemente aggiornati e il programma può rilevarne i contenuti per vedere dove l'utente ha collocato il puntatore sullo schermo.

- Il parametro Flags contiene un insieme di flag, fra cui WORKBENCH-SCREEN e CUSTOMSCREEN. Si veda anche il file INCLUDE intuition.h.

- Il parametro Title è un puntatore a una stringa di testo a terminazione nulla che rappresenta il titolo dello schermo.

- Il parametro DefaultTitle è un puntatore a una stringa di testo a terminazione nulla che rappresenta il titolo di default dello schermo. Si tratta del testo che appare nella barra titolo dello schermo se la finestra attiva non definisce un titolo di schermo. È anche il titolo che viene visualizzato quando lo schermo si apre per la prima volta o non vi è nessuna finestra attiva.

- Il parametro BarHeight contiene l'altezza della barra titolo dello schermo.

- Il parametro BarVBorder contiene la coordinata y della linea di fondo della barra titolo dello schermo.
- Il parametro BarHBorder contiene la coordinata x del lato destro della barra titolo dello schermo.
- Il parametro MenuVBorder contiene la coordinata y della linea di fondo della barra menu per i menu associati a questa struttura Screen.
- Il parametro MenuHBorder contiene la coordinata x del lato destro della barra menu per i menu associati a questa struttura Screen.
- Il parametro WBorTop contiene la coordinata y del bordo superiore dello schermo.
- Il parametro WBorLeft contiene la coordinata x del bordo sinistro dello schermo.
- Il parametro WBorRight contiene la coordinata x del bordo destro dello schermo.
- Il parametro WBorBottom contiene la coordinata y della linea di fondo dello schermo.
- Il parametro Font è un puntatore a una struttura TextAttr che rappresenta la fonte-carattere di default per i testi visualizzati sullo schermo.
- Il parametro ViewPort è una sotto-struttura ViewPort che contiene la definizione di viewport per lo schermo.
- Il parametro RastPort è una sotto-struttura RastPort che contiene le definizioni per il controllo del disegno sullo schermo.
- Il parametro BitMap è una sotto-struttura BitMap che contiene le definizioni della bitmap associata allo schermo.
- Il parametro LayerInfo è una sotto-struttura Layer che contiene la definizione di layer per lo schermo.
- Il parametro FirstGadget è un puntatore a una struttura Gadget che definisce il primo gadget di una lista concatenata di gadget relativi allo schermo.
- Il parametro DetailPen contiene il numero del registro colore della penna di disegno utilizzata dalle funzioni grafiche della libreria Graphics per presentare i dettagli grafici (gadget e testo nella barra titolo dello schermo).

- Il parametro BlockPen contiene il numero del registro colore della penna di disegno utilizzata dalle funzioni grafiche della libreria Graphics per effettuare il riempimento delle aree nello schermo considerato (per esempio l'area di sfondo della barra titolo).

- Il parametro SaveColor0 viene utilizzato dalla funzione DisplayBeep per tenere conto del valore da ripristinare dopo il lampeggio dello schermo.

- Il parametro BarLayer è un puntatore a una struttura Layer che rappresenta il layer utilizzato per definire informazioni grafiche per lo schermo e per le barre titolo dei menu a esso relativi.

- Il parametro ExtData è un puntatore alla prima delle due aree dati definite dal programmatore per lo schermo.

- Il parametro UserData è un puntatore alla seconda delle due aree dati definite dal programmatore per lo schermo.

Seguendo gli spostamenti e le variazioni richieste dall'utente, il sistema cambia i parametri nella struttura Screen per mantenere sempre informazioni aggiornate sullo stato dello schermo: dimensioni, posizione del puntatore e ogni altro elemento indicativo. La struttura Screen include nella sua definizione anche le sotto-strutture RastPort, ViewPort, BitMap e Layer_Info. Tali definizioni possono essere usate dai programmatori più esperti per esaminare e modificare i dettagli di uno schermo di Intuition, sfruttanto le funzioni della libreria Graphics trattate nel capitolo 2.

Le altre strutture

Le funzioni di Intuition operano con parecchie altre strutture:

BitMap	IntuiMessage	MenuItem	Remember
Border	IntuiText	Preferences	Requester
Gadget	Menu	PropInfo	StringInfo
Image			

Per approfondire la conoscenza di queste strutture consigliamo di stampare e studiare il file INCLUDE intuition.h. I nomi delle strutture suggeriscono l'impiego a cui sono destinate.

La programmazione in ambiente Intuition

Il sistema Intuition è una libreria di funzioni. I programmi possono adoperare queste funzioni insieme a semplici strutture di dati per generare interfacce utente molto efficaci.

Per utilizzare Intuition nei propri programmi è opportuno seguire alcuni passi fondamentali:

1. Si inseriscono nel proprio disco per la programmazione in linguaggio C i file INCLUDE intuition.h. Questi file contengono tutte le definizioni delle strutture relative a Intuition, i tipi di dato, le costanti e le macro necessarie. Intuition contiene sei macro che riguardano le operazioni con i menu: MENUNUM, ITEMNUM, SUBNUM, SHIFTMENU, SHIFTITEM e SHIFTSUB. Il solo argomento che adoperano è menuNumber, come viene indicato nella funzione ItemAddress. Si leggano, a questo proposito, le spiegazioni di ciascuna macro.

2. Dal momento che il software sistema di Intuition è realizzato sotto forma di libreria, si deve dichiarare una variabile puntatore denominata IntuitionBase e chiamare la funzione OpenLibrary dell'Exec prima di poterne usare le funzioni. Si noti che il nome della variabile è fondamentale per il corretto uso della libreria.

3. Si devono definire gli schermi personalizzati che si utilizzeranno. Per ognuno, si definisce una struttura NewScreen e si chiama la funzione OpenScreen, indicando l'indirizzo della struttura NewScreen come argomento.

4. Si definiscono le finestre che si utilizzeranno. Per ognuna, si definisce una struttura NewWindow e si chiama la funzione OpenWindow, indicando l'indirizzo della struttura NewWindow come argomento.

Se si vuole adoperare il linguaggio Assembly, si noti che i registri del 68000 sono disposti in ordine crescente dal registro 0 al registro 7 in ciascuna categoria (registri d'indirizzamento e registri dati). Per trovare la corretta associazione dei nomi di registro con gli argomenti delle funzioni Intuition, nella chiamata alla funzione si devono utilizzare i registri d'indirizzamento per gli argomenti puntatori della struttura a iniziare da A0. Allo stesso modo si procede per i registri dati, iniziando da D0. La sola eccezione a queste regole è la funzione UnlockIBase, come si osserva nella relativa spiegazione.

I programmatori che impiegano il linguaggio Assembly possono riferirsi al file INCLUDE intuition.i per sapere quali sono le strutture da usare nella programmazione.

L'interfaccia utente

La Figura 6.1 illustra gli aspetti del tipico schermo video di Intuition. Nell'esempio scelto si vede un unico schermo sul quale sono presenti tre finestre. Le principali caratteristiche dello schermo sono la barra titolo e i gadget per l'ordinamento in profondità. Si osservi che dietro allo schermo visibile potrebbero essercene altri con priorità video più bassa.

Se è stato definito un altro schermo, i gadget per l'ordinamento in profondità possono essere adoperati per mandare lo schermo di primo piano dietro all'altro. Vengono così mostrate all'utente le finestre del nuovo schermo (se ve ne sono di aperte). Questa gestione della sovrapposizione degli schermi, al pari di quella per le finestre, viene effettuata interamente da Intuition senza il benché minimo intervento del task che ha aperto lo schermo.

Le più importanti caratteristiche di ogni finestra sono il gadget di chiusura, il gadget per l'ordinamento in profondità, quello di ridimensionamento e la barra titolo.

Si noti che può essere attiva una sola finestra alla volta, fra tutte quelle aperte nei vari schermi; nella Figura 6.1 è la finestra 3. Si tratta della finestra destinata a ricevere gli input quando l'utente impiega il mouse o la tastiera; i menu associati alla finestra si possono aprire soltanto quando la finestra è attiva. Le finestre possono essere ridimensionate impiegando l'apposito gadget, e trascinate per lo schermo "afferrandone" la barra titolo, detta anche

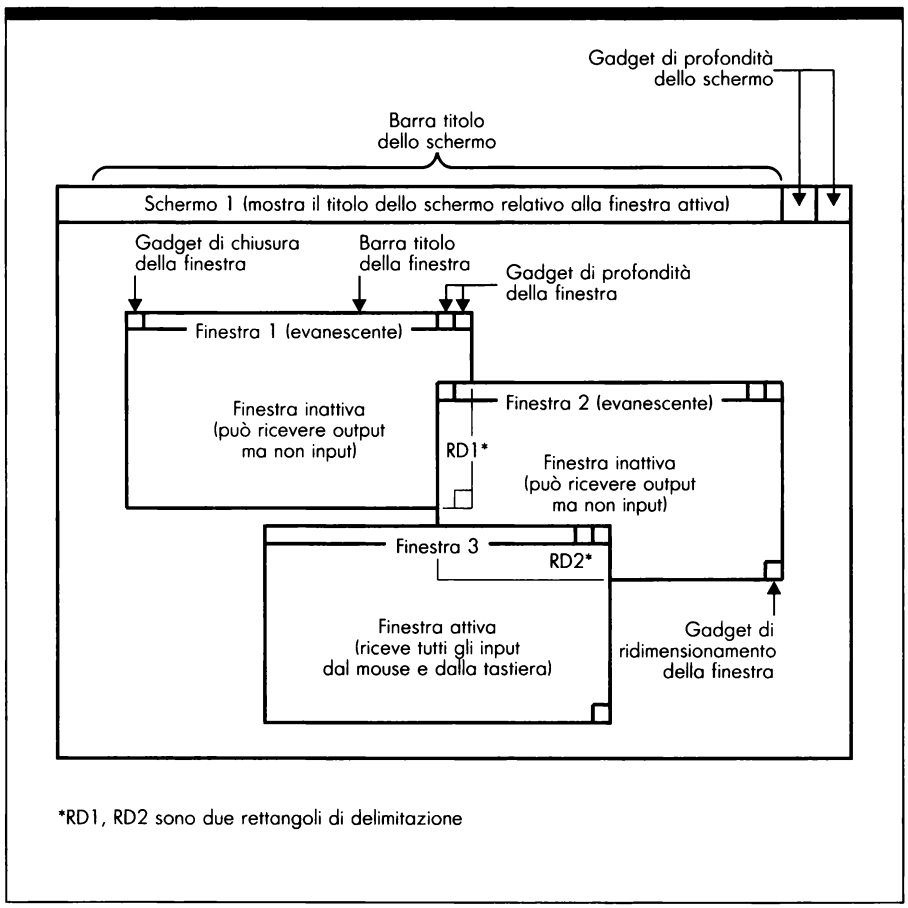


Figura 6.1: *Schermo di Intuition, finestre e gadget di sistema*

barra di spostamento. La finestra su cui si svolge una di queste operazioni diventa automaticamente quella attiva.

Una finestra diventa attiva quando l'utente la seleziona con il mouse in qualsiasi punto al suo interno. In quel momento, il titolo di schermo della finestra appare nella barra titolo dello schermo: quando viene attivata un'altra finestra, il titolo nella barra titolo dello schermo cambia di conseguenza.

Il testo nella barra titolo di una finestra attiva appare evidenziato, mentre quello delle finestre inattive è più "sbiadito".

Quando l'utente preme il pulsante destro del mouse, la barra menu associata alla finestra attiva appare sovrapposta alla barra titolo dello schermo. L'utente può poi portare il puntatore del mouse sul testo di un menu e procedere all'esplorazione di tutti i sotto-menu.

L'apertura dei menu è illustrata nella Figura 6.2: si vede uno schermo con una serie di menu associati alla finestra attiva. I nomi dei menu della finestra appaiono nella barra titolo dello schermo.

Nella figura si vede anche un sotto-menu relativo al primo menu. Si osservi che in ambiente Intuition non è possibile aprire contemporaneamente due ramificazioni diverse di sotto-menu.

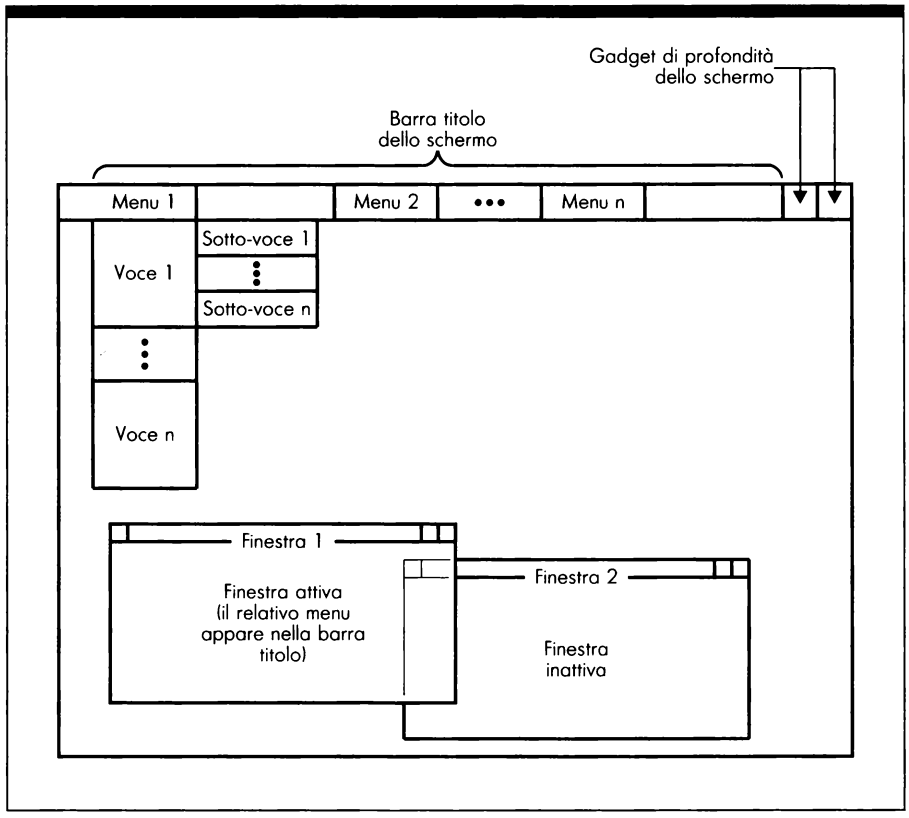


Figura 6.2: *Menu di Intuition, voci dei menu, sotto-voci dei menu*

collegamenti fra le strutture di Intuition

Le principali caratteristiche del software sistema di Intuition sono illustrate da una serie di diagrammi presentati nelle figure dalla 6.3/6.7. Questi diagrammi illustrano la maggior parte dei collegamenti che interessano le strutture di Intuition. La base della programmazione in ambiente Intuition consiste nella comprensione di questi collegamenti fra strutture di dati.

Si tenga presente che in queste figure non compaiono tutti i parametri di struttura, ma soltanto quelli che riguardano i collegamenti fra strutture. Per altre informazioni sui parametri richiesti per definire una struttura si consultino i file INCLUDE relativi a Intuition. Si tenga inoltre presente che non tutti questi parametri sono di competenza del programmatore; alcuni vengono inizializzati e mantenuti dal sistema.

Il rettangolo grande nelle figure 6.3/6.7 rappresenta la struttura principale della figura. Per esempio, nella Figura 6.3 viene trattata la struttura Screen. Ogni rettangolo piccolo all'interno del rettangolo che rappresenta la struttura Screen identifica una sotto-struttura. Le frecce che partono dai parametri puntatori della struttura individuano le strutture a essa correlate.

I collegamenti relativi alla struttura Screen

Ogni struttura Screen può essere collegata a una o più strutture Window. Le strutture Window sono rappresentate dalla serie di rettangolini in alto a destra. Il parametro FirstWindow punta alla prima di queste strutture Window fra loro concatenate. Inoltre, ciascuna struttura Screen può essere collegata a

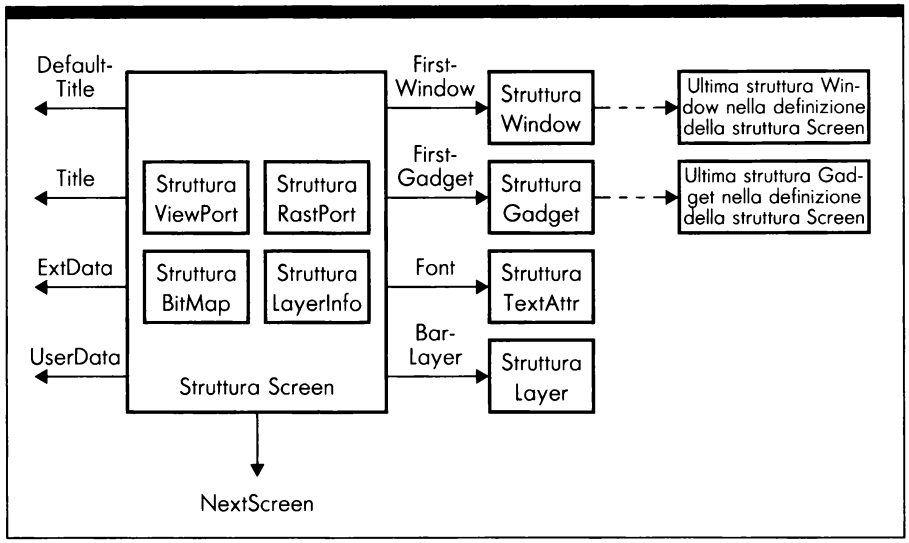


Figura 6.3: *Collegamenti fra strutture per la struttura Screen*

una o più strutture Gadget. Il parametro FirstGadget punta alla prima di queste strutture Gadget fra loro concatenate. Le strutture Window e Gadget contengono, a loro volta, puntatori alla successiva struttura nella lista a cui appartengono (e non alla precedente; liste di questo tipo si dicono a concatenazione semplice).

La struttura Screen punta inoltre a una struttura TextAttr tramite il parametro puntatore Font; a una struttura Layer tramite il parametro puntatore BarLayer; a due stringhe di testo tramite i parametri DefaultTitle e Title; a un insieme di dati di schermo definiti dal programmatore tramite i parametri ExtData e UserData. Questi ultimi appaiono sul lato sinistro della Figura 6.3.

Ogni struttura Window appartiene a una lista che comprende tutte le finestre aperte in un particolare schermo. La prima struttura della lista è individuata in memoria dal parametro FirstWindow della struttura Screen.

Infine ogni struttura Screen contiene un puntatore (denominato NextScreen) all'eventuale struttura Screen successiva.

Si tenga sempre presente che la gestione degli schermi consiste soltanto nell'instaurare questi collegamenti; compiuto questo lavoro, gli schermi possono essere riordinati in profondità dai task (utilizzando direttamente le funzioni di Intuition) o dall'utente (agendo con il mouse).

I collegamenti relativi alla struttura Window

La Figura 6.4 mostra una struttura Window, rappresentata da un grande rettangolo e da una serie di puntatori a rettangoli più piccoli che rappresentano le strutture puntate dalla struttura Window. Ogni struttura Window contiene inoltre puntatori a diversi tipi di dati in memoria: ScreenTitle, Title, Pointer, ExtData e UserData. Essi appaiono sul lato sinistro della figura.

A ogni struttura Window si possono associare tre liste semplici che definiscono i menu (strutture Menu), i requester (strutture Requester) e i gadget (strutture Gadget) relativi alla finestra.

In ogni struttura Window è memorizzato nel parametro puntatore WScreen l'indirizzo della struttura Screen che definisce lo schermo in cui la finestra è stata aperta.

Nove ulteriori parametri puntatore (da RPort a CheckMark) puntano alle altre strutture richieste per gestire la finestra.

Inoltre, ogni struttura Window contiene un puntatore (denominato NextWindow) a un'altra eventuale struttura Window collegata a quella considerata. Le strutture Window relative alle finestre contenute nello stesso schermo sono infatti concatenate in una lista semplice associata alla struttura Screen che definisce lo schermo.

I collegamenti relativi alla struttura Requester

La Figura 6.5 mostra una struttura Requester, rappresentata da un rettangolo grande, e una serie di puntatori a rettangoli più piccoli che identificano altre strutture. Ogni struttura Requester può puntare a un'altra

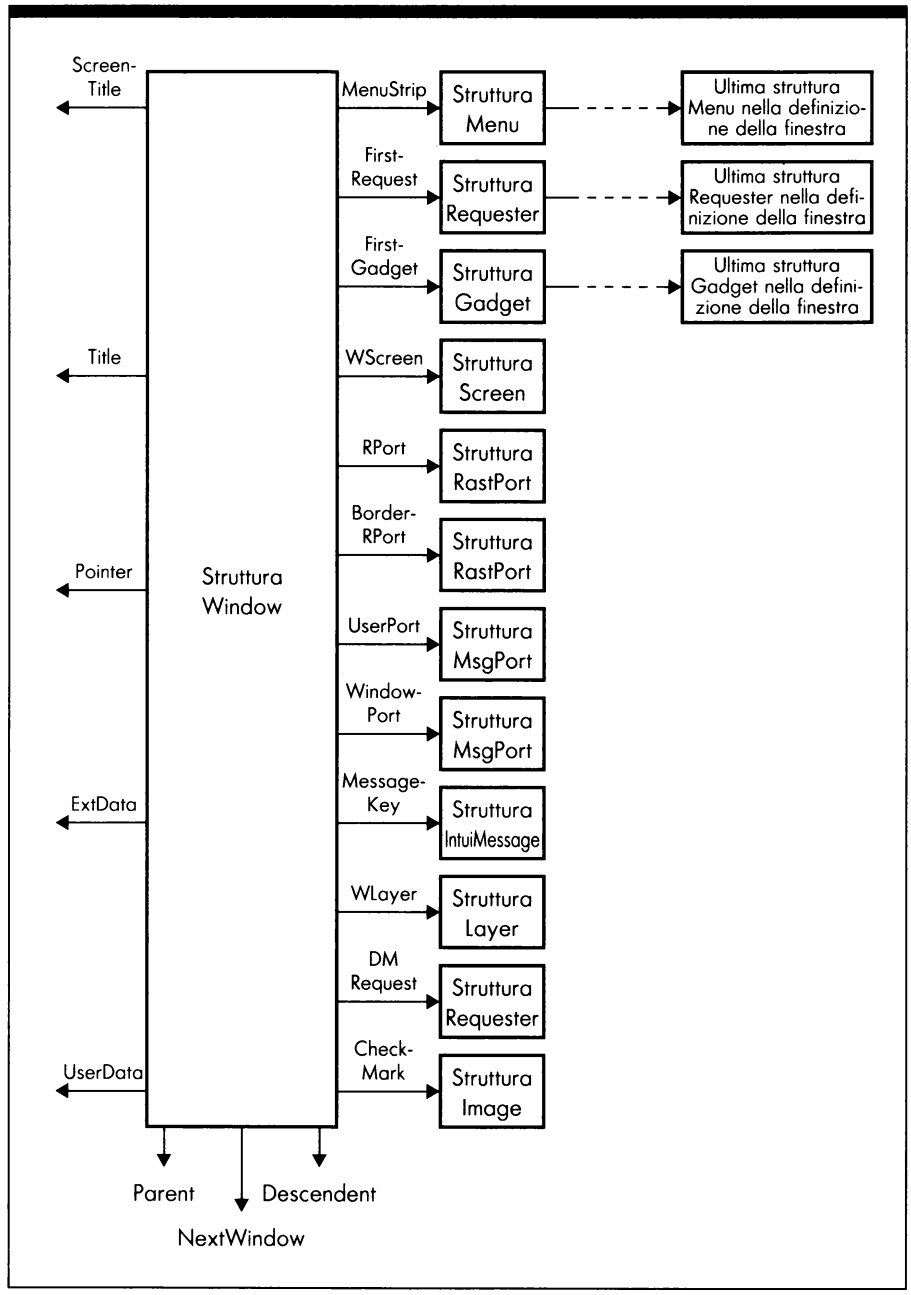


Figura 6.4: *Collegamenti fra strutture per la struttura Window*

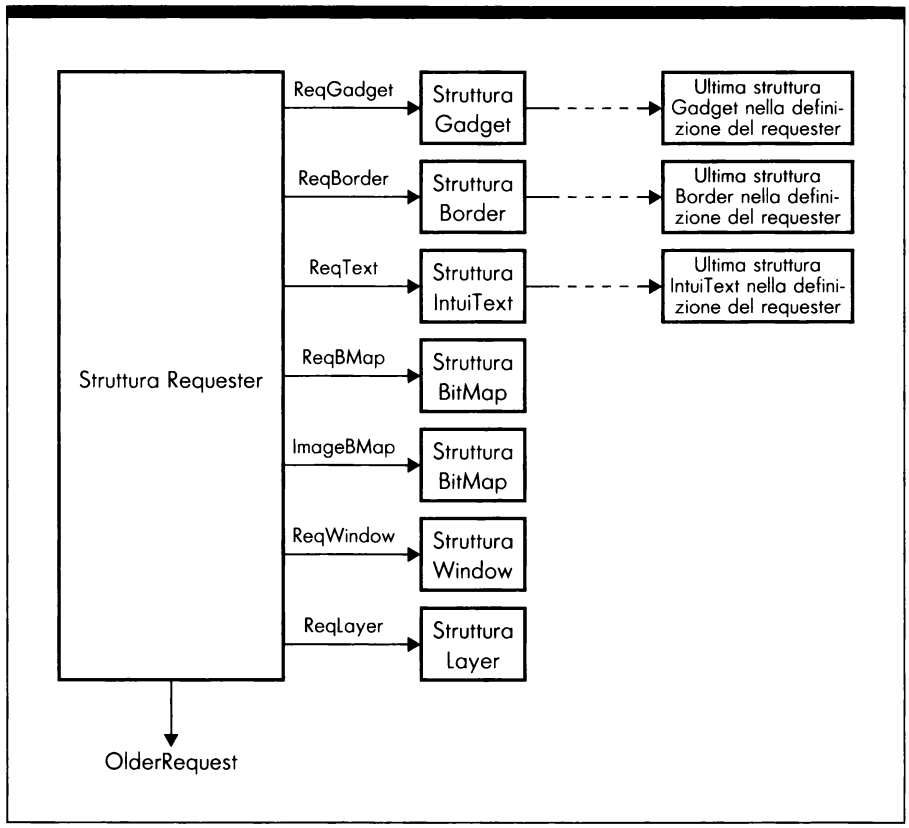


Figura 6.5: *Collegamenti fra strutture per la struttura Requester*

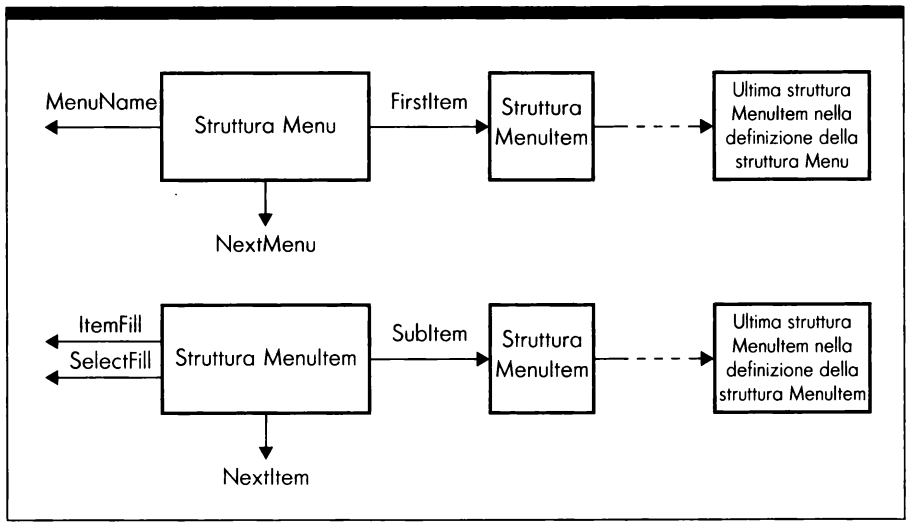


Figura 6.6: *Collegamenti fra strutture per le strutture Menu e MenuItem*

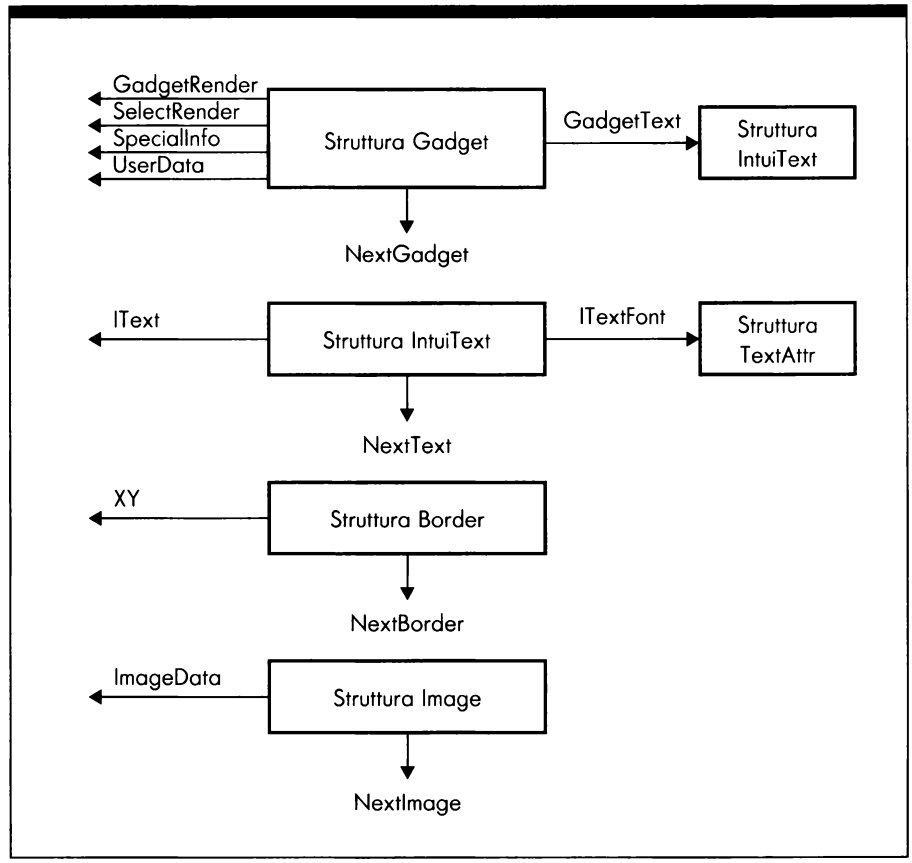


Figura 6.7: *Collegamenti delle strutture Gadget, IntuiText, Border e Image*

struttura dello stesso tipo attraverso il parametro puntatore OlderRequest.

Ciascuna struttura Requester contiene un insieme d'informazioni che definiscono una serie di strutture a concatenazione singola: Gadget, Border e IntuiText. Le strutture Gadget definiscono i gadget visualizzati quando il requester compare in una finestra; le strutture Border definiscono tutte le linee di contorno associate al requester; le strutture IntuiText definiscono il testo.

I collegamenti relativi alle strutture Menu e MenuItem

La Figura 6.6 descrive come progettare una barra menu per una finestra in ambiente Intuition. Una barra menu è composta da diversi menu, voci di menu e sotto-voci di menu. Innanzitutto, la figura mostra una struttura Menu, rappresentata da un rettangolo grande, e due rettangoli più piccoli che simboleggiano una lista concatenata di strutture MenuItem specificata nella struttura Menu. Il parametro FirstItem della struttura Menu punta alla prima

struttura MenuItem della lista. Inoltre ogni struttura Menu contiene un puntatore (denominato MenuName) a una stringa di testo a terminazione nulla che definisce il nome del menu. Infine ogni struttura Menu contiene un puntatore (denominato NextMenu) a un'altra struttura Menu relativa alla finestra considerata.

Il secondo grande rettangolo della figura rappresenta una struttura MenuItem, che può individuare a sua volta un'altra struttura MenuItem tramite il puntatore NextItem. Una serie di strutture MenuItem così collegate definiscono tutte le voci di un livello di menu. A ognuna di queste voci può essere associato un livello inferiore di menu, a sua volta composto da un certo numero di voci. Quindi, il parametro puntatore SubItem della struttura MenuItem di una voce punta alla prima struttura di una lista concatenata di strutture MenuItem impiegate per definire le voci del livello inferiore di menu.

I collegamenti relativi alle strutture Gadget, IntuiText, Border e Image

La Figura 6.7 mostra i collegamenti tra le strutture Gadget, IntuiText, Border e Image. Si noti che questi collegamenti sono molto semplici rispetto a quelli che interessano le altre strutture di Intuition.

Il primo rettangolo grande della Figura 6.7 mostra una struttura Gadget. In tale struttura sono presenti solo due collegamenti. Il primo è definito dal parametro puntatore GadgetText che individua una struttura IntuiText, la quale punta, a sua volta, a una stringa di testo a terminazione nulla che definisce il testo del gadget. Inoltre, ogni struttura Gadget contiene un puntatore (denominato NextGadget) a un'altra struttura Gadget che la collega a una lista di gadget.

Il successivo rettangolo grande della Figura 6.7 mostra una struttura IntuiText. In tale struttura sono presenti solo tre collegamenti. Il primo è definito dal parametro puntatore ITextFont, che punta a una struttura TextAttr. Il secondo è definito dal parametro puntatore IText che punta a una stringa di testo a terminazione nulla che definisce il testo della struttura IntuiText. Infine, ogni struttura IntuiText contiene un puntatore (denominato NextText) a un'altra struttura IntuiText che la collega a una lista di definizioni di testo.

Il successivo rettangolo grande della Figura 6.7 rappresenta una struttura Border. In tale struttura sono presenti solo due collegamenti. Il primo è definito dal parametro puntatore XY, che punta a un insieme di definizioni di vertici (valori x,y) per definire un contorno. Inoltre, ogni struttura Border contiene un puntatore (denominato NextBorder) a un'altra struttura Border, che la collega a una lista di definizioni di confini.

L'ultimo rettangolo grande della Figura 6.7 mostra una struttura Image che contiene il parametro puntatore ImageData, il quale punta a una definizione d'immagine in memoria; contiene anche un puntatore (denominato NextImage) a un'altra struttura Image che la collega a una lista d'immagini.

ActivateGadget

Sintassi di chiamata della funzione

```
success = ActivateGadget (gadget, window, requester)
DØ                        AØ      A1      A2
```

Scopo della funzione

Questa funzione attiva un gadget stringa già visualizzato, che si può trovare in una finestra o in un requester della finestra. Ciò risparmia all'utente di doverlo selezionare con il puntatore per introdurvi un testo. Se ha riscontrato le condizioni specificate, ActivateGadget restituisce il valore TRUE; altrimenti il valore FALSE.

Argomenti della funzione

gadget — Indirizzo della struttura Gadget che definisce il gadget stringa da attivare; la struttura Gadget considerata deve rappresentare un gadget stringa.

window — Indirizzo della struttura Window relativa alla finestra che contiene il gadget.

requester — Indirizzo della struttura Requester relativa al requester che contiene il gadget; va impostato a NULL se non si tratta di un gadget contenuto in un requester.

Discussione

ActivateGadget è una delle quattro funzioni che riguardano i gadget e le loro liste nel software sistema di Intuition.

ActivateGadget risparmia all'utente il fastidio di dover selezionare il gadget stringa con il puntatore per poter digitare un testo al suo interno.

Vi sono alcune condizioni da rispettare perché ActivateGadget operi appropriatamente:

- il gadget dev'essere un gadget stringa.

- L'argomento window deve puntare alla struttura Window relativa alla finestra che contiene il gadget puntato dall'argomento gadget.

- Se il gadget si trova in un requester, la finestra deve contenere il requester e l'indirizzo della relativa struttura Requester dev'essere anch'esso incluso nella chiamata alla funzione. L'argomento requester viene preso in considerazione soltanto se la struttura Gadget indicata ha il flag REQGADGET impostato.

- La finestra dev'essere attiva. Si può impostare il flag IDCMP ACTIVEWINDOW per essere certi che il task riceva da Intuition un messaggio ogni volta che la finestra viene attivata. Si può poi controllare il flag WINDOWACTIVE, nella struttura Window, per verificare che la finestra sia effettivamente attiva.

- La funzione attiva il gadget soltanto se l'utente non sta impiegando altri gadget, come i gadget di sistema per ridimensionare la finestra, trascinarla e così via.

- Se il gadget si trova in un requester, il requester dev'essere attivo. Si possono usare i flag IDCMP REQUEST e REQCLEAR per ricevere un messaggio IDCMP che consenta di rilevare quando un requester diventa attivo. Si può poi controllare il flag REQACTIVE nella struttura Requester per verificare che sia davvero attivo.

- Il pulsante destro del mouse non dev'essere premuto.

ActivateWindow

Sintassi di chiamata della funzione

ActivateWindow (window)
A0

Scopo della funzione

Questa funzione attiva la finestra indicata. Dovrebbe essere chiamata in seguito a particolari azioni dell'utente. La finestra non diventa attiva fino a che l'utente interagisce con i gadget, oppure svolge operazioni di ridimensionamento o spostamento di altre finestre e così via.

Argomenti della funzione

window Indirizzo della struttura Window relativa alla finestra che si vuole attivare.

Discussione

Le funzioni che riguardano le finestre nel software sistema di Intuition sono ActivateWindow e RefreshWindowFrame.

Ci sono diversi modi per rendere attiva una finestra. Primo, si può impostare il parametro ACTIVATE della struttura NewWindow, in modo che la finestra diventi attiva quando viene aperta sullo schermo per la prima volta. Secondo, l'utente può attivare una finestra selezionandola con il puntatore; non deve far altro che portare il puntatore al suo interno e premere il pulsante sinistro del mouse. Terzo, i task possono attivare le finestre di loro competenza tramite la funzione ActivateWindow.

Si noti che può trascorrere un certo tempo prima che la finestra diventi attiva. L'azione può infatti essere differita fino a quando l'utente non completa alcune azioni in corso di svolgimento, come la selezione di gadget o voci di menu di un'altra finestra. Il programma può controllare il flag WINDOWACTIVE della struttura Window, per rilevare quando la finestra diventa effettivamente attiva, impostando il flag IDCMP ACTIVATEWINDOW relativo alla finestra considerata. In tal modo, Intuition invia un messaggio IDCMP quando la finestra viene selezionata.

AddGadget

Sintassi di chiamata della funzione

```
gadgetPosition = AddGadget (window, gadget, position)
D0                          A0      A1      D0
```

Scopo della funzione

Questa funzione aggiunge il gadget specificato alla lista dei gadget relativa alla finestra indicata. AddGadget restituisce la posizione del gadget nella lista relativa alla finestra considerata.

Argomenti della funzione

window Indirizzo della struttura Window alla quale il gadget è associato.

gadget Indirizzo della nuova struttura Gadget.

position Posizione del nuovo gadget nella lista (si tratta di un numero intero). Zero corrisponde alla prima posizione della lista.

Discussione

Le funzioni di Intuition che gestiscono i gadget sono sei: AddGadget, OnGadget, OffGadget, RefreshGadgets, RemoveGadget e ModifyProp. Si vedano anche le spiegazioni relative a queste funzioni.

I gadget vengono aggiunti a una finestra tramite l'inserimento in una lista concatenata. L'argomento position nella chiamata alla funzione AddGadget determina dove viene collocato il nuovo gadget all'interno della lista concatenata. Se viene indicato il valore zero, il gadget diventa il primo elemento della lista; se viene indicato 1, il gadget viene inserito al secondo posto e così via. Se l'argomento position ha un valore superiore al numero dei gadget già presenti nella lista, il gadget viene comunque inserito al termine della lista. Un modo sicuro per essere certi che il gadget venga aggiunto in fondo è quello di usare per l'argomento position il valore –1.

Il sistema gestisce personalmente alcuni gadget nelle finestre e negli schermi. Tali gadget sono i primi nella lista e vengono detti "gadget di sistema". Si tratta di una misura di sicurezza: se il programmatore predispone gadget propri, i gadget di sistema appaiono comunque ai primi posti e risultano selezionabili per primi dall'utente. Se si aggiungono gadget all'inizio della lista, si possono ottenere sgradite interferenze.

AddGList

Sintassi di chiamata della funzione

```
actual_position = AddGList (window, gadget, position, number_gadgets,
D0                          A0      A1      D0        D1
                            requester)
                            A2
```

Scopo della funzione

Questa funzione aggiunge una lista di gadget alla lista dei gadget già esistenti di una finestra o di un requester. La nuova lista diventa quindi una sotto-lista all'interno di una lista preesistente, inserita nella posizione indicata dal task nell'argomento position. AddGList restituisce la posizione effettiva della nuova sotto-lista.

Argomenti della funzione

window Indirizzo della struttura Window relativa alla finestra destinata a ricevere la nuova lista di gadget.

gadget Indirizzo della struttura Gadget del primo gadget della sotto-lista da aggiungere.

position La posizione d'ordine in cui il primo gadget della sotto-lista dev'essere inserito nella lista dei gadget; lo zero colloca la sotto-lista al primo posto.

number_gadgets numero dei gadget da aggiungere, appartenenti alla sotto-lista; se dev'essere aggiunta l'intera sotto-lista, si indichi –1.

requester Indirizzo della struttura Requester che definisce il requester destinato a ricevere i gadget della sotto-lista.

Discussione

AddGList è una delle quattro funzioni che riguardano i gadget e le loro liste nel software sistema di Intuition.

Ogni finestra può essere associata a diversi gadget. In alcuni casi può essere utile aggiungere alcuni gadget alla lista di una finestra o di un requester. Questo è lo scopo della funzione AddGList.

Si osservi che:

- se il flag REQGADGET risulta impostato per i gadget, l'argomento requester deve puntare a una struttura Requester associata alla finestra. Qualsiasi errore su questo punto è destinato a far fallire AddGList.

- Il flag REQGADGET (contenuto nel parametro GadgetType) dev'essere impostato o azzerato in modo coerente per tutti i gadget della sotto-lista, che si tratti di gadget di tipo requester oppure no.

- Quando AddGList restituisce il controllo, il parametro NextGadget in una delle strutture Gadget (quella che si trova nella posizione number_gadgets della sotto-lista) conterrà il valore zero. Perciò, se si stanno aggiungendo tre gadget da una sotto-lista concatenata di cinque gadget, il legame tra il terzo e il quarto gadget della sotto-lista viene spezzato da AddGList; NextGadget viene azzerato nella terza struttura Gadget.

- AddGList aggiunge gadget provenienti dalla sotto-lista fino a che non raggiunge il numero di gadget indicato in number_gadgets, oppure fino a che non incontra nella sotto-lista l'ultima struttura Gadget, quella che evidentemente ha il parametro NextGadget azzerato.

AllocRemember

Sintassi di chiamata della funzione

```
memBlock = AllocRemember (rememberKey, size, flags)
D0                       A0            D0    D1
```

Scopo della funzione

Questa funzione alloca memoria per il task. Non solo alloca lo spazio di memoria richiesto, ma lo lega anche agli altri blocchi di memoria che il task ha eventualmente già allocato con precedenti chiamate ad AllocRemember. Tale legame consente al task di liberare in seguito tutti i blocchi di memoria in un sol colpo. Se l'allocazione ha successo, AllocRemember restituisce l'indirizzo del blocco di memoria richiesto. Nella lista che il task indica nell'argomento rememberKey viene anche inserito un nodo per definire il blocco.

Se invece l'allocazione fallisce, AllocRemember restituisce il valore NULL e la lista puntata da rememberKey, se esiste, non viene alterata.

Argomenti della funzione

rememberKey	Indirizzo di un puntatore a una struttura Remember; si deve impostare questo argomento a NULL, prima di chiamare AllocRemember per la prima volta.
size	Misura in byte della quantità di memoria da allocare.
flags	Argomento nel quale si impostano i flag per definire gli attributi della memoria, come accade con la funzione AllocMem.

Discussione

Ci sono due funzioni di Intuition che riguardano direttamente l'allocazione di memoria: AllocRemember e FreeRemember. I programmi possono allocare memoria utilizzando la funzione AllocMem della libreria Exec, ma in questo modo devono tener conto autonomamente di tutte le aree allocate, per poterle poi rilasciare con un ugual numero di chiamate a FreeMem. Le funzioni AllocRemember e FreeRemember servono anch'essere per allocare e rilasciare memoria, ma sollevano il task dal compito di "ricordare" gli indirizzi delle aree allocate e delle quantità di memoria. AllocRemember, infatti, oltre a chiamare AllocMem per allocare la memoria richiesta, alloca anche una struttura Remember, l'aggiorna con l'indirizzo e la grandezza dell'area allocata, e la inserisce nella lista indicata dal task. In questo modo, quando il task deve liberare le aree allocate, non fa altro che chiamare la funzione FreeRemember indicando l'indirizzo della lista semplice di strutture Remember.

La lista di strutture Remember si realizza creando una variabile puntatore definita come puntatore a un puntatore, che a sua volta individua in memoria una struttura di tipo Remember. Questa variabile va impostata a NULL prima di effettuare la prima chiamata ad AllocRemember.

Un esempio di questa situazione è il seguente:

```
struct Remember *RememberKey;
RememberKey = NULL;
ptr1 = AllocRemember (&RememberKey, BUFSIZE, MEMF_CHIP);
ptr2 = AllocRemember (&RememberKey, BUFSIZE, MEMF_CHIP);
...
FreeRemember (&RememberKey, TRUE);
```

AutoRequest

Sintassi di chiamata della funzione

```
response = AutoRequest (window, bodyText, positiveText, negativeText,
DØ                      AØ      A1        A2           A3
                        positiveFlags, negativeFlags, width, height)
                        DØ             D1             D2    D3)
```

Scopo della funzione

Questa funzione inserisce un requester nello schermo video. Attende poi una risposta dall'utente (o dal sistema). Se la risposta è positiva, AutoRequest restituisce TRUE, altrimenti FALSE.

Argomenti della funzione

window	Indirizzo della struttura Window.
bodyText	Indirizzo della struttura IntuiText che rappresenta il testo da inserire nel requester.
positiveText	Indirizzo della seconda struttura IntuiText.
negativeText	Indirizzo della terza struttura IntuiText.
positiveFlags	Bit di flag IDCMP.
negativeFlags	Bit di flag IDCMP.
width	Larghezza in pixel del requester.
height	Altezza in pixel del requester.

Discussione

Ci sono sette funzioni di Intuition che agiscono direttamente con i requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest,

InitRequester, Request e SetDMRequest. Tali funzioni forniscono tutti gli strumenti necessari per creare e manipolare requester in finestre e schermi. Si vedano anche le spiegazioni relative a queste funzioni.

AutoRequest produce un requester molto semplice sullo schermo video e chiede all'utente di scegliere solo fra una risposta affermativa e una risposta negativa. L'argomento PositiveText è un puntatore a una struttura IntuiText che contiene un puntatore al testo che dev'essere associato alle risposte dell'utente "Sì", "Vero" o "Riprova". L'argomento NegativeText è un puntatore a una struttura IntuiText che contiene un puntatore al testo che dev'essere associato alle risposte dell'utente "No", "Falso" o "Annulla".

Il testo per le risposte positive viene automaticamente disposto nell'angolo inferiore sinistro del requester. Il testo per le risposte negative viene automaticamente disposto nell'angolo inferiore destro. I bit di flag IDCMP consentono di far intervenire eventi esterni (negativi o positivi) per soddisfare il requester. Per esempio, l'introduzione di un disco in un disk drive può costituire un evento esterno positivo che soddisfa il requester.

Quando si chiama la funzione AutoRequest, Intuition presenta il requester e attende una risposta dall'utente.

La funzione AutoRequest chiama la funzione BuildSysRequest per creare un requester sullo schermo. Gli argomenti della funzione AutoRequest vengono automaticamente passati alla funzione BuildSysRequest. Se questa restituisce un puntatore a una struttura Window, la struttura Window avrà le sue message port IDCMP (user port e window port) e i suoi flag impostati in accordo con gli argomenti della funzione AutoRequest; altrimenti BuildSysRequest restituisce TRUE o FALSE alla funzione AutoRequest.

BeginRefresh

Sintassi di chiamata della funzione

BeginRefresh (window)
AØ

Scopo della funzione

Questa funzione prepara una finestra a refresh semplice per un'operazione ottimizzata di refresh. Imposta le condizioni interne di Intuition e poi chiama la funzione BeginUpdate della libreria Layers. Questa funzione non restituisce alcun valore.

Argomenti della funzione

window Indirizzo della struttura Window la cui finestra deve subire l'operazione di refresh.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Lo scopo della funzione BeginRefresh è quello di assicurare che la ricostruzione (refresh) della finestra venga attuata soltanto per le parti di finestra che devono effettivamente essere ridisegnate. La funzione WindowRefresh impiega i rettangoli di delimitazione del layer che definisce la finestra per determinare quali sono le parti della bitmap da ricostruire. Chiama la funzione BeginUpdate della libreria Layers per scambiare gli appropriati puntatori nelle strutture DamageList e ClipRect.

Intuition fornisce tre diversi modi per effettuare il refresh delle finestre. Si tratta dei metodi refresh semplice, refresh avanzato e refresh superbitmap. I concetti che interessano questi tre diversi metodi di ricostruzione sono strettamente legati ai concetti di layer e di rettangolo di delimitazione, trattati nei capitoli 2 e 5.

La finestra condivide sempre parte della bitmap di schermo. Tuttavia, nello spostare, ridimensionare o eseguire altre operazioni sulla finestra, alcune sue parti possono venir nascoste. I tre metodi di refresh differiscono tra loro soltanto per il modo in cui gestiscono le informazioni di bitmap per le aree nascoste. Ovviamente le informazioni di bitmap per le aree nascoste non fanno parte della bitmap di schermo (altrimenti sarebbero visibili), quindi devono trovarsi altrove nel sistema.

Con il metodo a refresh semplice, è il programma che deve occuparsi di ridisegnare le parti nascoste della finestra, dal momento che non vengono salvate in alcun buffer di memoria. Per farlo, il programma deve adoperare le funzioni della libreria Graphics (AreaDraw, AreaMove, AreaEnd...), limitando eventualmente il disegno alle sole parti tornate visibili: questa operazione corrisponde a ricostruire le informazioni contenute nelle aree oscurate che tornano alla luce. Se invece l'utente trascina semplicemente la finestra attraverso lo schermo (senza oscurarne alcuna parte) è Intuition che si occupa di tenerne aggiornata la presentazione, senza che sia necessario il supporto del programma.

Il metodo di refresh semplice condotto dal task è più lento di quello avanzato (smart-refresh) e di quello a superbitmap, dato che le operazioni di refresh semplice richiedono l'esecuzioni di funzioni di disegno. Tuttavia, poiché non viene salvato alcun buffer per le aree oscurate, questo metodo

risulta più efficiente dal punto di vista del consumo di memoria; per la presentazione delle finestre viene infatti utilizzata la sola memoria video di schermo; le finestre di questo tipo non possiedono cioè bitmap proprie.

Con il metodo a refresh avanzato, Intuition mantiene tutte le informazioni che riguardano le aree nascoste della finestra in un gruppo di buffer RAM. Quando poi un'area nascosta ritorna visibile, è Intuition che preleva le relative informazioni di bitmap e le trasferisce nella bitmap di schermo: quest'operazione corrisponde a ripristinare le informazioni contenute nelle aree oscurate che tornano alla luce. Il metodo a refresh avanzato adopera la memoria video di schermo per definire le parti in vista della finestra, e alcuni buffer addizionali per le parti oscurate.

Per predisporre una finestra a refresh avanzato basta impostare il flag SMART_REFRESH nel parametro Flags della struttura NewWindow. Ciò informa il sistema che deve eseguire la ricostruzione della finestra in accordo con il metodo di refresh avanzato. I buffer adoperati per le parti oscurate vengono assegnati automaticamente dal sistema. È ovvio che il metodo a refresh avanzato impiega più memoria, ma è in grado di ripristinare la presentazione delle finestre molto più velocemente del metodo a refresh semplice.

Con il metodo a refresh superbitmap, in un'unica area buffer viene mantenuta una copia dell'intera bitmap della finestra, che può essere visibile sullo schermo tutta, in parte, o per nulla. Quindi, per questo tipo di finestra, accanto alla rappresentazione sullo schermo video ci sarà sempre in memoria una copia completa dell'intera bitmap della finestra. Il termine superbitmap deriva dalla misura di quest'ultima bitmap: una superbitmap è infatti generalmente più grande della finestra, cioè della parte di bitmap di schermo occupata dalla finestra. Il programma non deve mai preoccuparsi di ricostruire sezioni nascoste di una finestra superbitmap. Una volta che una finestra è stata indicata come superbitmap, è Intuition che provvede automaticamente a ricostruirla adoperando le informazioni contenute nell'area RAM superbitmap.

I passi richiesti per preparare una finestra superbitmap sono i seguenti:

1. Si alloca lo spazio RAM per la superbitmap (usando per esempio la funzione AllocRaster della libreria Graphics). Si noti che tale area di memoria può essere più larga della sotto-sezione della bitmap di schermo costituita dalla finestra.

2. Si crea una struttura BitMap, nella quale si inseriscono gli indirizzi dei bitplane che costituiscono la superbitmap della finestra. Per impostare il contenuto della struttura BitMap si può impiegare la funzione InitBitMap della libreria Graphics.

3. Si imposta il flag SUPER_BITMAP nel campo Flags della struttura NewWindow e si assegna al parametro BitMap l'indirizzo della bitmap appena allocata.

4. Si chiama la funzione OpenWindow.

BuildSysRequest

Sintassi di chiamata della funzione

```
reqwindow = BuildSysRequest (window, bodyText, positiveText,
D0                           A0      A1        A2
                             negativeText, IDCMPFlags, width, height)
                             A3            D0          D2     D3
```

Scopo della funzione

Questa funzione crea un requester nella finestra indicata dall'argomento window. Se tutto va a buon fine, BuildSysRequest restituisce l'indirizzo della struttura Window associata al requester.

Argomenti della funzione

window	Indirizzo della struttura Window.
bodyText	Indirizzo della struttura IntuiText che costituisce il testo da inserire nel requester.
positiveText	Indirizzo della seconda struttura IntuiText, che definisce il testo del gadget per la scelta positiva.
negativeText	Indirizzo della terza struttura IntuiText, che definisce il testo del gadget per la scelta negativa.
IDCMPFlags	Flag IDCMP da impiegare per chiedere a Intuition di associare alcuni eventi standard al requester.
width	Larghezza in pixel del requester.
height	Altezza in pixel del requester.

Discussione

Ci sono sette funzioni di Intuition che agiscono direttamente con i requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest, InitRequester, Request e SetDMRequest. Si vedano anche le spiegazioni relative a queste funzioni.

Questa funzione costituisce automaticamente un requester secondo quanto indicato negli argomenti. Se il sistema riesce a costruire il requester, restituisce un puntatore alla nuova finestra in cui il requester è contenuto.

Se le condizioni di sistema sono tali che il requester non può essere presentato nella finestra, BuildSysRequest chiama la funzione DisplayAlert. Il testo per il requester viene passato alla funzione DisplayAlert. Quello che si verifica in seguito dipende dalle azioni dell'utente. In particolare, il valore (TRUE o FALSE) passato alla funzione DisplayAlert risulta TRUE se l'utente ha premuto il pulsante sinistro del mouse e FALSE se l'utente ha premuto il pulsante destro.

Quando si chiama la funzione BuildSysRequest, il task deve entrare in attesa degli eventi che Intuition gli invia in conseguenza di quanto richiesto e delle interazioni che l'utente compie con il requester. In questo modo, il task gestisce direttamente le risposte dell'utente, al contrario di quanto accade con AutoRequest, che fa tutto da sola e restituisce soltanto un valore booleano quando l'utente sceglie tra le due possibilità che gli vengono offerte. Proprio per questa particolare flessibilità d'impiego, la funzione BuildSysRequest non si preoccupa di rimuovere il requester dallo schermo. È il task che, a secondo degli eventi, deve a un certo momento rimuovere il requester chiamando la funzione FreeSysRequest.

ClearDMRequest

Sintassi di chiamata della funzione

```
notinuse = ClearDMRequest (window)
D0                         A0
```

Scopo della funzione

Questa funzione cancella da una finestra lo speciale requester associato a una doppia pressione del pulsante destro del mouse, quello che attiva la barra menu. Se questo speciale requester non è in uso, ClearDMRequest azzera il puntatore DMRequester nella struttura Window e restituisce il valore booleano

TRUE. Se il requester è invece in uso, il puntatore DMRequester nella struttura Window non viene alterato e la funzione ClearDMRequest restituisce il valore booleano FALSE.

Argomenti della funzione

window Indirizzo della struttura Window la cui finestra dev'essere privata dello speciale requester che appare premendo due volte in rapida successione il pulsante destro del mouse.

Discussione

Ci sono sette funzioni di Intuition che agiscono direttamente con i requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest, InitRequester, Request e SetDMRequest. Si vedano anche le spiegazioni relative a queste funzioni.

Un requester a doppio menu viene presentato soltanto quando l'utente preme due volte in rapida successione il pulsante destro del mouse; come conseguenza, tutti gli input verso quella finestra vengono bloccati. Si noti che è lo stesso tipo di blocco che si verifica quando Intuition o il programma visualizzano un requester senza intervento dell'utente.

Se richiesto tramite il flag REQSET, viene inoltre inserito nel flusso di input della finestra un messaggio che indica l'apparizione del requester. Se si vuole impedire all'utente di richiamare un requester a doppio menu in una determinata finestra, è possibile slegare tale requester dalla finestra considerata attraverso la funzione ClearDMRequest.

ClearMenuStrip

Sintassi di chiamata della funzione

ClearMenuStrip (window)
A0

Scopo della funzione

Questa funzione cancella una barra menu da una determinata finestra, e non restituisce alcun valore.

Argomenti della funzione

window Indirizzo della struttura Window.

Discussione

Ci sono cinque funzioni che agiscono direttamente con i menu: ClearMenuStrip, ItemAddress, OnMenu, OffMenu e SetMenuStrip. Queste funzioni consentono di creare e manipolare menu e voci di menu in finestre e schermi. Si vedano anche le spiegazioni a esse relative.

Una barra menu consiste in una lista concatenata di strutture Menu, da ognuna delle quali si possono diramare liste di strutture MenuItem. I menu sono sempre associati a finestre. Può capitare di voler disattivare il menu associato a una particolare finestra, e per farlo occorre servirsi della funzione ClearMenuStrip.

Una volta che una finestra è stata privata del suo menu, se l'utente la seleziona e preme il pulsante destro del mouse nessun menu è più disponibile nella barra titolo dello schermo. Si può comunque chiamare la funzione SetMenuStrip per agganciare alla finestra un'altra barra menu.

Il flusso degli eventi per le operazioni di menu è il seguente:

1. Chiamare la funzione OpenWindow per aprire una finestra.

2. Chiamare le funzioni SetMenuStrip e ClearMenuStrip rispettivamente per associare una barra menu alla finestra e per poi dissociarla.

3. Chiamare la funzione CloseWindow per chiudere le finestra.

La rimozione della barra menu prima di procedere alla chiusura della finestra evita i problemi che possono verificarsi se l'utente sta accedendo a un menu quando il programma chiude la finestra.

ClearPointer

Sintassi di chiamata della funzione

ClearPointer (window)
AØ

Scopo della funzione

Questa funzione rimuove l'immagine definita dal task per il puntatore del mouse. Ogni volta che, in seguito, la finestra specificata diventa attiva, in essa il puntatore appare con l'immagine di default. Se la finestra è attiva quando viene eseguita la funzione ClearPointer, l'immagine personalizzata del puntatore viene immediatamente cambiata e diventa l'immagine di default di Intuition. ClearPointer non restituisce valori.

Argomenti della funzione

window Indirizzo della struttura Window la cui finestra è destinata a perdere la propria definizione d'immagine per il puntatore.

Discussione

Ci sono due funzioni della libreria Intuition che agiscono direttamente con le immagini del puntatore: SetPointer e ClearPointer. Tali funzioni consentono rispettivamente di predisporre un'immagine personalizzata per il puntatore di schermo quando la finestra è attiva, e di rimuovere tale immagine ristabilendo quella di default. In tal modo è possibile creare immagini personalizzate del puntatore ed evitare di usare l'immagine del puntatore prevista per default da Intuition.

Il puntatore di default di Intuition è lo sprite hardware 0 ed è disegnato come se la sorgente di luce provenisse dall'angolo superiore destro dello schermo video. Si dovrebbe considerare la stessa prospettiva quando si progetta un proprio disegno per il puntatore; esso dovrà apparire più luminoso sul suo lato in alto a destra. Ecco le assegnazioni di colore usate con i dati per lo sprite puntatore di Intuition:

colore 16	Trasparente (registro di colore hardware numero 16)
colore 17	Colore di media intensità (registro di colore hardware numero 17)
colore 18	Colore di bassa intensità (registro di colore hardware numero 18)
colore 19	Colore di alta intensità (registro di colore hardware numero 19)

Per risultare coerente con il puntatore di default in ambiente Intuition, il puntatore personalizzato dovrebbe essere incorniciato con il colore 18. Si possono impostare i colori dell'immagine del puntatore chiamando la funzione SetRGB4 della libreria Graphics.

Per le informazioni relative alla costruzione e all'uso di un puntatore personalizzato nei programmi, si veda la spiegazione della funzione SetPointer.

CloseScreen

Sintassi di chiamata della funzione

CloseScreen (screen)
A0

Scopo della funzione

Questa funzione chiude uno schermo personalizzato di Intuition e libera tutta la memoria da esso occupata. CloseScreen non restituisce valori.

Argomenti della funzione

screen	Indirizzo della struttura Screen il cui schermo dev'essere chiuso.

Discussione

Ci sono sette funzioni di Intuition che agiscono direttamente con gli schermi video di Intuition: OpenScreen, CloseScreen, MoveScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront.

La funzione CloseScreen chiude lo schermo indicato e libera la memoria occupata dalla struttura Screen. Slega anche la struttura Screen dalla struttura ViewPort associata. CloseScreen chiude lo schermo anche se in esso vi sono finestre aperte (ma non provvede a chiuderle). Se lo schermo che si è chiuso è l'ultimo, riappare automaticamente lo schermo Workbench.

CloseWindow

Sintassi di chiamata della funzione

CloseWindow (window)
AØ

Scopo della funzione

Questa funzione chiude una finestra di Intuition, liberando tutta la memoria che le era stata assegnata. Se la finestra appare in uno schermo standard di sistema (con l'eccezione dello schermo Workbench) e si tratta dell'ultima finestra presente nello schermo, chiudendola si provoca anche la chiusura dello schermo in cui la finestra è contenuta. CloseWindow non restituisce valori.

Argomenti della funzione

window	Indirizzo della struttura Window che definisce la finestra da chiudere.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, Mo-

veWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Prima di chiamare CloseWindow, il programma deve accertarsi che siano stati elaborati tutti i messaggi presenti nella coda IDCMP della finestra, cioè nella coda alla user port. Se infatti la finestra viene chiusa, gli eventuali messaggi ancora presenti vanno persi e la memoria corrispondente viene liberata.

Allo stesso modo, se è stata predisposta una barra menu con la funzione SetMenuStrip, si dovrebbe procedere alla sua rimozione prima di chiudere la finestra. A questo scopo si può utilizzare ClearMenuStrip. La funzione CloseWindow, prima di andare in esecuzione, non effettua nessun controllo per vedere se i menu della finestra sono in uso. Se l'utente sta usando i menu nel momento in cui viene chiamata CloseWindow, il sistema va in crash.

Se la finestra è l'ultima che si chiude in uno schermo standard, anche lo schermo viene chiuso (a meno che non si tratti dello schermo Workbench). La stessa cosa non accade se lo schermo è personalizzato: in questo caso, se si vuole chiudere anche lo schermo è necessario chiamare CloseScreen dopo la chiamata a CloseWindow.

CloseWorkBench

Sintassi di chiamata della funzione

```
close = CloseWorkBench ()
D0
```

Scopo della funzione

Questa funzione tenta di chiudere lo schermo Workbench e restituisce il valore booleano TRUE se vi riesce, FALSE se non vi riesce. La funzione controlla se sullo schermo Workbench risultano aperte una o più finestre. Se risulta presente anche una sola finestra che non sia una finestra disk o drawer aperta dal programma *Workbench*, la funzione restituisce il valore FALSE e non chiude lo schermo. Se invece CloseWorkBench non trova finestre, provvede innanzitutto ad azzerare qualsiasi buffer speciale e poi chiude lo schermo restituendo il valore TRUE. Sullo schermo video possono apparire altri schermi, ma l'utente non ha più la possibilità di far apparire lo schermo Workbench.

Si noti che questa funzione nulla ha a che vedere con il programma *Workbench*, cioè con quell'applicazione, residente in ROM e attivabile con il

comando LOADWB da CLI, che permette di gestire i dischi, le directory e i file attraverso un sistema di icone. Infatti CloseWorkBench agisce comunque, che il *Workbench* sia in esecuzione o meno, e non ne sospende l'esecuzione.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Nella libreria Intuition ci sono quattro funzioni che consentono di agire direttamente con lo schermo Workbench: OpenWorkBench, CloseWorkBench, WBenchToBack e WBenchToFront. Si vedano anche le spiegazioni relative a queste funzioni.

Finora lo schermo Workbench è il solo schermo standard nel sistema Intuition. Potranno eventualmente esserci altri schermi standard in futuro. Lo schermo Workbench è il solo schermo, oltre a quelli personalizzati, che si può esplicitamente chiudere. Inoltre lo schermo Workbench, come gli schermi personalizzati, non si chiude quando il programmatore o l'utente chiudono tutte le sue finestre. Lo schermo Workbench si riapre automaticamente quando tutti gli altri schermi sono chiusi.

Se il nostro programma applicativo richiede molta memoria per funzionare, è possibile impiegare la funzione CloseWorkBench per liberare la memoria utilizzata dallo schermo Workbench. Quando si segue questa strada, si deve ricordare di chiamare nuovamente OpenWorkBench prima di abbandonare il programma. Chiamare la funzione OpenWorkBench prima di uscire da un programma, in ogni caso, è un'operazione che per sicurezza dovrebbe essere effettuata sempre.

CurrentTime

Sintassi di chiamata della funzione

```
CurrentTime (seconds, micros)
            A0       A1
```

Scopo della funzione

Questa funzione rileva i valori del tempo di sistema, in modo che il task li possa esaminare e impiegare; i valori dei secondi e dei microsecondi vengono memorizzati dalla funzione nelle locazioni di memoria indicate dal task come argomenti.

Argomenti della funzione

seconds Indirizzo della variabile long word destinata a ricevere il valore dei secondi.

micros Indirizzo della variabile long word destinata a ricevere il valore dei microsecondi.

Discussione

CurrentTime è la sola funzione di Intuition che riguarda direttamente le variabili di temporizzazione del sistema, e serve per rilevare i valori delle variabili di clock. Questi valori possono poi essere utilizzati in qualsiasi programma che voglia sincronizzare una sequenza di eventi.

Il programma, una volta che la funzione ha restituito il controllo, può riferirsi ai valori contenuti nelle variabili seconds e micros per i valori dei tempi.

Si noti che entrambi i valori occupano quattro byte. Ciò consente al valore dei secondi di giungere fino a $2^{32} - 1$, il che equivale a circa 136 anni. Al raggiungimento di questo valore si ritorna a zero.

CurrentTime restituisce i valori del tempo di sistema, che non è molto accurato e nemmeno preciso in termini di unità di tempo minima rilevabile, dal momento che viene aggiornato in media ogni cinquantesimo di secondo.

DisplayAlert

Sintassi di chiamata della funzione

```
alert = DisplayAlert (alertNumber, string, height)
D0                   D0            A0     D1
```

Scopo della funzione

Questa funzione crea uno specifico alert (messaggio d'errore) sullo schermo e restituisce un valore booleano (TRUE o FALSE) nella variabile alert per indicare quale scelta è stata operata dall'utente. Se si tratta di un alert di tipo DEADEND_ALERT viene sempre restituito FALSE. Se si tratta di un alert di tipo RECOVERY_ALERT il valore restituito sarà TRUE qualora, in risposta al messaggio di alert, l'utente abbia premuto il pulsante sinistro del mouse e FALSE qualora l'utente abbia premuto il pulsante destro.

Argomenti della funzione

alertNumber Numero del messaggio di alert; questo numero è la composizione di una serie di valori che attivano particolari bit dell'argomento. In particolare, i bit ALERT_TYPE devono essere impostati a RECOVERY_ALERT o DEADEND_ALERT. I rimanenti bit vengono ignorati.

string Indirizzo della stringa per il messaggio di alert.

height Numero minimo delle linee video richieste per l'alert.

Discussione

DisplayAlert è una funzione della libreria Intuition che non ricade in alcuna categoria specifica. In ambiente Intuition ci sono due tipi di alert: quelli di sistema e quelli richiesti dai programmi applicativi. Gli alert di sistema vengono definiti e gestiti interamente da Intuition; gli alert dei programmi vengono gestiti dalla funzione DisplayAlert. Entrambi presentano informazioni assolutamente essenziali; dovrebbero quindi essere usati soltanto per situazioni in cui l'utente deve intraprendere qualche azione immediata per prevenire lo sviluppo di situazioni critiche. Un imminente crash di sistema è un esempio di situazione critica; un altro esempio è costituito da un errore fatale in un programma.

Entrambi i tipi di alert presentano uno sfondo nero, un bordo rosso lampeggiante e una risoluzione orizzontale di 640 pixel; l'altezza è quella specificata nell'argomento height. L'alert appare sempre nella parte alta del video, spingendo verso il basso tutto ciò che occorre per essere completamente visualizzato. Se tuttavia il problema che ha prodotto l'alert è un errore fatale, il sistema si trova nell'imminenza di un abbandono disastroso. In tal caso sul video rimane solamente l'alert.

Ci sono due tipi di alert di sistema e di programma: RECOVERY_ALERT

e DEADEND_ALERT. RECOVERY_ALERT genera un contorno rosso lampeggiante e mostra il testo dell'alert. Se l'utente, in risposta al testo in questione, preme il pulsante sinistro del mouse, la funzione DisplayAlert restituisce il valore TRUE. DEADEND_ALERT mostra invece il testo dell'alert e restituisce immediatamente FALSE. Il solo modo per uscire da un alert DEADEND_ALERT è quello di riavviare il sistema.

Nell'argomento string della funzione DisplayAlert il task deve indicare l'indirizzo della stringa del messaggio di alert, costituita da una o più sotto-stringhe. Il primo componente di ciascuna sotto-stringa è formato da una coordinata x a 16 bit e da una coordinata y a 8 bit. Ciò definisce la posizione di schermo in cui si vuole fare apparire il testo. La coordinata y definisce la posizione in ordinata della linea base del testo. Il secondo componente è il testo stesso. Si deve trattare di una stringa a terminazione nulla. L'ultimo componente è il byte di continuazione. Se vale zero significa che la sotto-stringa che lo contiene è l'ultima del messaggio di alert. Se è diverso da zero significa che c'è un'altra stringa che continua il messaggio.

È opportuno evitare l'uso di alert, e riservarli alle situazioni estreme. Consigliamo piuttosto di ricorrere a requester con messaggi di avvertimento. I requester sono meno allarmanti per l'utente e possono gestire diversi tipi di risposte.

DisplayBeep

Sintassi di chiamata della funzione

DisplayBeep (screen)
AØ

Scopo della funzione

Questa funzione fa lampeggiare il colore di fondo dello schermo indicato. Se l'argomento screen è NULL, lampeggerà ogni schermo del sistema. DisplayBeep non restituisce valori.

Argomenti della funzione

screen	Indirizzo della struttura Screen associata allo schermo che si vuole far lampeggiare; se questo argomento è NULL, lampeggeranno tutti gli schermi.

Discussione

DisplayBeep è una funzione della libreria Intuition che non ricade in alcuna categoria specifica. L'Amiga non dispone di un campanello o di un altoparlante interno. Per questo, i problemi incontrati dal sistema o dal programma applicativo non possono essere portati all'attenzione dell'utente attraverso un suono, se non tramite l'altoparlante esterno collegato ai canali audio (generalmente contenuto nel monitor). È questo il motivo dell'esistenza della funzione DisplayBeep.

L'impiego di questa funzione dovrebbe essere riservato a situazioni non così gravi da richiedere un alert. Per esempio, Intuition impiega la funzione DisplayBeep quando l'utente introduce da tastiera caratteri non numerici in un gadget che accetta solo un input di tipo numerico.

Si noti che è possibile fare lampeggiare tutti gli schermi, compresi gli schermi personalizzati creati dai task e dai programmi che risiedono nel sistema. Il modo più semplice per ottenere questo risultato è quello di usare un argomento NULL nella chiamata alla funzione DisplayBeep. È tuttavia raro che si presenti una buona ragione per far lampeggiare tutti gli schermi di Intuition.

DoubleClick

Sintassi di chiamata della funzione

```
gooddoubleclick = DoubleClick (startSeconds, startMicros,
D0                             D0            D1
                               currentSeconds, currentMicros)
                               D2              D3
```

Scopo della funzione

Questa funzione confronta l'intervallo di tempo indicato negli argomenti con il tempo di double-click (doppia pressione) indicato nella struttura Preferences. Se l'intervallo indicato dal task è compreso nel tempo di double-click indicato dalla struttura Preferences, la funzione restituisce il valore booleano TRUE, altrimenti il valore FALSE.

Il task indica l'intervallo di tempo sotto forma di valore iniziale (startSeconds, startMicros) e valore finale (currentSeconds, currentMicros), entrambi espressi in secondi e microsecondi; è la funzione a eseguire la differenza fra questi due valori per ricavare l'intervallo di tempo.

Il tempo di double-click definisce l'intervallo di tempo massimo che può

trascorrere fra una pressione del pulsante sinistro del mouse e la successiva perché le due pressioni possano essere considerate come un unico evento, cioè come un particolare tipo di selezione.

Argomenti della funzione

startSeconds Valore in secondi dell'istante assoluto in corrispondenza del quale l'utente ha premuto il pulsante sinistro del mouse.

startMicros Valore in microsecondi dell'istante assoluto in corrispondenza del quale l'utente ha premuto il pulsante sinistro del mouse.

currentSeconds Valore in secondi dell'istante assoluto in corrispondenza del quale l'utente ha premuto il pulsante sinistro del mouse per la seconda volta.

currentMicros Valore in microsecondi dell'istante assoluto in corrispondenza del quale l'utente ha premuto il pulsante sinistro del mouse per la seconda volta.

Discussione

DoubleClick è una delle funzioni della libreria Intuition che non ricade in alcuna specifica categoria. Il programma dev'essere in grado di decidere quando due pressioni del pulsante del mouse eseguite in rapida successione corrispondano a due eventi distinti e quando corrispondano a un unico evento di double-click. Questo è il compito della funzione DoubleClick: esaminare il tempo indicato dal task e confrontarlo con i valori di tempo di riferimento per la doppia pressione già predisposti nella struttura Preferences.

Ci sono diverse occasioni in cui l'utente preme due volte in successione il pulsante sinistro del mouse; si può fare in modo che se questo accade quando il puntatore si trova sopra l'icona di un file, quel file venga mandato in esecuzione.

Il pulsante destro del mouse viene per lo più impiegato per interagire con i menu della finestra attiva. Premendolo una volta soltanto appare la barra menu, e se si indirizza il puntatore sulle voci tenendo premuto il pulsante si procede alla visita del menu. La doppia pressione del pulsante destro genera invece l'apparizione del requester di double-click.

DrawBorder

Sintassi di chiamata della funzione

```
DrawBorder (rastPort, border, leftOffset, topOffset)
            AØ        A1      DØ          D1
```

Scopo della funzione

Questa funzione disegna un insieme di linee (contorni) nella bitmap specificata. I contorni possono essere un insieme qualsiasi di linee disegnate ovunque nello schermo video. Le linee sono definite da un insieme di vertici x,y puntati da una struttura Border. Se parte di una definizione di linea ricade al di là dei limiti della bitmap, la linea viene limitata al bordo della bitmap. DrawBorder non restituisce valori.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
border	Indirizzo della struttura Border.
leftOffset	Offset da aggiungere alla coordinata x di ciascuna linea prima che venga disegnata.
topOffset	Offset da aggiungere alla coordinata y di ciascuna linea prima che venga disegnata.

Discussione

La funzione DrawBorder consente di aggiungere qualsiasi contorno a un elemento video contenitore (una finestra, un requester o un gadget). La funzione DrawBorder agisce con la struttura Border.

La struttura Border

La struttura Border include diversi parametri:

- i parametri LeftEdge e TopEdge contengono la posizione pixel di partenza della linea (cioè del contorno) rispetto all'angolo superiore sinistro dell'elemento video contenitore in cui verrà disegnata. Queste posizioni sono misurate in accordo con la risoluzione orizzontale e verticale dell'elemento video contenitore. A esse vengono aggiunti dalla funzione DrawBorder i valori indicati come leftOffset e topOffset. Ciò consente di riutilizzare la medesima struttura Border per produrre bordi identici ma in posizioni diverse.

- I parametri FrontPen, BackPen e DrawMode sono usati per definire il colore della linea. FrontPen è il valore di un registro di colore. BackPen per il momento non è usato. DrawMode può essere JAM1 o COMPLEMENT. Il modo di disegno JAM1 utilizza il colore FrontPen per disegnare la linea e non apporta cambiamenti allo sfondo. Il modo di disegno COMPLEMENT disegna la linea complementando lo stato dei pixel.

- Il parametro NextBorder punta a un'altra eventuale struttura Border. Tramite questo puntatore si possono legare insieme diverse strutture Border. Ciascuna di esse può definire un insieme di linee. DrawBorder termina la sua esecuzione quando incontra una struttura Border che ha questo parametro impostato a zero.

- Il parametro XY punta a un array di word costituito da coppie di coordinate x,y, associato alla struttura Border. Ad altre strutture Border collegate possono essere associati ulteriori array XY.

- Il parametro Count contiene il numero di coppie di valori x,y presenti nell'array XY associato a questa struttura Border.

DrawImage

Sintassi di chiamata della funzione

DrawImage (rastPort, image, leftOffset, topOffset)
A0 A1 D0 D1

Scopo della funzione

Questa funzione disegna nella bitmap l'immagine specificata, provvedendo a troncarla se esce dai confini della bitmap. DrawImage non restituisce alcun valore.

Argomenti della funzione

rastPort	Indirizzo della struttura RastPort di controllo.
image	Indirizzo della struttura Image che definisce i pixel dell'immagine.
leftOffset	Offset da aggiungere alla coordinata x dell'immagine prima che questa venga disegnata.
topOffset	Offset da aggiungere alla coordinata y dell'immagine prima che questa venga disegnata.

Discussione

La funzione DrawImage consente d'inserire un'immagine in un elemento grafico contenitore (una finestra, un requester o un gadget). Questa funzione agisce con la struttura Image.

La struttura Image

La struttura Image contiene diversi parametri:

- le variabili LeftEdge e TopEdge contengono la posizione pixel iniziale dell'immagine rispetto all'angolo superiore sinistro dell'elemento video contenitore. Tali posizioni sono misurate in accordo con la risoluzione orizzontale e verticale dell'elemento video contenitore. A esse vengono aggiunti dalla funzione DrawImage i valori indicati come leftOffset e topOffset. Ciò consente di riutilizzare la medesima struttura Image per produrre immagini identiche ma in posizioni differenti.

- Le variabili Width e Height contengono la larghezza e l'altezza in pixel dell'immagine.

- La variabile Depth contiene il numero dei bitplane richiesti per definire l'immagine.

- Il puntatore ImageData punta a una struttura ImageData che definisce i pixel dell'immagine. La struttura ImageData contiene diverse word che definiscono il colore di ciascun pixel nella definizione dell'immagine. Per le informazioni relative alla costruzione della struttura ImageData si veda la spiegazione della struttura ImageData riguardante i bob (a pagina 403), nella descrizione della funzione AddVSprite.

- Il parametro PlanePick dice al sistema quali sono i bitplane della bitmap dell'elemento video contenitore destinati a ricevere i bit che definiscono l'immagine. Se il parametro PlanePick contiene tutti bit impostati a 1, ogni bitplane relativo alla bitmap dell'elemento contenitore riceverà i bit provenienti dalla bitmap dell'immagine. Se qualche bit del parametro PlanePick è invece impostato a 0, il bitplane della bitmap dell'elemento contenitore che corrisponde a quel bit non riceverà i bit provenienti dalla bitmap dell'immagine.

- Il parametro PlaneOnOff dice al sistema come comportarsi con i bitplane dell'elemento contenitore che non sono interessati all'inserimento dei dati relativi all'immagine. Questo parametro può spegnere la visualizzazione dei pixel in uno dei bitplane. Se, per esempio, si ha un elemento contenitore con una bitmap a due bitplane e si imposta PlaneOnOff a 00, i bit nel bitplane escluso dal parametro PlanePick verranno impostati a zero.

I parametri PlanePick e PlaneOnOff richiedono un certo approfondimento per essere compresi. Si veda anche il capitolo 3.

EndRefresh

Sintassi di chiamata della funzione

EndRefresh (window, complete)
A0 D0

Scopo della funzione

Questa funzione segna la fine della ricostruzione di una finestra a refresh semplice, iniziata con una chiamata alla funzione BeginRefresh. Completa il processo chiamando la funzione EndUpdate della libreria Layers per invertire lo scambio realizzato da BeginUpdate e per consentire alla struttura ClipRect di riacquistare ancora una volta il controllo del processo di disegno. Questa funzione non restituisce alcun valore.

Argomenti della funzione

window Indirizzo della struttura Window sottoposta a refresh semplice "ottimizzato".

complete Valore booleano TRUE o FALSE che indica se la bitmap della finestra è stata ricostruita completamente oppure no.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Una chiamata a BeginRefresh deve sempre essere seguita da una chiamata a EndRefresh. La funzione BeginRefresh avvia un'operazione di refresh ottimizzato; refresh ottimizzato significa che vengono ridisegnate soltanto le parti di finestra nascoste e tornate visibili. EndRefresh conclude il processo di refresh, ristabilendo lo stato delle strutture interne collegate alla finestra. Dopo l'esecuzione della funzione, il disegno nella finestra non avverrà più soltanto nelle parti nascoste tornate visibili, ma in tutta la finestra.

L'argomento complete nella chiamata della funzione EndRefresh consente d'indicare se la chiamata a BeginRefresh effettuata dal task è l'ultima che dev'essere effettuata per la finestra considerata. Se ci sono altri task in attesa di aggiornare tale finestra, questo flag dovrebbe essere impostato a FALSE. Quando poi i task si alternano, il task successivo può procedere a introdurre le proprie informazioni di refresh nella finestra. Quando l'ultimo task interessato alle operazioni di refresh nella finestra considerata ha terminato il suo lavoro, è possibile chiamare la funzione EndRefresh da tale task, impiegando TRUE per l'argomento complete.

EndRequest

Sintassi di chiamata della funzione

EndRequest (requester, window)
A0 A1

Scopo della funzione

Questa funzione provoca la cancellazione dal video di un requester associato alla finestra indicata, ed effettua il reset della finestra. Viene cancellato dal video soltanto il requester specificato. EndRequest non restituisce valori.

Argomenti della funzione

requester Indirizzo della struttura Requester il cui requester dev'essere rimosso dal video.

window Indirizzo della struttura Window nella quale appare il requester.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

I requester sono aree rettangolari per lo scambio d'informazioni, destinate a essere presentate nelle finestre da Intuition o dai programmi applicativi. Vengono chiamati requester perché porgono una richiesta alla quale l'utente deve rispondere perché il programma possa proseguire l'esecuzione. È possibile disporre requester in qualsiasi punto della finestra.

I requester possono apparire su esplicita richiesta di Intuition, del programma applicativo, o dell'utente quando preme due volte in rapida successione il pulsante destro del mouse.

Sebbene la finestra di un requester sia bloccata in input fino a quando non

viene soddisfatta la richiesta, l'output verso l'utente può continuare tranquillamente. Nulla impedisce a un task o a un programma di scrivervi; tuttavia così facendo si corre il rischio di sovrascrivere il requester, rendendolo inutilizzabile.

La funzione EndRequest ha lo scopo di rimuovere la presentazione del requester dalla finestra. Si noti che gli altri requester rimangono nella finestra; ciascuno di essi dev'essere rimosso dal video singolarmente, attraverso una chiamata alla funzione EndRequest.

FreeRemember

Sintassi di chiamata della funzione

FreeRemember (rememberKey, reallyForget)
AØ DØ

Scopo della funzione

Questa funzione libera un insieme di blocchi di memoria allocati singolarmente con la funzione AllocRemember. Può liberare le strutture Remember da sole, oppure anche le aree di memoria a esse relative (indirizzo e dimensione), a seconda di quanto indica il task nell'argomento reallyForget.

La struttura Remember permette di definire un'area di memoria allocata dalla funzione AllocRemember, e di essere collegata ad altre strutture Remember, così da formare una lista che descrive un insieme di aree allocate singolarmente con la funzione AllocRemember. Tramite FreeRemember, il task ha la facoltà di liberare in un colpo solo tutte le aree allocate e la memoria occupata dalle rispettive strutture Remember, o di liberare semplicemente la memoria occupata dalla lista di strutture Remember lasciando allocate le rispettive aree.

Argomenti della funzione

rememberKey Indirizzo del puntatore a una struttura Remember; l'indirizzo contenuto in questo puntatore dovrebbe essere quello memorizzato dall'ultima chiamata alla funzione AllocRemember.

reallyForget Valore booleano TRUE o FALSE che determina se si devono liberare soltanto i nodi Remember (FALSE) oppure tutta la memoria costituita dai nodi e dalle aree di memoria da essi definite (TRUE).

Discussione

Ci sono due funzioni della libreria Intuition che riguardano direttamente l'assegnazione della memoria: AllocRemember e FreeRemember. Tramite loro, il sistema Intuition crea automaticamente una lista concatenata di riferimenti ad aree allocate di memoria, grazie alla quale il task può liberare quelle aree molto facilmente.

La funzione FreeRemember ha due scopi. Primo, se un task sta procedendo con diverse allocazioni di memoria ma una di queste fallisce per mancanza di memoria, può essere costretto ad abbandonare la procedura per provvedere a liberare la maggior quantità di memoria possibile. L'abbandono della procedura coinvolge la liberazione di tutti i blocchi di memoria già allocati. Se sono stati allocati tramite ripetute chiamate ad AllocRemember, tutto quello che è necessario fare è chiamare la funzione FreeRemember; i diversi blocchi di memoria indicati nella lista vengono così liberati e si può riavviare la procedura.

Secondo, quando si entra nella fase di chiusura di un programma è necessario liberare tutta la memoria allocata dinamicamente. Se i blocchi sono stati allocati e raggruppati in una lista tramite ripetute chiamate della funzione AllocRemember si possono liberare tutti attraverso una chiamata alla funzione FreeRemember.

FreeSysRequest

Sintassi di chiamata della funzione

FreeSysRequest (window)
A0

Scopo della funzione

Questa funzione libera la memoria che era stata assegnata a una struttura Requester con una chiamata alla funzione BuildSysRequest.

Argomenti della funzione

window Indirizzo della struttura Window, precedentemente restituito da una chiamata alla funzione BuildSysRequest.

Discussione

Nella libreria Intuition ci sono sette funzioni per gestire i requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest, InitRequester, Request e SetDMRequest. Si vedano anche le spiegazioni relative a queste funzioni.

Se la funzione BuildSysRequest si è svolta con successo, deve aver restituito l'indirizzo della struttura Window, e il programma ha quindi potuto attendere alla user port della finestra considerata un evento che soddisfacesse il requester. In questo caso, quando l'utente provoca l'evento che soddisfa il requester il task può cancellarlo chiamando la funzione FreeSysRequest.

GetDefPrefs

Sintassi di chiamata della funzione

```
prefBuffer = GetDefPrefs (prefBuffer, size)
DØ                        AØ       DØ
```

Scopo della funzione

Questa funzione effettua una copia della struttura dati Preferences di default e la colloca nel buffer indicato dal task. Impostando appropriatamente l'argomento size, la funzione GetDefPrefs consente anche di realizzare una copia parziale della struttura dati Preferences di default del sistema. GetDefPrefs restituisce l'indirizzo del buffer indicato dal task come primo argomento.

Argomenti della funzione

prefBuffer Indirizzo del buffer di memoria destinato a ricevere la copia della struttura Preferences di default.

size Dimensione in byte del buffer; questo valore indica in pratica la quantità di byte della struttura Preferences che si vogliono copiare nel buffer.

Discussione

Nella libreria Intuition ci sono due funzioni che operano direttamente con la struttura Preferences: GetPrefs e GetDefPrefs. Tali funzioni consentono di ottenere una copia delle impostazioni di default di Intuition (la funzione GetDefPrefs) oppure delle impostazioni in uso (la funzione GetPrefs). Le informazioni così ottenute possono essere impiegate nel programma, per esempio, per leggere e modificare la velocità di trasmissione della porta seriale (terzo elemento della struttura Preferences).

Le preferenze di default sono quelle fornite quando Intuition viene aperto per la prima volta. Se s'imposta in modo adeguatamente ridotto il parametro size, si ottiene una copia parziale delle preferenze di default più importanti, che sono riunite nella parte alta della struttura Preferences.

La struttura Preferences

La struttura Preferences definisce i colori dello schermo video, la temporizzazione per i tasti del mouse e altri parametri di controllo del sistema che riguardano l'hardware dell'Amiga. Ecco un sommario elenco dei parametri della struttura Preferences.

- Il parametro PrinterPort ha due possibili valori: nove linee di scansione per la fonte Topaz-60 oppure otto linee di scansione per la fonte Topaz-80.

- Il parametro PrinterPort viene impostato a PARALLEL_PRINTER oppure a SERIAL_PRINTER per indicare al dispositivo Printer a quale delle due porte è collegata la stampante, e quindi se deve utilizzare il dispositivo Parallel o il dispositivo Serial.

- Il parametro BaudRate contiene la velocità di trasmissione della porta seriale.

- I parametri KeyRptSpeed e KeyRptDelay sono sotto-strutture timeval della struttura Preferences, che hanno a che fare con la velocità di

ripetizione dei tasti, ovvero la soglia di tempo dopo il quale un tasto mantenuto premuto inizia a ripetere l'evento relativo. Ogni struttura timeval contiene due parametri, uno per il valore dei secondi e uno per il valore dei microsecondi.

- Il parametro DoubleClick contiene il massimo intervallo di tempo consentito tra due pressioni dei tasti del mouse, per considerarle un unico evento double-click.

- Le informazioni per definire i dati dello sprite che costituisce il puntatore video di default di Intuition: (1) i parametri XOffset e YOffset sono gli offset dall'angolo superiore sinistro dell'immagine di default del puntatore di Intuition per identificare il punto attivo dell'immagine. (2) Color 17, Color 18 e Color 19 sono le definizioni dei registri di colore per il puntatore del mouse. (3) Il parametro pointerTicks contiene il numero degli scatti di movimento del mouse richiesti affinché l'immagine del puntatore effettui uno spostamento sullo schermo video.

- I parametri Color0, Color1, Color2 e Color3 sono le definizioni dei registri di colore per i colori di default dello schermo Workbench.

- I parametri ViewXOffset e ViewYOffset descrivono gli offset della view (cioè della schermata) rispetto al punto in cui era stata originariamente aperta. Tali offset registrano la posizione dello schermo sul monitor.

- I parametri ViewInitX e ViewInitY contengono le coordinate iniziali x e y della view.

- Il parametro EnableCLI controlla la presentazione dell'icona relativa al CLI (Command Line Interface, interfaccia linea comando). Non più in uso nella versione 1.3.

- Il parametro PrinterType contiene la definizione dei tipi di stampanti disponibili.

- Il buffer PrinterFileName[FILENAME_SIZE] contiene il nome del file di default che costituisce il driver di stampa selezionato. FILENAME_SIZE (= 30) è il numero massimo di caratteri che possono comporre il nome del driver.

- I parametri PrintPitch, PrintQuality e PrintSpacing contengono diversi attributi per la stampa.

- I parametri PrintLeftMargin e PrintRightMargin indicano i margini di stampa.

- I parametri PrintImage, PrintAspect e PrintShade contengono gli attributi per la pagina di stampa.

- Il parametro PrintThreshold nella stampa in bianco e nero regola l'intensità del nero.
- I parametri PaperSize, PaperLength e PaperType indicano gli attributi della carta per la stampante.

GetPrefs

Sintassi di chiamata della funzione

```
prefBuffer = GetPrefs (prefBuffer, size)
D0                     A0          D0
```

Scopo della funzione

Questa funzione copia la struttura dati Preferences nel buffer indicato dal task. Impostando appropriatamente l'argomento size, la funzione GetPrefs consente anche di realizzare una copia parziale. GetPrefs restituisce l'indirizzo del buffer indicato dal task come primo argomento.

Argomenti della funzione

prefBuffer	Indirizzo del buffer di memoria destinato a ricevere la copia della struttura Preferences.
size	Dimensione in byte del buffer; questo valore indica in pratica la quantità di byte della struttura Preferences che si vogliono copiare nel buffer.

Discussione

Nella libreria Intuition ci sono due funzioni che agiscono direttamente con la struttura Preferences: GetPrefs e GetDefPrefs. Dal momento che l'utente può cambiare le impostazioni di Intuition mandando in esecuzione il tool Preferences attraverso la relativa icona (oppure da CLI), il programma deve avere la possibilità di essere informato dei cambiamenti: a questo scopo si impiega GetPrefs. Se il programma desidera invece conoscere le impostazioni

di default, deve chiamare la funzione GetDefPrefs.

Tramite il sistema IDCMP, si possono impostare i flag IDCMP a NEWPREFS, in modo che ogni volta che l'utente cambia le impostazioni di preferenza il task venga automaticamente avvertito. Si può poi chiamare GetPrefs per immagazzinare alcuni o tutti i dati della struttura Preferences in un buffer per esaminarli in seguito.

I programmatori che realizzano driver di stampa dovrebbero sempre chiamare la funzione GetPrefs prima di ogni operazione di stampa, dato che l'utente potrebbe aver alterato i parametri di stampa attraverso il programma Preferences.

GetScreenData

Sintassi di chiamata della funzione

```
success = GetScreenData (buffer, size, screen_type, screen)
D0                       A0     D0  D1           A1
```

Scopo della funzione

Questa funzione copia i dati contenuti nella struttura Screen, tutti o in parte, nel buffer indicato dal task nell'argomento buffer. Il programma può poi esaminare e usare questi dati. Se nell'argomento screen_type viene indicato lo schermo standard (WBENCHSCREEN) e tale schermo non è aperto, la funzione GetScreenData provvede ad aprirlo. Se la chiamata si conclude positivamente, viene restituito il valore TRUE, altrimenti FALSE.

Argomenti della funzione

buffer — Indirizzo del buffer destinato a ricevere tutti o una parte dei dati che costituiscono la struttura Screen.

size — Dimensione in byte del buffer. È possibile specificare un numero di byte inferiore a quello della struttura Screen.

screen_type — Il tipo di schermo indicato nella struttura Screen. Può essere CUSTOMSCREEN oppure WBENCHSCREEN.

screen Indirizzo della struttura Screen per uno schermo personalizzato. Se l'argomento che definisce il tipo è impostato a WBENCHSCREEN, l'argomento screen viene ignorato.

Discussione

Per rilevare i dati relativi allo schermo Workbench si deve utilizzare la chiamata:

```
success = GetScreenData (buffer, sizeof (struct Screen), WBENCHSCREEN, NULL);
```

Per rilevare i dati di uno schermo personalizzato si deve utilizzare la chiamata:

```
success = GetScreenData (buffer, sizeof (struct Screen), CUSTOMSCREEN, screen);
```

In questo caso il quarto argomento è l'indirizzo della struttura Screen relativa al particolare schermo personalizzato.

I primi sei elementi della struttura Screen sono i parametri NextScreen, FirstWindow, LeftEdge, TopEdge, Width e Height. Se si specifica un buffer per immagazzinare questi parametri, il programma può poi esaminarli in seguito, dopo la chiamata alla funzione GetScreenData. Altrimenti si può specificare un buffer per l'intera struttura Screen, com'è già stato spiegato.

In genere la funzione GetScreenData serve per ottenere la misura, l'altezza della barra titolo e altri parametri relativi allo schermo Workbench; la struttura Screen per lo schermo Workbench è definita da Intuition e questa funzione permette di accedervi.

InitRequester

Sintassi di chiamata della funzione

```
InitRequester (requester)
              A0
```

Scopo della funzione

Questa funzione inizializza la struttura Requester indicata dal task. Quando la funzione restituisce il controllo, si devono inizializzare solo pochi valori specifici; tutti gli altri vengono impostati a NULL. InitRequester non restituisce alcun valore.

Argomenti della funzione

requester Indirizzo della struttura Requester.

Discussione

Nella libreria Intuition ci sono sette funzioni che hanno direttamente a che fare con la gestione dei requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest, InitRequester, Request e SetDMRequest. Si vedano anche le spiegazioni relative a queste funzioni.

Una struttura Requester può essere inizializzata impiegando esplicite istruzioni di assegnazione di parametro, oppure impiegando la funzione InitRequester. In molti casi l'impostazione della struttura Requester si esegue impiegando entrambi i metodi. In un primo momento la funzione InitRequester viene adoperata per impostare a zero la maggior parte dei parametri della struttura. Quando poi vanno definiti i valori iniziali di particolari elementi della struttura, vengono impiegate esplicite istruzioni di assegnazione di parametro.

La struttura Requester

Ecco in sintesi i parametri della struttura Requester:

- il parametro OlderRequest realizza un collegamento (mantenuto da Intuition) con la struttura Requester relativa al precedente requester nello stesso elemento video contenitore.
- I parametri LeftEdge e TopEdge contengono l'offset in pixel dai bordi dell'elemento video contenitore in corrispondenza del quale andrà collocato il requester.
- I parametri Width e Height indicano le misure del requester.
- I parametri RelLeft e RelTop contengono la posizione del requester rispetto all'immagine del puntatore video nella finestra. È uno dei modi per collocare il requester nella finestra.

- Il puntatore ReqGadget punta al primo gadget di una lista concatenata di gadget associati all'elemento video contenitore.
- Il puntatore ReqBorder punta alla struttura Border che definisce i contorni del requester considerato.
- Il puntatore ReqText punta a una struttura IntuiText che definisce il testo del requester.
- Flags contiene i bit che vengono impostati sia da Intuition che dai programmi applicativi. Il programma può impostare il flag POINTREL per informare il sistema che il requester deve apparire in una posizione relativa all'immagine del puntatore video nella finestra; se viene impostato il flag PREDRAWN il sistema deve utilizzare un'immagine bitmap personalizzata appositamente realizzata per il requester. Intuition provvede a impostare REQOFFWINDOW (che indica se parte di un gadget del requester è stata presentata al di fuori della relativa finestra). Gli altri due flag impostati da Intuition sono REQACTIVE, che indica quando il requester è in uso e SYSREQUEST che viene impostato quando si tratta di un requester generato dal sistema.
- Il parametro BackFill contiene la penna per il disegno dello sfondo del requester.
- Il parametro ReqLayer viene usato da Intuition per creare l'immagine del requester per una bitmap personalizzata.
- Il puntatore ImageBMap punta a una bitmap personalizzata per il requester.

IntuiTextLength

Sintassi di chiamata della funzione

```
textWidth = IntuiTextLength (intuiText)
DØ                          AØ
```

Scopo della funzione

Questa funzione rileva i parametri della fonte-carattere in uso e calcola la lunghezza in pixel che occuperebbe sul video la stringa di testo definita dalla

struttura IntuiText indicata come argomento; infine, restituisce il valore calcolato.

Argomenti della funzione

intuiText Indirizzo della struttura IntuiText.

Discussione

Nella libreria Intuition ci sono due funzioni che operano direttamente con i testi: IntuiTextLength e PrintIText. Queste funzioni riguardano in modo particolare la struttura IntuiText, che serve per definire una stringa a terminazione nulla. Le funzioni IntuiTextLength e PrintIText consentono rispettivamente di ottenere la lunghezza (cioè la misura effettiva in pixel) della stringa di testo considerata e di stampare (cioè di disegnare in una bitmap) tale stringa, in accordo con gli argomenti di controllo nella funzione PrintIText. La funzione IntuiTextLength può, per esempio, essere impiegata quando si vuole conoscere la lunghezza di una stringa testo destinata ad apparire nella voce di un menu.

La struttura IntuiText

Ecco in sintesi il contenuto della struttura IntuiText:

- il parametro FrontPen è il colore usato per disegnare i caratteri del testo. Il parametro BackPen è il colore impiegato per disegnare le aree di fondo intorno ai caratteri; il colore BackPen, però, viene adoperato soltanto nel modo di disegno JAM2.

- Il parametro DrawMode indica uno dei modi di disegno usati per il testo in ambiente Intuition. Si veda anche la spiegazione relativa alla funzione PrintIText.

- Il parametro LeftEdge indica la coordinata x iniziale del testo e costituisce l'offset orizzontale dall'angolo superiore sinistro dell'elemento video contenitore.

- Il parametro TopEdge indica la coordinata y iniziale del testo e costituisce l'offset verticale dall'angolo superiore sinistro dell'elemento video contenitore.

- Il puntatore ITextFont punta a una struttura TextAttr che contiene una descrizione personalizzata della fonte-carattere. Può essere impostato a

NULL, se si vuole utilizzare la fonte di default di Intuition (Topaz-60 oppure Topaz-80).

- Il puntatore IText punta alla stringa di testo a terminazione nulla.
- Il puntatore NextText punta a un'altra struttura IntuiText. Questo consente di definire una lista concatenata di stringhe di testo da presentare in un elemento contenitore video. Il puntatore va impostato a NULL se la struttura IntuiText non fa parte di una lista concatenata o se la struttura è l'ultima della lista.

ItemAddress

Sintassi di chiamata della funzione

```
menuItem = ItemAddress (menu, menuNumber)
D0                      A0   D0
```

Scopo della funzione

Questa funzione restituisce l'indirizzo della struttura MenuItem per la voce di menu specificata dall'argomento menuNumber. Si può usare questa funzione, dopo che l'utente ha effettuato una selezione di menu, per rilevare l'indirizzo della voce di menu corrispondente al valore menuNumber inviato al task da Intuition.

Argomenti della funzione

menu	Indirizzo della prima struttura Menu in una barra di menu.
menuNumber	Valore che contiene, in formato compresso, i dati necessari per selezionare il menu con tutte le relative voci e sotto-voci.

Discussione

Ci sono cinque funzioni della libreria Intuition che gestiscono direttamente i menu: ClearMenuStrip, ItemAddress, OnMenu, OffMenu e SetMenuStrip. Si vedano anche le spiegazioni relative a queste funzioni.

La funzione ItemAddress richiede che gli argomenti siano dettagliatamente definiti. Se l'argomento menuNumber è un valore NULL, la funzione ItemAddress restituisce un valore NULL. Se il valore di menuNumber non è nullo, deve includere un numero valido di voci e di sotto-voci di menu.

Per maggiori informazioni sui menu, sulle voci e le sotto-voci si veda la spiegazione della funzione SetMenuStrip.

ITEMNUM

Sintassi di chiamata della macro

ITEMNUM (menuNumber)

Scopo della macro

Questa macro estrae dall'argomento menuNumber il numero d'ordine della voce di menu, che inizia da zero e giunge a 63. Questa macro è necessaria per permettere il confronto tra il flag di messaggio di selezione menu IDCMP (MENUPICK) e le definizioni delle voci di menu, determinando quindi quale voce del menu è stata selezionata dall'utente.

Argomenti della macro

menuNumber Numero binario a 16 bit che definisce in formato compresso il numero di menu, il numero della voce e della sotto-voce di menu.

Discussione

Le macro che riguardano i menu, le voci e le sotto-voci di menu sono sei: MENUNUM, ITEMNUM, SUBNUM, SHIFTMENU, SHIFTITEM e SHIFTSUB. Si

vedano anche le spiegazioni relative a queste macro.

La macro ITEMNUM assume un valore compreso tra 0 e 63; il numero indica quale particolare menu è stato selezionato dall'utente. L'argomento menuNumber può appartenere a una delle seguenti classi di valori:

- un numero di menu valido che definisca un menu, una voce di menu ed eventualmente una sotto-voce di menu. Si noti che il valore menuNumber deve sempre definire almeno un menu e una voce di menu. Il sistema, infatti, non considera un "evento" la semplice apertura di un menu.

- MENUNULL se non è stata effettuata alcuna selezione di menu o se una selezione è stata tentata impropriamente. Per esempio, se l'utente ha selezionato un menu ma non ha proseguito scegliendo almeno una voce di menu, il valore di menuNumber risulterà MENUNULL.

Per determinare il numero relativo alla voce di menu, si adoperi la seguente istruzione di programma:

```
MyItemNumber = ITEMNUM (menuNumber);
```

MyItemNumber definirà quindi un valore valido (compreso tra 0 e 63) oppure lo speciale valore NOITEM, nel caso in cui nessuna voce di menu risulti selezionata. I valori di MENUNULL e NOITEM sono definiti nell'ultima parte del file INCLUDE intuition.h, insieme alla definizione particolareggiata della macro MENUITEM.

LockIBase

Sintassi di chiamata della funzione

```
lock_value = LockIBase (lock_number)
DØ                      DØ
```

Scopo della funzione

Questa funzione s'impadronisce di un semaforo interno di Intuition, così che il programma possa esaminare con tranquillità la struttura IntuitionBase, quella che definisce la libreria Intuition. Quando LockIBase ritorna, la struttura IntuitionBase non potrà più essere alterata da Intuition o da altri task. LockIBase restituisce un valore ULONG, che deve poi essere adoperato per togliere il blocco attraverso la funzione UnlockIBase.

Argomenti della funzione

lock_number Numero intero di tipo ULONG, che rappresenta il blocco che si vuole attivare. Tale argomento dovrebbe valere zero per tutti i prevedibili impieghi della funzione LockIBase. Futuri perfezionamenti potranno permettere di determinare in modo esplicito tale argomento.

Discussione

Nel software sistema di Intuition ci sono due funzioni che riguardano le operazioni di blocco: LockIBase e UnlockIBase.

La struttura IntuitionBase contiene tutti i parametri di gestione necessari a Intuition per svolgere il proprio lavoro. Normalmente solo Intuition può accedere ai parametri nella struttura IntuitionBase; quando lo fa, blocca tale struttura impiegando i suoi semafori interni e impedisce che qualsiasi altro task nel sistema possa accedervi e cambiarla.

La funzione LockIBase fa in modo che i programmi si comportino come Intuition; per la precisione, blocca la struttura IntuitionBase impedendo a Intuition e a ogni altro task di alterarne i contenuti per un certo tempo. Il task chiamante assume quindi il completo controllo della struttura IntuitionBase. In tal modo si può vincolare una struttura IntuitionBase prima che Intuition ne alteri i parametri. Si possono, per esempio, esaminare i parametri ActiveWindow e FirstScreen con la certezza che i loro valori non si altereranno mentre si esegue il controllo. In tal modo, sotto la protezione del vincolo, si può anche esaminare una lista concatenata di finestre e schermi.

Si noti che le chiamate alle funzioni LockIBase e UnlockIBase devono sempre essere accoppiate. Inoltre il programma non dovrebbe impiegare tali vincoli per un periodo di tempo prolungato. Intuition è infatti in attesa che il programma liberi tali blocchi attraverso una chiamata alla funzione UnlockIBase, per poter continuare i propri compiti.

MakeScreen

Sintassi di chiamata della funzione

MakeScreen (screen)
AØ

Scopo della funzione

Questa funzione genera la lista di istruzioni Copper per uno schermo personalizzato di Intuition, tramite la chiamata alla funzione MakeVPort della libreria Graphics.

Argomenti della funzione

screen	Indirizzo della struttura Screen che definisce lo schermo personalizzato.

Discussione

Nella libreria Intuition ci sono sette funzioni che agiscono direttamente con gli schermi video di Intuition: OpenScreen, CloseScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront. Si vedano anche le spiegazioni relative a queste funzioni.

Ogni schermo personalizzato di Intuition viene creato inizializzando una struttura NewScreen e chiamando poi la funzione OpenScreen per allocare la memoria necessaria alla struttura Screen e a tutte le sue sotto-strutture. Ogni struttura Screen di Intuition contiene una sotto-struttura ViewPort. Ciò è coerente con il fatto che ogni schermo personalizzato Intuition è una viewport. Per questa ragione, la procedura interna di sistema, richiesta per presentare sullo schermo video la schermata desiderata, è identica alla procedura adoperata dalle funzioni della libreria Graphics per presentare qualsiasi altra viewport. Tali operazioni vengono svolte chiamando la funzione MakeVPort della libreria Graphics.

MakeVPort definisce la lista delle istruzioni Copper richieste per una viewport. Quando la funzione MakeScreen restituisce il controllo, occorre chiamare la funzione RethinkDisplay per incorporare la nuova viewport dello schermo personalizzato nella presentazione video di Intuition.

I flag della struttura ViewPort

Ecco una sintesi dei flag della struttura ViewPort:

- HIRES. Se questo flag è impostato a 1, la viewport dello schermo è in alta risoluzione.
- LACE. Se questo flag è impostato a 1, la viewport dello schermo è in interlace.
- SPRITES. Se questo flag è impostato a 1, la viewport dello schermo considerato è destinata ad accogliere sprite.
- DUALPF. Se questo flag è impostato a 1, la viewport dello schermo considerato è destinata ad avere due playfield.
- EXTRA_HALFBRITE. Se questo flag è impostato a 1, la viewport dello schermo considerato è in modo Extra Half-Brite. Ciò permette di avere sullo schermo 64 colori diversi.
- HAM. Se questo flag è impostato a 1, la viewport dello schermo è in modo Hold And Modify. Ciò permette di mostrare simultaneamente 4096 colori.

MENUNUM

Sintassi di chiamata della macro

MENUNUM (menuNumber)

Scopo della macro

Questa macro estrae dall'argomento menuNumber il numero d'ordine del menu, che può andare da 0 a 63. Questa macro serve per permettere il confronto tra il flag di messaggio di selezione menu IDCMP (MENUPICK) e le definizioni di menu, ovvero per determinare quale menu è stato selezionato dall'utente.

Argomenti della macro

menuNumber Numero binario da 16 bit che definisce in formato compresso il numero del menu, il numero della voce di menu e il numero della sotto-voce di menu.

Discussione

Le macro che riguardano i menu, le voci e le sotto-voci dei menu sono sei: MENUNUM, ITEMNUM, SUBNUM, SHIFTMENU, SHIFTITEM e SHIFTSUB. Si vedano anche le spiegazioni relative a queste macro.

Intuition individua le posizioni dei menu, delle voci e delle sotto-voci dei menu usando due metodi. Il primo impiega un numero binario da 16 bit (menuNumber) per rappresentare in formato compresso tutti e tre i componenti del menu come sotto-campi del numero binario. I bit da 0 a 4 rappresentano il numero del menu; i bit da 5 a 10 rappresentano il numero della voce; i bit da 11 a 15 rappresentano il numero della sotto-voce. La funzione ItemAddress agisce con la variabile menuItem, la quale restituisce l'indirizzo di una struttura MenuItem che rappresenta una determinata sezione di menu (voce o sotto-voce di menu).

Il secondo metodo rappresenta ogni parte del menu come un numero d'ordine che identifica il componente considerato. Il sistema di messaggi IDCMP di Intuition richiede la rappresentazione come numero d'ordine per riconoscere correttamente le selezioni di menu dell'utente, quando un messaggio IDCMP MENUPICK viene generato da una selezione di menu effettuata dall'utente.

Il flag di IDCMP, MENUPICK, rappresenta la specifica scelta di menu come posizione d'ordine in ciascuno dei sotto-campi del numero binario compresso. Perciò, al fine di permettere al programma di determinare il menu selezionato dall'utente, è necessaria questa macro per estrarre i numeri d'ordine corrispondenti a ciascuno dei sotto-campi della rappresentazione menuNumber.

Per determinare il numero di menu si usa la seguente istruzione di programma:

```
MyMenuNumber = MENUNUM (menuNumber)
```

Il modo più conveniente per elaborare una selezione di menu effettuata dall'utente è il seguente:

```
while (menuNumber! = MENUNULL){
     MenuItem = ItemAddress (menuStrip, menuNumber);
     /* elabora la voce considerata */
     MenuNumber = MenuItem->NextSelect;
};
```

Qui NextSelect è uno dei parametri della struttura MenuItem. Se l'utente ha effettuato la selezione di una sola voce di menu, questo parametro risulta nullo.

ModifyIDCMP

Sintassi di chiamata della funzione

```
ModifyIDCMP (window, IDCMPFlags)
            AØ       DØ
```

Scopo della funzione

Questa funzione modifica lo stato delle impostazioni IDCMP della finestra in accordo con i bit di flag della variabile IDCMPFlags. Se l'argomento IDCMPFlags è nullo, ModifyIDCMP provvede a chiudere le message port IDCMP. In caso contrario, la funzione reimposta i flag delle message port IDCMP già aperte, o ne apre due di nuove nella finestra. ModifyIDCMP non restituisce alcun valore.

Argomenti della funzione

window	Indirizzo della struttura Window.
IDCMPFlags	Bit di flag che descrivono lo stato desiderato per le message port IDCMP

Discussione

ModifyIDCMP è una funzione della libreria Intuition che non ricade in alcuna categoria specifica. Questa funzione permette di svolgere le quattro azioni seguenti:

1. Se nessuna message port IDCMP è assegnata alla finestra e la variabile IDCMPFlags è nulla, la finestra rimane senza nessuna message port (né user port, né window port).

2. Se nessuna message port IDCMP è assegnata alla finestra e la variabile IDCMPFlags non è nulla (cioè alcuni flag sono impostati), alla finestra viene assegnata una coppia di message port (una user port e una window port).

3. Se le message port IDCMP relative alla finestra sono aperte e la variabile IDCMPFlags è nulla, le message port vengono chiuse.

4. Se le message port IDCMP relative alla finestra sono aperte e la variabile IDCMPFlags non è nulla, vengono impostati nuovi flag per la finestra. Ciò consente al programma di riconoscere ed elaborare un completo nuovo insieme di eventi prodotti dall'utente.

I task possono ricevere l'input dell'utente attraverso una finestra di Intuition in due modi: attraverso un'unità del dispositivo Console o una message port IDCMP (Intuition Direct Communications Message Ports, message port per comunicazioni dirette con Intuition). Se si adopera il sistema IDCMP, si possono indicare a Intuition quali sono gli eventi di input di cui si desidera essere informati. Questi eventi ricadono in sei categorie: messaggi provenienti dal mouse, dai gadget, dai menu, dai requester, dalla finestra, e altri messaggi vari.

Per la maggior parte, questi messaggi vengono generati dall'utente attraverso l'intervento sul mouse e sulla tastiera. Se il programma ha bisogno di essere informato di queste azioni, prepara un canale IDCMP per ricevere i relativi messaggi. Nella finestra viene quindi preparata una user port che appartiene al programma, e nella quale il programma riceverà i messaggi di Intuition. Alla finestra viene inoltre assegnata un'altra message port, detta window port, nella quale Intuition si aspetta di ottenere le risposte ai messaggi che ha inviato alla user port. Si veda anche la spiegazione relativa alla funzione AddPort nel capitolo 1.

Le informazioni scambiate tra la user port e la window port, vengono controllate dall'impostazione dei flag nella definizione IDCMP. Si possono impostare questi flag nella struttura NewWindow e in seguito modificarli con la funzione ModifyIDCMP. Ecco un breve sommario dei flag IDCMP.

I flag IDCMP relativi al mouse

- MOUSEBUTTONS. Se questo flag è impostato, gli eventi di pressione e rilascio del pulsante sinistro del mouse vengono inviati al programma dopo essere stati elaborati da parte di Intuition (se il flag RMBTRAP è impostato, saranno trasmessi anche gli eventi riguardanti il pulsante destro). Ogni evento che Intuition usa ed elabora (ad esempio la selezione di un gadget), non viene trasmesso al programma.

- MOUSEMOVE. Se questo flag è impostato, il programma riceve informazioni sui movimenti del mouse.

- DELTAMOVE. Se questo flag è impostato, gli eventi che riguardano i movimenti del mouse vengono inviati come valori relativi all'ultima posizione, anziché come coordinate assolute.

I flag IDCMP relativi ai gadget

- GADGETDOWN. Se questo flag è impostato, il task riceve gli eventi che riguardano le selezioni dei gadget.
- GADGETUP. Se questo flag è impostato, il task riceve gli eventi che riguardano le deselezioni dei gadget.
- CLOSEWINDOW. Se questo flag è impostato, viene inviato un messaggio al programma quando l'utente seleziona il gadget di chiusura della finestra.

I flag IDCMP relativi ai menu

- MENUPICK. Se questo flag è impostato, viene inviato un messaggio al programma quando l'utente seleziona un menu.
- MENUVERIFY. Se questo flag è impostato, tutte le operazioni di disegno in una finestra verranno completate prima che sia consentito all'utente d'iniziare operazioni con i menu.

I flag IDCMP relativi ai requester

- REQSET. Se questo flag è impostato, viene inviato un messaggio al programma quando nella finestra si apre il primo requester.
- REQCLEAR. Se questo flag è impostato, viene inviato un messaggio al programma quando l'ultimo requester viene cancellato dalla finestra.
- REQVERIFY. Se questo flag è impostato, prima che un requester venga inserito nella finestra tutte le operazioni di disegno devono essere finite.

I flag IDCMP relativi alle finestre

- NEWSIZE. Se questo flag è impostato, Intuition invia un messaggio al programma quando l'utente ridimensiona la finestra.
- REFRESHWINDOW. Se questo flag è impostato, viene inviato un messaggio al programma se una finestra ha bisogno di essere ricostruita (nel caso, per esempio, delle finestre a refresh semplice).

- SIZEVERIFY. Se questo flag è impostato, prima che l'utente possa ridimensionare la finestra tutte le operazioni di disegno devono essere finite.

- ACTIVEWINDOW. Se questo flag è impostato, il programma riceve un messaggio ogni volta che la finestra viene attivata.

- INACTIVEWINDOW. Se questo flag è impostato, il programma riceve un messaggio ogni volta che la finestra viene disattivata.

Flag IDCMP vari

- RAWKEY. Se questo flag è impostato, i codici grezzi di tastiera conseguenti alle pressioni dei tasti vengono trasmessi al programma senza essere convertiti.

- VANILLAKEY. Se questo flag è impostato, ogni volta che l'utente preme un tasto (quando la finestra è attiva), il task riceve un messaggio IDCMP nel quale la classe è VANILLAKEY e il codice è il codice ASCII del carattere associato al tasto. Si tratta di uno dei metodi che hanno i task per acquisire caratteri da tastiera, con la peculiarità però che il codice ASCII di cui il task entra in possesso è già stato filtrato dalla mappa di tastiera nazionale prescelta. Quindi, se il programma acquisisce i caratteri da tastiera con questo tipo di evento, non scavalca la mappa di tastiera impostata dal comando SETMAP del CLI, come purtroppo spesso accade con molti programmi applicativi in commercio.

- NEWPREFS. Se questo flag è impostato, viene inviato un messaggio al programma quando l'utente cambia le impostazioni di preferenza.

- DISKINSERTED. Se questo flag è impostato, viene inviato un messaggio al programma quando l'utente inserisce un disco nel disk drive.

- DISKREMOVED. Se questo flag è impostato, viene inviato un messaggio al programma quando l'utente rimuove un disco da un disk drive.

ModifyProp

Sintassi di chiamata della funzione

```
ModifyProp (gadget, pointer, requester, flags, horizPot, vertPot,
            AØ      A1       A2         DØ     D1        D2
            horizBody, vertBody)
            D3         D4
```

Scopo della funzione

Questa funzione permette di modificare i parametri di un gadget proporzionale, com'è specificato nella sua struttura PropInfo. La struttura Gadget per il gadget proporzionale contiene un puntatore alla struttura PropInfo. La struttura Gadget viene aggiornata e l'immagine del gadget viene nuovamente presentata. Il parametro pointer indica a quale finestra appartiene il gadget o il requester contenente il gadget. Se la struttura Gadget ha il flag REQGADGET impostato, il parametro requester deve puntare alla struttura Requester corrispondente. Se il gadget non è associato a un requester, l'argomento puntatore requester dev'essere NULL. ModifyProp non restituisce alcun valore.

Argomenti della funzione

gadget — Indirizzo della struttura Gadget del gadget proporzionale.

pointer — Puntatore alla struttura Window contenente il gadget.

requester — Indirizzo della struttura Requester di cui fa parte il gadget; se il gadget non fa parte di un requester il valore del parametro dev'essere NULL.

flags — Valore da 2 byte da inserire nel parametro Flags della struttura PropInfo.

horizPot — Valore da 2 byte da inserire nel parametro HorizPot della struttura PropInfo.

VertPot Valore da 2 byte da inserire nel parametro VertPot della struttura PropInfo.

horizBody Valore da 2 byte da inserire nel parametro HorizBody della struttura PropInfo.

vertBody Valore da 2 byte da inserire nel parametro VertBody della struttura PropInfo.

Discussione

Nella libreria Intuition ci sono sei funzioni che riguardano la gestione dei gadget: AddGadget, OnGadget, OffGadget, RefreshGadget, RemoveGadget e ModifyProp. Si vedano anche le spiegazioni relative a queste funzioni.

Un gadget proporzionale permette ai task di specificare alcuni dati proporzionali e di vederli rappresentati appunto in modo proporzionale. I gadget di controllo del colore del tool Preferences sono un buon esempio di gadget proporzionali. Questo genere di gadget può essere progettato in due modi: primo, si possono impiegare le immagini fornite da Intuition; secondo, si può disegnare per l'utente l'immagine di un potenziometro a cursore o di un interruttore e scegliere un'impostazione proporzionale.

Un gadget proporzionale ha diverse parti che operano insieme per produrre il risultato voluto, ovvero la variabile pot, la variabile body, il cursore e il contenitore. La parola *pot* è l'abbreviazione di potenziometro. Ci sono due variabili pot perché i gadget proporzionali possono essere regolati secondo l'asse verticale e secondo quello orizzontale. In genere le variabili pot vengono impostate a 0 e poi regolate dall'utente muovendo il cursore del gadget proporzionale. Una volta che il gadget proporzionale è stato definito e attivato, si possono leggere in ogni momento i valori assegnati dall'utente alle variabili pot.

Le variabili horizBody e vertBody descrivono l'incremento delle variabili pot in ciascuna direzione. Le variabili body possono anche avere lo stesso valore. Quando l'utente effettua una selezione con il puntatore in un contenitore di gadget proporzionale, le variabili pot agiscono a seconda della quantità impostata nelle variabili body.

Il cursore è l'oggetto su cui agisce effettivamente l'utente per cambiare le variabili pot secondo gli incrementi specificati nelle variabili body. A ogni pressione del pulsante sinistro, la variabile pot viene aumentata o diminuita del valore unitario impostato nelle variabili body.

Il contenitore è l'area nella quale il cursore può essere spostato. Il contenitore è praticamente il rettangolo di selezione del gadget. La misura del contenitore può essere relativa alla misura della finestra, e crescere o diminuire parallelamente agli ingrandimenti e rimpicciolimenti della finestra.

MoveScreen

Sintassi di chiamata della funzione

```
MoveScreen (screen, dx, dy)
            AØ      DØ  D1
```

Scopo della funzione

Questa funzione tenta di effettuare uno spostamento dello schermo pari ai valori indicati negli argomenti dx e dy. Se questi argomenti superano l'ampiezza dei movimenti consentiti per lo schermo, esso verrà spostato per quanto possibile. MoveScreen non restituisce alcun valore.

Argomenti della funzione

screen	Indirizzo della struttura Screen.
dx	Spostamento dello schermo lungo l'asse x.
dy	Spostamento dello schermo lungo l'asse y.

Discussione

Nella libreria Intuition ci sono sette funzioni che operano direttamente con gli schermi video di Intuition: OpenScreen, CloseScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront. Si vedano anche le spiegazioni relative a queste funzioni.

La funzione MoveScreen permette di spostare lo schermo nelle direzioni x e y. Tuttavia, finora, il movimento dello schermo ha un vincolo: non è concesso alcun movimento nella direzione x. Quindi l'argomento dx attualmente non è sfruttato.

Si noti che tutti gli schermi di Intuition, tanto quelli standard quanto quelli personalizzati, sono attualmente limitati alla piena larghezza dello schermo video dell'Amiga. In un futuro aggiornamento software, tali restrizioni saranno verosimilmente cambiate per consentire agli schermi spostamenti completi nelle direzioni x e y.

MoveWindow

Sintassi di chiamata della funzione

```
MoveWindow (window, dx, dy)
            AØ      DØ  D1
```

Scopo della funzione

Questa funzione tenta di effettuare uno spostamento dello schermo pari ai valori indicati negli argomenti dx e dy. La finestra viene spostata non appena Intuition riceve un evento di input. MoveWindow non restituisce alcun valore.

Argomenti della funzione

window	Indirizzo della struttura Window.
dx	Valore con segno che definisce lo spostamento della finestra lungo l'asse x.
dy	Valore con segno che definisce lo spostamento della finestra lungo l'asse y.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Quando si chiama la funzione MoveWindow, la finestra viene spostata non appena Intuition riceve un evento di input. Ciò accade con una frequenza minima di dieci volte al secondo e fino a un massimo di 50.

Si noti che la funzione MoveWindow non controlla gli errori. Se i valori dx e dy spingono la finestra al di là dei limiti dello schermo, una parte della finestra (o anche tutta) scompare dallo schermo.

OffGadget

Sintassi di chiamata della funzione

OffGadget (gadget, window, requester)
A0 A1 A2

Scopo della funzione

Questa funzione disabilita, il gadget indicato come argomento. L'argomento window punta alla finestra corrispondente. L'argomento requester punta a una struttura Requester. Se la struttura Gadget ha il flag REQGADGET impostato, il gadget appartiene a un requester e l'argomento requester dev'essere impostato. Se il gadget non è associato a un requester, l'argomento requester può essere nullo. OffGadget non restituisce alcun valore.

Argomenti della funzione

gadget	Indirizzo della struttura Gadget che si vuole disabilitare.
window	Indirizzo della struttura Window.
requester	Indirizzo della struttura Requester; può essere nullo se non si tratta di un gadget per requester.

Discussione

Nella libreria Intuition ci sono sei funzioni che riguardano le gestione dei gadget: AddGadget, OnGadget, OffGadget, RefreshGadget, RemoveGadget e ModifyProp. Si vedano anche le spiegazioni relative a queste funzioni.

Quando si esegue la funzione OffGadget, l'immagine del gadget diventa evanescente, in conseguenza della sovrapposizione di un retino alla normale immagine. Quando un gadget assume questo stato non può essere selezionato dall'utente. La struttura Gadget contiene un flag per indicare questo stato: GADGETDISABLE. Questo flag va impostato a 1 se si vuole che il gadget appaia evanescente quando viene presentato sullo schermo per la prima volta.

Dopo che un gadget è apparso sullo schermo, è possibile cambiare lo stato in disabilitato o selezionabile utilizzando le funzioni OffGadget e OnGadget. Se il gadget appartiene a un requester, questo deve apparire sullo schermo perché le funzioni OnGadget e OffGadget abbiano effetto.

OffMenu

Sintassi di chiamata della funzione

OffMenu (window, menuNumber)
A0 D0

Scopo della funzione

Questa funzione può svolgere tre operazioni: disabilitare una determinata voce di menu, disabilitare una certa sotto-voce di menu, disabilitare un intero menu. OffMenu non restituisce alcun valore.

Argomenti della funzione

window	Indirizzo della struttura Window.
menuNumber	Valore da due byte che definisce la parte di menu che dev'essere disabilitata.

Discussione

Ci sono cinque funzioni, nella libreria Intuition, che gestiscono direttamente i menu: ClearMenuStrip, ItemAddress, OnMenu, OffMenu e SetMenuStrip. Si vedano anche le spiegazioni relative a queste funzioni.

L'argomento menuNumber usato nelle funzioni OnMenu e OffMenu è una variabile a 16 bit. Cinque bit (dal bit 0 al bit 4) si riferiscono al numero di menu, sei bit (dal bit 5 al bit 10) al numero della voce di menu, e cinque bit (dal bit 11 al bit 15) al numero della sotto-voce.

I tre sotto-campi nella variabile menuNumber sono numeri binari compresi tra 0 e 11111 (oppure tra 0 e 111111 nel caso del campo relativo alla voce di menu). Ogni parte dell'informazione è determinata dalla sua posizione in una

lista di voci dello stesso livello. Per esempio, se si vuole selezionare il menu 1, basta impostare il primo campo a 00001. Se si vuole selezionare la terza voce, basta impostare il secondo sotto-campo a 000011.

Ciò significa che per ciascuna voce e sotto-voce di menu possono essere precisate fino a 31 sezioni. Sotto a ciascun menu possono essere quindi essere presentati 63 elementi. Se tutti i bit sono impostati a 0, significa che non è stata effettuata alcuna selezione.

Se il numero di voce nell'argomento menuNumber indica uno degli elementi voce in una struttura Menu o MenuItem, tale componente di menu viene disabilitato. Per abilitare o disabilitare una singola voce e tutte le sotto-voci a essa collegate, l'argomento MenuNumber va impostato al numero d'ordine della voce. Per esempio, per abilitare la voce 3 e tutte le sue sotto-voci, la variabile in questione va impostata a 0000000001100000. Se la voce di menu ha una lista di sotto-voci, si deve impostare la sotto-voce dell'argomento menuNumber a NOSUB. Per disabilitare una singola sotto-voce, si devono impostare appropriatamente la voce di menu e le sezioni di sotto-voce per l'argomento menuNumber.

Se il componente voce riferito dall'argomento menuNumber vale NOITEM, l'intero menu verrà disabilitato dalla chiamata alla funzione OffMenu, comprese tutte le sue voci e sotto-voci.

La definizione attuale dei parametri NOSUB e NOITEM può essere reperita nel file INCLUDE intuition.h.

OnGadget

Sintassi di chiamata della funzione

OnGadget (gadget, pointer, requester)
A0 A1 A2

Scopo della funzione

Questa funzione abilita un determinato gadget in una finestra. Presenta l'immagine del gadget e consente all'utente di selezionarlo. L'argomento pointer si riferisce a una struttura Window. OnGadget non restituisce alcun valore.

Argomenti della funzione

gadget	Indirizzo della struttura Gadget da abilitare.
pointer	Indirizzo della struttura Window.
requester	Indirizzo della struttura Requester; può essere nullo se non si tratta di un gadget per requester.

Discussione

Nella libreria Intuition ci sono sei funzioni che riguardano la gestione dei gadget: AddGadget, OnGadget, OffGadget, RefreshGadget, RemoveGadget e ModifyProp. Si vedano anche le spiegazioni relative a queste funzioni.

Per aggiungere uno o più gadget alla propria presentazione video si seguono i seguenti passi:

1. Si crea una struttura Gadget per ciascun gadget.
2. Si crea una lista concatenata di strutture Gadget per ciascun elemento video contenitore. Questi elementi video possono essere finestre o requester.
3. Si imposta la variabile di puntamento gadget relativa alla finestra o requester, affinché punti al primo gadget della lista.

Una struttura Gadget contiene generalmente le seguenti informazioni:

- l'immagine e/o il bordo del gadget. Se il gadget non ha immagini o bordi di contorno, si impiegano valori NULL.
- Il rettangolo di selezione del gadget. È l'area che Intuition sorveglia per rilevare se l'utente ha selezionato il gadget.
- Gli offset da sinistra e dall'alto, all'interno dell'elemento video contenitore relativo al gadget. Possono essere assoluti oppure relativi ai bordi della finestra o del requester.
- La larghezza e l'altezza del gadget rispetto all'elemento video che lo contiene. Possono essere assolute oppure relative ai bordi della finestra o del requester.
- Il tipo di gadget. Può essere booleano (con impostazioni TRUE o FALSE), numerico (con impostazioni numeriche), proporzionale (con una gamma di valori nelle direzioni x e y) o stringa (con un valore testo).

- Il comportamento che deve assumere Intuition mentre l'utente manipola il gadget.

OnMenu

Sintassi di chiamata della funzione

```
OnMenu (window, menuNumber)
        A0      D0
```

Scopo della funzione

Questa funzione, al contrario di OffMenu, abilita la voce di menu specificata o l'intero menu. Il tipo di azione è determinato dal valore dell'argomento menuNumber. Se il numero di menu corrisponde al menu visualizzato, viene ripresentata anche la barra menu. OnMenu non restituisce alcun valore.

Argomenti della funzione

window	Indirizzo della struttura Window.
menuNumber	Valore a due byte che definisce il menu, le voci di menu e le sotto-voci di menu che devono essere abilitate.

Discussione

Ci sono cinque funzioni, nella libreria Intuition, che agiscono direttamente con i menu: ClearMenuStrip, ItemAddress, OnMenu, OffMenu e SetMenuStrip. Si vedano anche le spiegazioni relative a queste funzioni.

I flag della struttura Menu

I due flag più importanti della struttura Menu sono:

- il flag MENUENABLED, che indica se il menu è abilitato. Se questo flag è impostato, l'intestazione del menu e tutte le voci che gli appartengono risultano abilitate. L'utente sarà quindi in condizione di selezionare una voce del menu. Se il flag non è impostato, tutte le voci risultano disabilitate.

- Il flag MIDRAWN indica se le voci di menu sono correntemente visibili.

I flag della struttura MenuItem

Sono parecchi i flag degni di nota della struttura MenuItem:

- il flag CHECKIT informa Intuition che questa voce è una voce-attributo; si vuole che al suo fianco appaia un segno di riscontro quando la voce viene selezionata e scompaia quando viene selezionata una seconda volta, per ricomparire la terza volta e così via.

- Il flag CHECKED viene impostato o azzerato quando s'inizializza la struttura MenuItem prima di sottoporla a Intuition. Se si vuole che questa voce risulti selezionata fin dall'inizio (cioè che appaia accompagnata dal segno di riscontro), si deve impostare questo flag. In seguito Intuition mantiene aggiornato questo flag per riportare lo stato di attivazione e disattivazione della voce-attributo.

- Il flag COMMSEQ indica che questa voce di menu è associata a un'equivalente combinazione di tasti.

- Il flag ITEMENABLED indica che la voce di menu è abilitata.

- Il flag HIGHCOMP produce il complemento di tutti i pixel contenuti nel rettangolo di selezione della voce di menu considerata quando essa viene selezionata.

- Il flag HIGHBOX disegna un rettangolo intorno al rettangolo di selezione della voce quando essa viene selezionata.

- Il flag HIGHIMAGE produce la presentazione di un'immagine alternativa, definita tramite SelectFill.

- Il flag HIGHNONE disabilita qualsiasi tipo di visualizzazione della selezione.

- Il flag ISDRAWN viene impostato da Intuition quando le sotto-voci della voce sono presentate all'utente; in caso contrario, Intuition lo azzera.

- Il flag HIGHITEM viene impostato da Intuition quando l'utente evidenzia la voce, portandovi sopra il puntatore del mouse; in caso contrario, Intuition lo azzera. Si noti che se l'utente porta il puntatore sulla voce e rilascia il pulsante destro del mouse (operazione che corrisponde per Intuition alla selezione della voce) il flag rimane impostato; viene azzerato soltanto quando l'utente porta il puntatore fuori dai limiti geometrici della voce.

OpenScreen

Sintassi di chiamata della funzione

```
screen = OpenScreen (newScreen)
D0                  A0
```

Scopo della funzione

Questa funzione apre uno schermo personalizzato di Intuition. Assegna la memoria necessaria per una struttura di tipo Screen, la inizializza e inizializza tutte le strutture correlate. OpenScreen lega inoltre questo schermo allo schema completo degli schermi di Intuition. Questa funzione restituisce al task l'indirizzo della struttura Screen che definisce il nuovo schermo.

Argomenti della funzione

newScreen	Indirizzo della struttura NewScreen.

Discussione

Nella libreria Intuition ci sono sette funzioni che agiscono direttamente con gli schermi video personalizzati: OpenScreen, CloseScreen, MoveScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront. Si vedano anche le spiegazioni relative a queste funzioni.

Il primo passo per la creazione di un nuovo schermo personalizzato è quello

di creare una struttura NewScreen. Per questa ragione la struttura NewScreen è molto importante. Per la definizione della struttura NewScreen si veda l'introduzione di questo capitolo.

Gli schermi di Intuition

Gli schermi Intuition ricadono in due generiche categorie: gli schermi standard e quelli personalizzati (custom). Lo schermo Workbench è il solo schermo standard attualmente disponibile nel sistema Intuition dell'Amiga; sono tuttavia previsti altri schermi standard. Gli schermi standard differiscono da quelli personalizzati per quattro motivi:

1. Gli schermi standard hanno sempre la piena larghezza e altezza dello schermo video. Al momento, agli schermi personalizzati è richiesto di avere la piena larghezza dello schermo video ma non la piena altezza.

2. Gli schermi standard (con l'eccezione di quello Workbench) si chiudono e scompaiono se il programmatore o l'utente chiudono tutte le finestre in essi presenti.

3. Gli schermi standard si aprono in modo diverso da quelli personalizzati, che impiegano la funzione OpenScreen.

4. Negli schermi standard (con l'eccezione di quello Workbench) i colori, il modo video standard e altri parametri non possono essere alterati.

In ambiente Intuition si possono creare due tipi di schermo personalizzato. Il primo è interamente gestito da Intuition. Il secondo è controllato tramite le funzioni della libreria Graphics ed è evidentemente più difficile da creare e da controllare.

Se si progetta uno schermo personalizzato in ambiente Intuition si possono impostare diversi parametri, tra cui i seguenti:

- l'altezza e il punto d'inizio verticale dello schermo, quando questo viene aperto per la prima volta.

- La profondità della bitmap di schermo; ciò determina quanti colori possono comparire nello schermo considerato.

- I colori per i dettagli del disegno dello schermo, inclusi i gadget e la barra titolo.

- Il modo video per lo schermo: dalla bassa risoluzione senza interlace fino all'alta risoluzione con interlace.

- La scelta tra il modo single-playfield, quello dual-playfield, quello Extra Half-Brite e quello Hold And Modify.

Il parametro flags nella struttura NewScreen include, inoltre, i seguenti flag:

- SCREENBEHIND. Al momento dell'apertura, lo schermo viene posto dietro tutti gli altri.
- SCREENQUIET. Intuition non disegna la barra titolo e i gadget di sistema in questo schermo.
- La posizione in RAM della memoria video.

Ecco i passi da seguire per creare il secondo tipo di schermo personalizzato:

1. Si deve allocare memoria per una struttura NewScreen. La struttura NewScreen va impostata con le caratteristiche di schermo che si desiderano per lo schermo personalizzato.
2. Si deve chiamare la funzione OpenScreen indicando come argomento l'indirizzo della struttura NewScreen. La funzione OpenScreen restituisce un puntatore alla struttura Screen che definisce il nuovo schermo.
3. Si può liberare o riutilizzare la memoria allocata per la struttura NewScreen.

OpenWindow

Sintassi di chiamata della funzione

```
window = OpenWindow (newWindow)
D0                  A0
```

Scopo della funzione

Questa funzione apre una finestra di Intuition utilizzando i parametri specificati nella struttura NewWindow. Dispone tutti i gadget di sistema nella finestra e alloca la RAM necessaria per creare la struttura Window di gestione della finestra. OpenWindow restituisce l'indirizzo della struttura Window che gestirà la nuova finestra.

Argomenti della funzione

newWindow Indirizzo della struttura NewWindow.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Prima di chiamare la funzione OpenWindow, si deve creare e inizializzare una struttura NewWindow, il cui indirizzo dev'essere poi indicato come argomento della funzione OpenWindow. La struttura NewWindow è molto importante nel sistema Intuition. Si veda anche l'introduzione di questo capitolo.

Quando OpenWindow restituisce il controllo, il task ottiene l'indirizzo della struttura Window creata dalla funzione e la finestra appare sullo schermo. A questo punto si può liberare la memoria allocata per la struttura NewWindow.

I flag della struttura NewWindow

Il parametro Flags nella struttura NewWindow include i seguenti flag:

- WINDOWSIZING. Se questo flag è impostato, la finestra viene dotata del gadget di ridimensionamento. L'utente può quindi ridimensionarla impiegando il mouse. Il gadget di ridimensionamento si trova nell'angolo inferiore destro della finestra. Se è impostato il flag SIZEBRIGHT (il valore di default), il gadget viene considerato parte del bordo destro. Se invece è impostato il flag SIZEBOTTOM, viene considerato parte del bordo inferiore.

- WINDOWDEPTH. Se questo flag è impostato, nella finestra compaiono i gadget per ordinare la profondità della finestra rispetto alle altre.

- WINDOWCLOSE. Se questo flag è impostato, nella finestra compare il gadget di chiusura.

- WINDOWDRAG. Se questo flag è impostato, la barra titolo della finestra diviene un gadget di trascinamento.

- GIMMEZEROZERO. Se questo flag è impostato, la finestra risulta di tipo gimmezerozero, cioè dotata di un sistema di coordinate che permette di disegnare soltanto al suo interno, senza il pericolo di sovrascrivere la barra titolo e i gadget standard.

- SIMPLE_REFRESH. Se questo flag è impostato, ogni volta che torna in vista una parte nascosta della finestra, il programma deve occuparsi di ricostruirla.

- SMART_REFRESH. Se questo flag è impostato, viene attivato il modo di refresh avanzato per le operazioni di ripristino delle parti danneggiate della finestra.

- SUPER_BITMAP. Se questo flag è impostato, la finestra utilizza il refresh a superbitmap.

- BACKDROP. Se questo flag è impostato, la finestra risulta di tipo backdrop.

- REPORTMOUSE. Se questo flag è impostato, i movimenti del mouse all'interno della finestra possono essere riferiti al programma se i corrispondenti flag IDCMP sono attivati.

- BORDERLESS. Se questo flag è impostato, la finestra viene creata senza bordi.

- ACTIVATE. Se questo flag è impostato, la finestra risulta attiva al momento dell'apertura.

- NOCAREREFRESH. Se questo flag è impostato, il programma non riceve alcun messaggio che lo avvisi quando sezioni nascoste della finestra tornano visibili.

- ACTIVEWINDOW. Se questo flag è impostato, il programma riceve un messaggio quando la finestra diventa attiva.

- INACTIVEWINDOW. Se questo flag è impostato, il programma riceve un messaggio quando la finestra cessa di essere attiva.

OpenWorkBench

Sintassi di chiamata della funzione

```
open = OpenWorkBench ()
DØ
```

Scopo della funzione

Questa funzione apre lo schermo Workbench (che può essere stato chiuso con una chiamata alla funzione CloseWorkBench). Se la chiamata alla funzione ha successo, la funzione restituisce il valore TRUE, altrimenti il valore FALSE.

Argomenti della funzione

Questa funzione non richiede argomenti.

Discussione

Nella libreria Intuition ci sono quattro funzioni che agiscono direttamente con lo schermo Workbench: OpenWorkBench, CloseWorkBench, WBenchToBack e WBenchToFront.

Con il nome Workbench ci si riferisce in genere sia allo schermo principale di Intuition, sia all'applicazione che tramite menu, finestre e icone permette d'interagire molto intuitivamente con i disk drive, i dischi, le directory e i file. Sebbene il *Workbench* (l'applicazione) agisca sull'omonimo schermo, questo schermo è presente anche prima che venga mandata in esecuzione l'applicazione, ed è da essa completamente indipendente.

Al momento, lo schermo Workbench è il solo schermo standard di Intuition. Si tratta di uno schermo in alta risoluzione con o senza interlace (a seconda delle impostazioni di Preferences). I colori di default sono il blu per lo sfondo, il bianco e il nero per i dettagli e l'arancio per il cursore del mouse.

Lo schermo Workbench è a disposizione di qualsiasi task. Per esempio, l'AmigaDOS (tramite CLI) e il programma *Workbench* utilizzano lo schermo Workbench. Se il *Workbench* è attivo, lo schermo Workbench mostra sia le finestre del programma sia tutte le altre finestre aperte dai task. Non si è quindi obbligati ad aprire uno schermo personalizzato per le finestre dei propri programmi, dal momento che lo schermo Workbench viene aperto per default

ed è sempre disponibile. Basta impostare il tipo di schermo a WORKBENCH-SCREEN nella struttura NewWindow quando si definisce una nuova finestra.

I programmi che utilizzano interfacce utente a caratteri sono particolarmente efficienti se tutte le finestre vengono aperte sullo schermo Workbench. In questo modo, infatti, l'utente è in grado di lavorare senza la confusione prodotta da schermi che cambiano con il cambiare delle finestre. Inoltre il fatto che il programma non debba aprire nessuno schermo personalizzato produce un ovvio risparmio di memoria. Nonostante questo, talvolta i programmi sono costretti ad aprire schermi personalizzati per scopi particolari, come ad esempio per raggiungere un'elevata velocità di scroll dei caratteri con gli schermi a un solo bitplane (due colori).

La funzione OpenWorkBench apre semplicemente lo schermo Workbench. Se non esiste abbastanza spazio RAM, la chiamata alla funzione è destinata a fallire.

Se all'atto della chiamata il programma *Workbench* è attivo, sullo schermo si aprono automaticamente tutte le sue finestre e icone. Ovviamente, questo vale anche per tutti gli altri task che abbiano aperto finestre su quello schermo.

PrintIText

Sintassi di chiamata della funzione

```
PrintIText (rastPort, intuiText, leftEdge, topEdge)
            A0        A1         D0        D1
```

Scopo della funzione

Questa funzione disegna i testi definiti dalla struttura IntuiText indicata dal task nell'argomento intuiText. PrintIText non restituisce alcun valore.

Argomenti della funzione

rastPort	Indirizzo della struttura rastPort nella cui bitmap si desidera visualizzare il testo.
intuiText	Indirizzo della struttura IntuiText che definisce il testo.

leftEdge Offset che viene aggiunto alla variabile LeftEdge della struttura IntuiText per definire la posizione orizzontale iniziale dei caratteri.

topEdge Offset che viene aggiunto alla variabile TopEdge della struttura IntuiText per definire la posizione verticale iniziale dei caratteri.

Discussione

Ci sono due funzioni della libreria Intuition che riguardano direttamente i testi nel sistema Amiga: IntuiTextLength e PrintIText. Queste funzioni operano con la struttura IntuiText, che contiene una rappresentazione del testo. Il parametro IText nella struttura IntuiText punta alla stringa di testo a terminazione nulla destinata a essere visualizzata.

IntuiTextLength e PrintIText consentono rispettivamente di ottenere il numero di pixel in orizzontale richiesti dal testo indicato come argomento (tenendo conto della fonte-carattere selezionata), e di visualizzare un testo all'interno di un elemento video contenitore.

La struttura IntuiText fornisce un modo semplice per presentare stringhe di testo in ambiente Intuition sullo schermo dell'Amiga. Nel sistema Intuition il miglior esempio è l'array di stringhe IText usato per i menu.

Per definire e presentare un testo vi sono alcune scelte preliminari da fare:

- i colori per il testo.
- I colori per lo sfondo del testo.
- La posizione iniziale del testo nell'elemento video contenitore (o all'esterno, nel caso di un gadget).
- La fonte-carattere da impiegare, che può essere quella di default (Topaz-60 oppure Topaz-80) o qualsiasi altra.
- L'eventuale creazione di una lista concatenata di strutture IntuiText per conservarvi più stringhe di testo.

Per definire i colori si possono scegliere tre modi di disegno. Con JAM1, i caratteri vengono disegnati con il colore FrontPen; l'area di fondo intorno ai caratteri non subisce variazioni. JAM1 è anche chiamato modo a sovraimpressione (overstrike).

Con JAM2, l'immagine del carattere viene disegnata con il colore FrontPen, mentre l'area di fondo intorno ai caratteri viene disegnata con il colore BackPen. Usando questo modo, si cambiano completamente i colori che costituiscono i caratteri di testo e le aree che li circondano.

Con il modo di disegno COMPLEMENT, i caratteri vengono disegnati con

i colori che si ottengono complementando i loro pixel. Per esempio, se un pixel ha tutti i corrispondenti bit dei vari bitplane a 1, si ottiene il colore relativo a tutti i bit a 0. Le impostazioni di FrontPen e BackPen presenti nella struttura IntuiText vengono ignorate.

RefreshGadgets

Sintassi di chiamata della funzione

```
RefreshGadgets (gadgets, window, requester)
                A0       A1      A2
```

Scopo della funzione

Questa funzione effettua il refresh di tutti i gadget di una lista, iniziando da quello specificato. RefreshGadgets non restituisce alcun valore.

Argomenti della funzione

gadgets	Indirizzo della struttura Gadget dalla quale la funzione deve iniziare il refresh, proseguendo poi con tutti i gadget che seguono.
window	Indirizzo della struttura Window.
requester	Indirizzo della struttura Requester; dev'essere specificato se la lista di gadget appartiene a un requester.

Discussione

Ci sono sei funzioni della libreria Intuition che operano con i gadget: AddGadget, OnGadget, OffGadget, RefreshGadgets, RemoveGadget e ModifyProp. Si vedano anche le spiegazioni relative a queste funzioni.

Ci sono due ragioni fondamentali per usare la funzione RefreshGadgets. Prima di tutto per mostrare la nuova immagine dei gadget dopo qualche modifica. In secondo luogo, per procedere al refresh nel caso che qualche operazione sui gadget ne abbia danneggiato la presentazione.

RefreshGList

Sintassi di chiamata della funzione

RefreshGList (gadget, window, requester, number_gadgets)
A0 A1 A2 D0

Scopo della funzione

Questa funzione ridisegna sullo schermo un numero prefissato di gadget appartenenti a una lista, iniziando da quello indicato. È utile per il refresh dei gadget in una finestra o in un requester. Questa funzione non restituisce alcun valore.

Argomenti della funzione

gadget Indirizzo della struttura Gadget relativa al primo gadget della lista.

window Indirizzo della struttura Window relativa alla finestra che contiene i gadget.

requester Indirizzo della struttura Requester relativa al requester che contiene i gadget. Per quanto riguarda l'associazione dei gadget ai requester, il parametro REQGADGET nella struttura Gadget dev'essere impostato a 1 in tutti i gadget della lista. Se i gadget non fanno parte di un requester questo argomento s'imposta a NULL.

number_gadgets Numero dei gadget che devono essere sottoposti al refresh, a partire da quello indicato. Se si indica il valore -1, la funzione esegue il refresh su tutti i gadget che seguono quello indicato. In ogni caso, quando la funzione incontra una struttura Gadget nella quale il parametro NextGadget contiene un indirizzo nullo, considera finita la lista.

Discussione

RefreshGList è una delle quattro funzioni che riguardano i gadget e le liste di gadget nel software sistema di Intuition. Si veda anche la spiegazione della funzione RefreshGadgets.

RefreshGList è utile per ridisegnare una serie limitata di gadget in una finestra o in un requester.

RefreshWindowFrame

Sintassi di chiamata della funzione

RefreshWindowFrame (window)
AØ

Scopo della funzione

Questa funzione effettua il refresh del bordo della finestra indicata, compresi i gadget e la barra titolo. Generalmente viene impiegata quando il bordo della finestra è stato danneggiato da altri disegni.

Argomenti della funzione

window	Indirizzo della struttura Window il cui bordo dev'essere ricostruito.

Discussione

Le caratteristiche del bordo di una finestra sono precisate da cinque parametri della relativa struttura Window. Si tratta dei parametri BorderLeft, BorderRight, BorderTop, BorderBottom e BorderRPort. La funzione RefreshWindowFrame adopera questi parametri per determinare come deve eseguire il refresh del bordo della finestra.

RemakeDisplay

Sintassi di chiamata della funzione

RemakeDisplay ()

Scopo della funzione

Questa funzione rigenera l'intero schermo video di Intuition. Per prima cosa chiama la funzione MakeScreen per ogni schermo del sistema. Stabilisce poi le relazioni tra i vari schermi e determina la corretta lista Copper per produrre il nuovo quadro video. RemakeDisplay non restituisce alcun valore.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Ci sono due funzioni della libreria Intuition che operano sull'intero schermo video, ovvero su tutti gli schermi e le finestre che vi compaiono: RemakeDisplay e RethinkDisplay.

La funzione RemakeDisplay consente di ridefinire automaticamente la presentazione di tutte le finestre (layer), schermi (viewport), e view dello schermo video. Quando si chiama RemakeDisplay, per ogni schermo viene chiamata automaticamente la funzione MakeScreen. RemakeDisplay può essere considerata la via più completa per manipolare la presentazione Intuition dell'Amiga.

Lo schermo video dell'Amiga può essere molto complesso. Possono esserci diversi schermi personalizzati e ognuno può avere diverse finestre. Le finestre aperte in uno schermo personalizzato possono anche appartenere a programmi diversi.

Le finestre possono essere di tipo backdrop, superbitmap, senza bordi o gimmezerozero. Inoltre, sullo schermo video può essere contemporaneamente aperto lo schermo Workbench.

A causa della quantità d'informazioni coinvolte, la funzione RemakeDisplay può impiegare diversi millisecondi per portare a termine l'esecuzione. Nel

frattempo il sistema provvede automaticamente a tenere bloccati gli altri task, circondando le istruzioni di chiamata alla funzione RethinkDisplay (chiamata indirettamente da RemakeDisplay) con chiamate alle funzioni di sistema Forbid e Permit.

In un certo senso, MakeScreen, RethinkDisplay e RemakeDisplay possono essere ordinate gerarchicamente. La funzione MakeScreen si trova all'ultimo gradino della gerarchia: riguarda un singolo schermo personalizzato nella presentazione video. Segue la funzione RethinkDisplay che riguarda l'intero video dell'Amiga, in tutta la sua potenziale complessità. RethinkDisplay produce una nuova presentazione di tutte le finestre e gli schermi. Infine c'è la funzione RemakeDisplay, che svolge il lavoro delle funzioni MakeScreen e RethinkDisplay in combinazione; rigenera tutti gli schermi personalizzati e ne crea poi la nuova presentazione video.

Se si vogliono ricostruire soltanto uno o due schermi personalizzati, può essere più efficiente chiamare la funzione MakeScreen per ognuno di essi e chiamare poi la funzione RethinkDisplay. RethinkDisplay prende le nuove definizioni di schermo e le combina con le altre rimaste inalterate (includendo anche lo schermo Workbench) e produce la nuova presentazione. Se il numero complessivo degli schermi è apprezzabilmente più alto, probabilmente questa strada è più rapida del ricorso alla funzione RemakeDisplay.

RemoveGadget

Sintassi di chiamata della funzione

```
gadgetposition = RemoveGadget (window, gadget)
D0                             A0      A1
```

Scopo della funzione

Questa funzione rimuove un gadget da una finestra o da un requester. Se il flag REQGADGET è impostato, l'argomento window si riferisce a una struttura Requester, altrimenti a una struttura Window. RemoveGadget restituisce un numero relativo alla posizione del gadget nella lista. Se l'indirizzo indicato come secondo argomento individua una struttura Gadget che non appartiene alla lista (magari perché la lista è vuota), viene restituito il valore -1.

Argomenti della funzione

window Indirizzo della struttura Window oppure Requester.

gadget Indirizzo della struttura Gadget da rimuovere.

Discussione

Ci sono sei funzioni della libreria Intuition che operano con i gadget: AddGadget, OnGadget, OffGadget, RefreshGadget, RemoveGadget e ModifyProp. Si vedano anche le spiegazioni relative a queste funzioni.

È importante distinguere chiaramente tra "disabilitazione" e "rimozione" di un gadget. Se il gadget viene disabilitato (tramite la funzione OffGadget) rimane comunque nella lista dei gadget. La funzione OnGadget può quindi restituirgli la normale immagine di presentazione. Se invece lo si rimuove (tramite la funzione RemoveGadget), il gadget viene cancellato dalla lista e non è possibile impiegare la funzione OnGadget per ripresentarlo. Per inserirlo nuovamente nel sistema bisogna chiamare AddGadget. Si noti che non è possibile impiegare RemoveGadget con un gadget di sistema.

RemoveGList

Sintassi di chiamata della funzione

```
position = RemoveGList (window, gadget, number_gadgets)
D0                      A0      A1      D0
```

Scopo della funzione

Questa funzione rimuove una sotto-lista di gadget da una lista appartenente a una finestra o a un requester. L'argomento number_gadgets indica alla funzione quanti gadget devono essere rimossi dalla lista, mentre l'argomento gadget indica da quale gadget della lista bisogna iniziare la rimozione. Se quest'ultimo non viene individuato (magari perché la lista è vuota), viene restituito il valore -1.

Argomenti della funzione

window — Indirizzo della struttura Window della finestra o del requester da cui i gadget devono essere rimossi.

gadget — Indirizzo della struttura Gadget del primo gadget da rimuovere.

number_gadgets — Numero dei gadget da rimuovere; se dev'essere rimossa l'intera lista a partire dal gadget indicato, si deve inserire il valore -1.

Discussione

RemoveGList è una delle quattro funzioni che riguardano i gadget e le liste di gadget nel software sistema di Intuition.

Le strutture Gadget che definiscono i gadget nelle finestre e nei requester sono mantenute in opportune liste (si veda anche la spiegazione della funzione AddGList). RemoveGList consente di rimuovere una lista di gadget da una finestra o da un requester appartenente a una finestra. In un certo senso, la funzione RemoveGList costituisce l'inverso della funzione AddGList.

La prima struttura Gadget indica se la sotto-lista di gadget specificata appartiene a una finestra o a un requester. Infatti, se in quella prima struttura il flag REQGADGET risulta impostato, significa che tutti i gadget della sotto-lista appartengono a un requester e l'argomento window viene considerato come un puntatore a una struttura Requester.

ReportMouse

Sintassi di chiamata della funzione

```
ReportMouse (report, window)
             DØ      AØ
```

Scopo della funzione

Questa funzione ordina a Intuition di segnalare al task o al programma i movimenti del mouse in una finestra.

Argomenti della funzione

report	Valore TRUE o FALSE che specifica se gli spostamenti del mouse devono essere riferiti oppure no.
window	Indirizzo della struttura Window.

Discussione

Ci sono dodici funzioni della libreria Intuition che operano direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Sia le finestre che i gadget hanno bisogno di avere informazioni sui movimenti del mouse. La funzione ReportMouse consente di tenere sotto controllo il flusso d'informazioni sui movimenti del mouse nelle finestre. In particolare, ogni volta che il puntatore del mouse si sposta in un programma applicativo o in un gadget si può fare in modo che Intuition informi il task circa gli spostamenti del mouse. Per essere sicuri che ciò accada, basta chiamare la funzione ReportMouse immediatamente dopo la chiamata a OpenWindow. Il programma diventa così in grado d'interpretare correttamente i movimenti del mouse e quindi, per esempio, di spostare la finestra.

Quando si chiama la funzione ReportMouse (con l'argomento report impostato a TRUE), la finestra specificata può essere quella attiva oppure no. Nel primo caso gli spostamenti del mouse vengono riferiti con inizio immediato, altrimenti si deve attendere che la finestra diventi attiva.

Sia la struttura NewWindow che la struttura Gadget hanno un flag per rilevare gli spostamenti del mouse. La struttura NewWindow ha il flag REPORTMOUSE; la struttura Gadget ha FOLLOWMOUSE. Se il flag REPORTMOUSE è impostato, dal momento in cui la finestra viene aperta i movimenti del mouse vengono riferiti al programma; la funzione ReportMouse, tuttavia, può alterarne l'impostazione. Se è impostato il flag FOLLOWMOUSE, il programma riceve gli opportuni messaggi sullo spostamento del mouse all'interno del gadget; ReportMouse non può alterare tale impostazione.

Se si chiama ReportMouse quando un gadget è selezionato, gli spostamenti del mouse verranno riferiti soltanto finché dura la condizione di selezionato. Una volta che il gadget cessa di essere selezionato, il flag FOLLOWMOUSE della struttura Gadget viene esaminato ancora una volta. Se non risulta impostato, i movimenti del mouse all'interno del gadget ricominceranno a essere trasmessi soltanto quando il gadget verrà nuovamente selezionato.

Se si chiama ReportMouse quando nessun gadget è selezionato, s'inverte lo stato del flag REPORTMOUSE della finestra; tuttavia ciò non ha effetto sui gadget selezionati in seguito.

È importante rendersi conto che il flag REPORTMOUSE, alterato da questa funzione, si limita a permettere che i movimenti del mouse vengano trasmessi alla finestra. Se e come ciò avviene, è determinato dai flag IDCMP MOUSEMOVE e DELTAMOVE.

Request

Sintassi di chiamata della funzione

```
reqopen = Request (requester, window)
D0                A0        A1
```

Scopo della funzione

Questa funzione attiva un requester in una finestra. Se l'operazione riesce, la funzione Request restituisce il valore TRUE; se il requester non può essere aperto, la funzione restituisce FALSE.

Argomenti della funzione

requester	Indirizzo della struttura Requester.
window	Indirizzo della struttura Window destinata a ricevere il requester.

Discussione

Nella libreria Intuition ci sono sette funzioni impiegate nella gestione dei requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest, InitRequester, Request e SetDMRequest. Si vedano anche le spiegazioni relative a queste funzioni.

AutoRequest, BuildSysRequest, InitRequester e SetDMRequest preparano semplicemente un requester per la presentazione sul video. La funzione Request è l'unica che visualizza effettivamente il requester nella finestra.

Per usare appropriatamente un requester in un programma sono necessarie le seguenti operazioni:

1. Si deve allocare una struttura Requester.

2. Si deve inizializzare la struttura Requester con una chiamata alla funzione InitRequester.

3. Si devono specificare i gadget, il testo, i confini e le immagini che costituiscono il requester.

4. Se si sta adoperando il canale IDCMP per l'input del requester, è necessario decidere se si vuole fare uso delle speciali funzioni fornite dal sistema IDCMP.

5. Si deve chiamare la funzione Request oppure la funzione SetDMRequest per ottenere la presentazione del requester.

Per inizializzare una struttura Requester si devono seguire i seguenti passi:

1. Inizializzare la struttura Requester con una chiamata alla funzione InitRequester (ciò imposta la maggior parte delle variabili della struttura Requester ai loro valori di default, che in diversi casi saranno valori NULL).

2. Preparare una lista dei gadget. Il requester può gestire una lista concatenata di gadget. Si faccia attenzione a non specificare gadget che si estendano oltre i confini del requester. Se s'impiegano gadget personalizzati, progettati su una bitmap particolare, ci si deve assicurare della coerenza tra rettangoli di selezione dei gadget e immagini personalizzate per il requester.

3. Se si tratta di un requester personalizzato, con una bitmap propria, va predisposta la corrispondente struttura BitMap.

Se si sta adoperando il sistema IDCMP come metodo di I/O per la finestra (si veda, a questo proposito, la funzione ModifyIDCMP), si possono utilizzare alcuni flag IDCMP per aggiungere precisazioni ai requester progettati. Vi sono tre flag IDCMP particolarmente importanti:

- REQVERIFY. Se il flag è impostato a 1, controlla che il programma sia pronto ad accogliere un requester in una finestra. Quando un programma riceve un messaggio REQVERIFY, il requester non appare nella finestra finché il programma non risponde al messaggio in questione.

- REQSET. Se il flag è impostato a 1, il programma riceve un messaggio quando il primo requester appare nella finestra.

- REQCLEAR. Se il flag è impostato a 1, il programma riceve un messaggio quando l'ultimo requester sparisce dalla finestra.

Si possono impostare questi flag sia quando si crea una struttura NewWindow sia chiamando la funzione ModifyIDCMP.

Il sistema può rilevare le informazioni pixel per la presentazione di un requester in due modi. Il primo metodo richiede che vengano fornite le informazioni necessarie per ricorrere al requester interno di Intuition. Il secondo richiede che venga fornita in RAM un'immagine completa della bitmap del requester. L'impostazione della struttura Requester dipende da quale metodo s'intende usare.

Per presentare un requester in ambiente Intuition, si deve predisporre una lista di gadget, un colore di penna per il riempimento dello sfondo del requester e una o più strutture IntuiText e Border per definire il testo e i bordi del requester. Se si vuole definire un proprio requester attraverso una bitmap personalizzata, si devono definire anche tutte le immagini per i gadget.

RethinkDisplay

Sintassi di chiamata della funzione

RethinkDisplay ()

Scopo della funzione

Questa funzione agisce in coppia con MakeScreen per produrre una nuova presentazione sullo schermo dell'Amiga.

Argomenti della funzione

Questa funzione non ha ha argomenti.

Discussione

Ci sono due funzioni della libreria Intuition che operano sull'intero schermo video, ovvero su tutti gli schermi e le finestre che vi compaiono: RemakeDisplay e RethinkDisplay.

La funzione RethinkDisplay consente a Intuition di ridefinire le liste Copper necessarie per guidare l'hardware, in vista della definizione del quadro video, combinando in questo modo tutti gli schermi che sono stati precedentemente definiti.

Nel sistema video multi-schermo dell'Amiga si possono avere, in una stessa schermata, diversi schermi personalizzati insieme allo schermo Workbench.

Ogni volta che un processo di disegno altera le caratteristiche della viewport di uno schermo personalizzato (come la risoluzione orizzontale o verticale), si devono visualizzare queste nuove informazioni. Per farlo si possono seguire due strade. Primo, si cambiano alcune o tutte le definizioni dello schermo personalizzato e si chiama la funzione RemakeDisplay, che rileva le nuove definizioni, le unisce a quelle rimaste inalterate, e realizza una lista di istruzioni Copper per definire la nuova presentazione.

Secondo, si possono cambiare alcuni degli schermi personalizzati chiamando per ognuno la funzione MakeScreen. Si può poi chiamare RethinkDisplay per creare la nuova presentazione. Avendo un discreto numero di schermi e dovendone aggiornare soltanto un paio, questa strada risulterà probabilmente più veloce che non chiamare la funzione RemakeDisplay.

A causa della quantità d'informazioni coinvolte, la funzione RethinkDisplay può impiegare diversi millisecondi per portare a termine l'esecuzione. Nel frattempo il sistema provvede automaticamente a tenere bloccati gli altri task, circondando le istruzioni di chiamata alla funzione RethinkDisplay con chiamate alle funzioni di sistema Forbid e Permit.

ScreenToBack

Sintassi di chiamata della funzione

ScreenToBack (screen)
A0

Scopo della funzione

Questa funzione sposta uno schermo personalizzato al livello più profondo nella presentazione video.

Argomenti della funzione

screen Indirizzo della struttura Screen.

Discussione

Nella libreria Intuition ci sono sette funzioni che operano direttamente con gli schermi video personalizzati: OpenScreen, CloseScreen, MoveScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront. Si vedano anche le spiegazioni relative a queste funzioni.

Quando s'impiega una presentazione Intuition multi-strato (o multi-layer), l'ordine di apparizione degli elementi video in un dato momento dipende in parte dall'ordine con cui gli schermi e le finestre vengono aperti e in parte dalle operazioni di ordinamento di profondità effettuate dall'utente o dai task.

In ogni caso, l'ordine degli schermi e delle finestre segue queste regole:

1. Se ci sono finestre di tipo backdrop in uno schermo della presentazione video, la parte più profonda di quello schermo sarà una finestra backdrop. La presenza di finestre backdrop in uno schermo dipende dall'impostazione del flag BACKDROP nella struttura NewWindow. Sebbene in uno schermo possa esserci più di un layer di finestre backdrop, non esiste in genere alcun motivo utile per crearne più di una. Se si dispone di più di una finestra backdrop in uno schermo, l'ordine di sovrapposizione segue l'ordine con cui sono state create (aperte). La finestra backdrop più profonda è quella che è stata creata per ultima.

2. Ci sono poi i vari schermi personalizzati. Il loro numero è limitato soltanto dalla disponibilità di memoria chip. Ciascuno di questi schermi personalizzati può avere diverse finestre aperte, alcune delle quali possono ancora una volta essere finestre backdrop. Ogni finestra aperta, di qualunque tipo, è per il sistema un terminale virtuale. Quando è attiva, tutte le operazioni di I/O provenienti dal mouse, dalla tastiera o da altri dispositivi di I/O, passano attraverso di lei. Le finestre aperte su uno schermo personalizzato possono appartenere anche a più di un programma.

3. Infine c'è lo schermo Workbench, attualmente l'unico schermo standard disponibile. In ogni schermo standard (in futuro potrebbero essercene altri), si può aprire un numero di finestre limitato solo della disponibilità di memoria. Programmi diversi possono condividere lo stesso schermo standard per aprire e presentare le loro finestre.

   Si noti che l'ordine di profondità degli schermi e delle finestre cambia a seconda degli interventi effettuati dall'utente o dai programmi.

ScreenToFront

Sintassi di chiamata della funzione

ScreenToFront (screen)
A0

Scopo della funzione

Questa funzione porta un determinato schermo personalizzato al primo livello di profondità nella presentazione video.

Argomenti della funzione

screen	Indirizzo della struttura Screen.

Discussione

Nella libreria Intuition ci sono sette funzioni che operano direttamente con gli schermi video personalizzati: OpenScreen, CloseScreen, MoveScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront. Si vedano anche le spiegazioni relative a queste funzioni.

Gli schermi personalizzati possono essere riordinati in profondità facendo uso delle funzioni ScreenToFront e ScreenToBack. Questo tipo d'intervento sull'ordine di profondità degli schermi si aggiunge al riordinamento relativo alle finestre; si tratta di due meccanismi separati e distinti per variare l'ordine di profondità.

Le funzioni ScreenToFront e ScreenToBack si possono impiegare, per esempio, se l'utente effettua una scelta di menu che richiede una nuova presentazione di schermo. A questo scopo si può spostare più in profondità uno schermo e richiamarne un altro in primo piano. Si noti che per lo schermo Workbench sono disponibili due funzioni specifiche per i cambiamenti del livello di profondità: WBenchToBack e WBenchToFront.

SetDMRequest

Sintassi di chiamata della funzione

```
reqdisplay = SetDMRequest (window, dMRequester)
DØ                         AØ      A1
```

Scopo della funzione

Questa funzione abbina alla finestra indicata un particolare requester double-click. Il requester double-click è uno speciale requester che Intuition visualizza nella finestra in seguito alla doppia pressione del pulsante destro del mouse. SetDMRequest non ha effetto se il requester double-click è già visualizzato nella finestra. La funzione restituisce il valore TRUE se nella finestra non c'era nessun requester double-click visualizzato, altrimenti restituisce FALSE.

Argomenti della funzione

window	Indirizzo della struttura Window.
dMRequester	Indirizzo della struttura Requester relativa al requester double-click.

Discussione

Nella libreria Intuition ci sono sette funzioni che agiscono direttamente con i requester: AutoRequest, BuildSysRequest, ClearDMRequest, FreeSysRequest, InitRequester, Request e SetDMRequest. Si vedano anche le spiegazioni relative a queste funzioni.

I requester double-click, come gli altri, bloccano l'input nella finestra dove compaiono. Per questo, tutti i requester devono avere almeno un gadget che ne consenta la rimozione perché l'utente possa proseguire con il proprio input.

Ogni gadget destinato a concludere l'interazione utente-requester può avere il flag ENDGADGET impostato nella struttura Gadget. Infatti, ogni volta che viene selezionato il gadget di un requester, Intuition esamina il flag ENDGADGET della struttura Gadget, e se lo trova impostato a 1 cancella il requester dalla finestra e lo toglie dalla relativa lista concatenata.

SetMenuStrip

Sintassi di chiamata della funzione

SetMenuStrip (window, menu)
A0 A1

Scopo della funzione

Questa funzione associa a una finestra una barra menu. La barra menu compare sullo schermo quando la finestra viene attivata e l'utente preme il pulsante destro del mouse.

Argomenti della funzione

window	Indirizzo della struttura Window.
menu	Indirizzo della prima struttura Menu relativa alla barra menu.

Discussione

Ci sono cinque funzioni nella libreria Intuition che operano direttamente con i menu: ClearMenuStrip, ItemAddress, OnMenu, OffMenu e SetMenuStrip. Si vedano anche le spiegazioni relative a queste funzioni.

La barra menu è lo strumento che contiene tutte le informazioni necessarie per definire e impostare menu interattivi. Le barre menu di Intuition funzionano nel modo seguente: premendo il pulsante destro del mouse appaiono alcune intestazioni, ognuna delle quali si apre portandovi sopra il puntatore del mouse. In tal modo appare una serie di voci che possono essere predisposte per aprire altre serie di voci, dette sotto-voci.

Ci sono due strutture impiegate direttamente dai menu e dalle loro voci: le strutture Menu e MenuItem. Una barra menu non è altro che una lista concatenata di strutture Menu. La struttura Menu contiene i dati per definire l'unità fondamentale della barra menu e quelli per definire l'intestazione delle voci di menu a essa sottesa (compreso il nome del menu). Si noti che Intuition non considera la semplice selezione di un menu come un evento: perché si

possa parlare di "evento" l'utente deve anche selezionare una voce.

Ecco la procedura generale che si dovrebbe seguire per progettare una barra menu associata a una finestra:

1. Si costruiscono le strutture Menu concatenandole insieme. A ognuna si collega una lista di strutture MenuItem per definire le voci del menu, e da queste ultime si dirama eventualmente un'altra lista di strutture MenuItem per definire le sotto-voci. Si noti che per individuare tutta questa complessa ramificazione di strutture è sufficiente conoscere l'indirizzo della prima struttura Menu della barra.

2. Si sottopone la barra menu a Intuition tramite la funzione SetMenuStrip; ciò collega la barra menu alla finestra indicata.

3. Si organizza il programma per metterlo in grado di rispondere ai messaggi di selezione dei menu di Intuition (si veda, a questo proposito, la spiegazione della funzione ModifyIDCMP).

Per ciascuna struttura Menu si devono effettuare alcune scelte preliminari.

- I nomi di menu destinati ad apparire nella barra titolo dello schermo.
- Le voci destinate ad apparire quando l'utente apre il menu.
- La posizione di ogni voce nella lista corrispondente.
- L'immagine grafica o il testo relativi a ciascuna voce di menu.
- Il metodo per evidenziare ciascuna voce di menu, nel momento in cui l'utente vi porta sopra il puntatore del mouse.
- L'eventuale combinazione di tasti da associare a ciascuna voce.
- Infine, l'eventuale associazione di ogni voce a una serie di sotto-voci; si ricordi che per ciascuna sotto-voce devono essere prese le stesse decisioni illustrate per le voci dei menu.

La struttura Menu

La struttura Menu contiene le seguenti informazioni:

- il testo della barra menu che appare lungo la barra titolo dello schermo quando viene premuto il pulsante destro del mouse.
- La posizione del testo nella barra menu.

- Il puntatore alla successiva struttura Menu nella lista concatenata che contribuisce a definire la barra menu.

- Il puntatore alla prima struttura della lista concatenata di strutture MenuItem che contribuisce a definire la barra menu.

La struttura MenuItem

La struttura MenuItem contiene le seguenti informazioni:

- la posizione della voce all'interno del rettangolo che contiene il menu.

- Un puntatore a un testo o a un'immagine grafica.

- L'identificazione del metodo di evidenziazione da impiegare quando l'utente seleziona la voce.

- La combinazione di tasti che l'utente può impiegare per selezionare la voce senza usare il mouse.

- La definizione del rettangolo di selezione: viene usato per rilevare la selezione della voce e per l'evidenziazione.

- L'indicazione delle voci di menu che risultano mutuamente esclusive con la selezione della voce considerata.

- Il puntatore alla prima sotto-voce di una lista concatenata.

- Il numero della successiva voce selezionata; quando viene selezionata più di una voce, questo campo fornisce un legame tra esse.

Ciascuna sotto-voce di una voce di menu impiega la stessa forma di struttura dati, con alcune minime differenze: la posizione della sotto-voce è relativa al rettangolo di selezione della voce e non a quello del menu, e il parametro di legame della sotto-voce con altre voci di livello inferiore viene ignorato.

SetPointer

Sintassi di chiamata della funzione

SetPointer (window, pointer, height, width, XOffset, YOffset)
A0 A1 D0 D1 D2 D3

Scopo della funzione

Questa funzione associa a una finestra un puntatore personalizzato. Se la funzione ha successo, quando la finestra diventa attiva il puntatore del mouse appare con la nuova immagine. Se la finestra è già attiva quando SetPointer viene chiamata, il nuovo puntatore viene visualizzato immediatamente.

Argomenti della funzione

window	Indirizzo della struttura Window.
pointer	Indirizzo dei dati che costituiscono l'immagine.
height	Altezza dello sprite relativo al puntatore.
width	Larghezza dello sprite relativo al puntatore.
XOffset	Posizione dello sprite rispetto alla coordinata x del punto di selezione.
YOffset	Posizione dello sprite rispetto alla coordinata y del punto di selezione.

Discussione

Nella libreria Intuition ci sono due funzioni che operano con la rappresentazione del puntatore. La prima consente di associare a una finestra un particolare disegno del puntatore, e l'altra di ripristinare il disegno di default. Con queste funzioni si possono creare immagini del puntatore diverse dal consueto disegno della freccia disponibile per default in ambiente Intuition. Per definire il puntatore dev'essere impostata la struttura SpriteImage illustrata nel capitolo 3.

Una volta che il puntatore personalizzato è stato impostato, è possibile cambiarne i colori in qualsiasi momento usando la funzione SetRGB4, come viene spiegato nel capitolo 2. Si noti che il puntatore di Intuition è sempre lo sprite 0.

Vediamo come ci si deve comportare per sostituire con un puntatore personalizzato quello di Intuition. Per prima cosa si crea una struttura SpriteImage, ovvero un insieme di word di dati che definiscono i colori di ciascun pixel dello sprite usato per il puntatore personalizzato. Le prime due word e le ultime due sono azzerate, le altre definiscono l'aspetto del puntatore.

Si chiama poi la funzione SetPointer. Se la finestra è attiva, il nuovo puntatore vi appare immediatamente. Le variabili XOffset e YOffset, impiegate nella funzione SetPointer, sono adoperate per spostare l'angolo superiore sinistro del puntatore personalizzato rispetto a quella che Intuition considera la posizione del suo punto di selezione. Intuition misura lo spostamento partendo dal centro del suo puntatore (chiamato *hot spot*, punto caldo), ovvero dalla posizione 7,7.

Se per esempio XOffset e YOffset vengono specificati come 0,0, l'angolo superiore sinistro del puntatore personalizzato verrà a coincidere con il punto di selezione. Se si specifica per XOffset e YOffset il valore -7, il puntatore personalizzato risulterà centrato sulla posizione del punto di selezione. Se infine si specifica per XOffset e YOffset il valore -15, l'angolo inferiore destro del puntatore personalizzato verrà a coincidere con il punto di selezione.

SetPrefs

Sintassi di chiamata della funzione

```
prefBuffer = SetPrefs (prefBuffer, dimensione, inform)
DØ                     AØ          DØ          D1
```

Scopo della funzione

Questa funzione imposta i nuovi parametri della struttura Preferences secondo quanto indicato dal parametro prefBuffer. Se il parametro inform risulta impostato, tutte le finestre con il flag IDCMP NEWPREFS attivo vengono informate del cambiamento.

Argomenti della funzione

prefBuffer — Puntatore a una struttura Preferences contenente i nuovi parametri da inserire nella struttura Preferences di sistema usata da Intuition.

dimensione — Il numero di byte da alterare, a partire dall'inizio della struttura. Dal momento che i parametri più utilizzati sono raggruppati all'inizio della struttura Preferences, questo parametro in genere permette un certo risparmio di memoria, definendo una struttura Preferences di dimensioni ridotte.

inform — Valore booleano che indica a Intuition se deve comunicare l'avvenimento a tutto il sistema inviando messaggi di classe NEWPREFS a tutte le finestre che abbiano impostato a 1 il relativo flag IDCMP.

Discussione

Questa funzione della libreria Intuition serve a modificare la struttura Preferences e a porre immediatamente in atto tutti i cambiamenti apportati. È buona norma, in generale, che il parametro inform sia impostato a TRUE (vero) in modo che tutti i task interessati ricevano la notizia dell'avvenuta modifica tramite un messaggio di classe NEWPREFS alle message port delle loro finestre. In tal modo potranno poi, se lo ritengono opportuno, ottenere una copia delle nuove impostazioni tramite la funzione GetPrefs, e agire di conseguenza.

Per una descrizione della struttura Preferences e dei parametri in essa contenuti si veda la spiegazione della funzione GetDefPrefs.

SetWindowTitles

Sintassi di chiamata della funzione

```
SetWindowTitles (window, windowTitle, screenTitle)
                 A0      A1           A2
```

Scopo della funzione

Questa funzione imposta il testo dei titoli della finestra, sia quello che compare nella finestra sia quello che compare nella barra titolo dello schermo. Se gli argomenti windowTitle o screenTitle sono impostati a -1, i titoli in uso rimangono inalterati; impostandoli a 0, alla finestra non vengono associati titoli.

Argomenti della funzione

window	Indirizzo della struttura Window.
windowTitle	Indirizzo della stringa a terminazione nulla che costituisce il titolo della finestra.
screenTitle	Indirizzo della stringa a terminazione nulla che costituisce il titolo dello schermo quando la finestra è selezionata.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Il titolo di finestra appare all'interno della barra titolo della finestra. Il titolo di schermo appare nella barra titolo dello schermo ogni volta che la finestra considerata diventa attiva.

Quando la funzione SetWindowTitles restituisce il controllo, il titolo della finestra viene immediatamente cambiato. Comunque non è necessario che la finestra sia attiva per chiamare SetWindowTitles: nel caso che non lo sia, il titolo dello schermo cambia soltanto quando la finestra viene attivata.

A questo proposito si vedano anche l'introduzione a questo capitolo e la spiegazione della funzione ShowTitle.

SHIFTITEM

Sintassi di chiamata della macro

SHIFTITEM (menuNumber)

Scopo della macro

Questa macro svolge un'operazione di AND logico tra il valore indicato dal task nell'argomento menuNumber e il valore esadecimale 0x3F per azzerare tutti i bit fuorché i sei meno significativi; fa poi scorrere il risultato di cinque bit a sinistra. In altre parole, viene compiuta un'operazione di shift sul numero di menu in uso, al fine di definire un nuovo valore per il numero della voce di menu.

Argomenti della macro

menuNumber Valore a sedici bit che contiene, in formato compresso, i dati destinati a definire il numero di menu, il numero della voce di menu e il numero della sotto-voce.

Discussione

Le macro che riguardano i menu, le voci di menu e le sotto-voci di menu sono sei: MENUNUM, ITEMNUM, SUBNUM, SHIFTMENU, SHIFTITEM e SHIFTSUB. Si vedano anche le spiegazioni relative a queste macro.

Per una completa definizione della macro SHIFTITEM si veda l'ultima parte del file INCLUDE intuition.h.

SHIFTMENU

Sintassi di chiamata della macro

SHIFTMENU (menuNumber)

Scopo della macro

Questa macro estrae dall'argomento menuNumber il numero d'ordine del menu, che risulta compreso tra 0 e 31 (cinque bit). Il numero risultante viene sostituito alla macro. Tale operazione è identica a quella svolta dalla macro MENUNUM. In effetti, queste due macro sono del tutto intercambiabili. SHIFTMENU è stata inclusa soltanto al fine di avere per i menu un equivalente delle macro SHIFTITEM e SHIFTSUB.

Argomenti della macro

menuNumber Valore a sedici bit che contiene, in formato compresso, i dati destinati a definire il numero di menu, il numero della voce e il numero della sotto-voce.

Discussione

Si veda la spiegazione della macro MENUNUM. La precisa definizione della macro SHIFTMENU si può trovare nell'ultima parte del file INCLUDE intuition.h.

SHIFTSUB

Sintassi di chiamata della macro

SHIFTSUB (menuNumber)

Scopo della macro

Questa macro svolge un'operazione di AND logico tra il valore menuNumber e il valore esadecimale 0x1F, così da azzerare tutti i bit tranne i cinque meno significativi. Poi fa scorrere il risultato di 11 bit a sinistra. In altre parole, viene compiuta un'operazione di shift sul numero di menu, al fine di definire un nuovo valore per il numero della sotto-voce di menu.

Argomenti della macro

menuNumber Valore a sedici bit che contiene, in formato compresso, i dati destinati a definire il numero di menu, il numero della voce e il numero della sotto-voce.

Discussione

Le macro che riguardano i menu, le voci di menu e le sotto-voci di menu sono sei: MENUNUM, ITEMNUM, SUBNUM, SHIFTMENU, SHIFTITEM e SHIFTSUB. Si vedano anche le spiegazioni a esse relative.

La precisa definizione della macro SHIFTSUB si può trovare nell'ultima parte del file INCLUDE intuition.h.

ShowTitle

Sintassi di chiamata della funzione

```
ShowTitle (screen, showIt)
           A0      D0
```

Scopo della funzione

Questa funzione serve per presentare o nascondere la barra titolo dello schermo, quando in esso viene aperta una finestra backdrop.

Argomenti della funzione

screen Indirizzo della struttura Screen.

showIt Valore booleano TRUE o FALSE; se il task indica TRUE, la barra titolo viene presentata, se indica FALSE non viene presentata.

Discussione

Nella libreria Intuition ci sono sette funzioni che operano direttamente con gli schermi video personalizzati: OpenScreen, CloseScreen, MoveScreen, MakeScreen, ShowTitle, ScreenToBack e ScreenToFront. Si vedano anche le spiegazioni relative a queste funzioni.

Il testo per il titolo dello schermo serve per identificare lo schermo e per sapere qual è la finestra attiva.

Anche se durante l'inizializzazione della struttura NewScreen viene esplicitamente indicato un titolo di schermo, esso può cambiare quando nuove finestre vi vengono aperte. Ogni schermo ha due tipi di titoli che possono essere presentati nella relativa barra titolo:

1. Il titolo di default, che è il titolo indicato dal task nella struttura NewScreen; viene mostrato quando lo schermo appare per la prima volta.

2. Il titolo corrente di lavoro, che è quello associato alla finestra attiva nello schermo, ed è indicato nella struttura Window della finestra attiva.

Il parametro DefaultTitle fa parte della definizione della struttura NewScreen. Si tratta di un puntatore alla stringa di testo a terminazione nulla destinata a essere visualizzata nella barra titolo dello schermo. La struttura NewScreen contiene inoltre un puntatore alla struttura TextAttr di default, che contiene la descrizione della fonte-carattere da impiegare per il testo della barra titolo dello schermo. Si può adoperare la fonte di default di Intuition (Topaz-60 o Topaz-80) oppure predisporre una propria struttura TextAttr. Si veda a questo proposito il capitolo 4.

La barra titolo dello schermo può trovarsi davanti o dietro alle finestre di tipo backdrop: quando l'argomento showIt è TRUE, la barra titolo di schermo viene mostrata davanti a qualsiasi finestra backdrop.

La barra titolo dello schermo viene sempre presentata davanti a qualsiasi finestra backdrop quando si apre lo schermo per la prima volta. Gli altri tipi di finestra appaiono sempre davanti alla barra titolo dello schermo, qualunque sia l'impostazione di ShowTitle.

SizeWindow

Sintassi di chiamata della funzione

```
SizeWindow (window, dx, dy)
             AØ      DØ D1
```

Scopo della funzione

Questa funzione ridimensiona una finestra con i valori indicati dagli argomenti dx e dy.

Argomenti della funzione

window	Indirizzo della struttura Window.
dx	Valore con segno che definisce il ridimensionamento in pixel della finestra sull'asse x.
dy	Valore con segno che definisce il ridimensionamento in pixel della finestra sull'asse y.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Quando si chiama la funzione SizeWindow, la finestra non viene ridimensionata fino a quando Intuition non riceve il successivo evento di input per la finestra in questione; gli eventi si succedono a un ritmo minimo di dieci al secondo e fino a un massimo di cinquanta al secondo. Si può verificare se la finestra è stata ridimensionata impostando il flag IDCMP NEWSIZE (in questo modo Intuition invia un messaggio IDCMP al task dopo il ridimensionamento della finestra).

Si noti che la funzione SizeWindow non controlla la coerenza dei dati. Se quindi si specificano per dx e dy valori che escono dai limiti imposti allo schermo, una parte della finestra viene tagliata.

SUBNUM

Sintassi di chiamata della macro

SUBNUM (menuNumber)

Scopo della macro

Questa macro estrae il numero d'ordine della sotto-voce di **menu** dall'argomento menuNumber. Si tratta di un valore compreso tra 0 e 31. SUBNUM consente al task di confrontare le informazioni ricevute da Intuition (messaggio IDCMP) riguardo alla selezione di un menu, con le definizioni delle varie voci al fine di identificare quale voce è stata selezionata dall'utente.

Argomenti della macro

menuNumber Valore binario a 16 bit che contiene, in formato compresso, i dati destinati a definire il numero di menu, il numero della voce e il numero della sotto-voce.

Discussione

Ci sono sei macro che riguardano i menu, le voci di menu e le sotto-voci di menu: MENUNUM, ITEMNUM, SUBNUM, SHIFTMENU, SHIFTITEM e SHIFTSUB. Si vedano anche le spiegazioni relative a queste macro.

Per determinare il numero della sotto-voce di menu si adopera la seguente istruzione di programma:

```
MySubItemNumber = SUBNUM (menuNumber);
```

MySubItemNumber sarà un valore valido (da 0 a 31) oppure, nel caso che non sia stata selezionata alcuna voce, lo speciale valore NOSUB. I valori di MENUNULL e NOSUB sono inclusi nella parte finale del file INCLUDE intuition.h, insieme alla precisa definizione della macro SUBNUM.

UnlockIBase

Sintassi di chiamata della funzione

UnlockIBase (lock_value)
AØ

Scopo della funzione

Questa funzione libera Intuition dal blocco imposto tramite la funzione LockIBase per congelare la struttura IntuitionBase. Si noti che l'argomento lock_value viene passato nel registro A0 (anziché nel registro D0, come avviene in genere per il primo argomento numerico di una funzione).

Argomenti della funzione

lock_value Numero intero senza segno da 32 bit che rappresenta la chiusura da sbloccare; si tratta del valore originariamente restituito dalla chiamata alla funzione LockIBase.

Discussione

Le funzioni che riguardano le chiusure nel software sistema di Intuition sono due: LockIBase e UnlockIBase. UnlockIBase è l'inverso di LockIBase. Queste due funzioni devono essere chiamate in coppia e in momenti ragionevolmente ravvicinati. Quando UnlockIBase restituisce il controllo, Intuition può tornare ad alterare i parametri nella struttura IntuitionBase, proseguendo le sue normali operazioni di gestione.

ViewAddress

Sintassi di chiamata della funzione

```
view = ViewAddress ()
DØ
```

Scopo della funzione

Questa funzione restituisce l'indirizzo della struttura View utilizzata da Intuition per la presentazione video. Se s'intende impiegare una funzione della libreria Graphics relativa a video, disegno, testo o animazione, ovvero qualsiasi funzione che inserisca elementi grafici in ambiente Intuition, si può ricorrere a ViewAddress per ottenere l'indirizzo dell'appropriata struttura View. Tramite questa struttura si può risalire alle strutture ViewPort che definiscono gli schermi di Intuition presenti nella view.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Si può progettare qualunque procedura in ambiente Intuition senza effettuare chiamate alla funzione ViewAddress. Se tuttavia si vuole usare una qualsiasi funzione della libreria Graphics per disegnare direttamente sulle view di Intuition, può essere necessario conoscere l'indirizzo della struttura View utilizzata da Intuition.

Un esempio dell'uso di questa funzione è contenuto nella spiegazione della funzione VievPortAddress.

ViewPortAddress

Sintassi di chiamata della funzione

```
viewPort = ViewPortAddress (window)
DØ                         AØ
```

Scopo della funzione

Questa funzione restituisce l'indirizzo della struttura ViewPort che definisce lo schermo al cui interno si trova la finestra specificata. Se si vuole impiegare una funzione della libreria Graphics relativa a video, disegno, testo o animazione, ovvero qualsiasi funzione che inserisca elementi grafici in ambiente Intuition, si può ricorrere alla funzione ViewPortAddress per procurarsi un puntatore all'appropriata struttura ViewPort.

Argomenti della funzione

window	Indirizzo della struttura Window.

Discussione

ViewPortAddress agisce in modo molto simile a ViewAddress. La discussione relativa alla funzione ViewAddress si applica egualmente bene alla funzione ViewPortAddress, con la differenza che quest'ultima restituisce l'indirizzo della struttura ViewPort relativa a un particolare schermo.

Per una certa struttura Window si possono ottenere gli indirizzi delle strutture RastPort, ViewPort o View utilizzando le seguenti istruzioni di programma, nelle quali si suppone che la variabile puntatore miaFinestra contenga l'indirizzo della struttura Window.

```
struct Window *miaFinestra;
struct RastPort *miaRastPort;
struct ViewPort *miaViewPort;
struct View *miaView;
...
miaRastPort = miaFinestra->RPort;
```

```
miaViewPort = ViewPortAddress (miaFinestra);
miaView = ViewAddress ();
```

WBenchToBack

Sintassi di chiamata della funzione

```
wasopen = WBenchToBack ()
DØ
```

Scopo della funzione

Questa funzione dispone lo schermo Workbench dietro a tutti gli altri schermi. L'operazione influenza soltanto il livello di profondità dello schermo Workbench rispetto agli altri schermi. Se lo schermo Workbench è aperto al momento della chiamata, la funzione restituisce il valore TRUE; altrimenti restituisce FALSE.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Nella libreria di Intuition ci sono quattro funzioni che agiscono direttamente con lo schermo Workbench: OpenWorkBench, CloseWorkBench, WBenchToBack e WBenchToFront.

Quando la funzione WBenchToBack fa arretrare all'ultimo livello di profondità lo schermo Workbench, molte (se non tutte) le finestre in esso aperte risulteranno nascoste o inaccessibili per l'utente. Quali finestre restano visibili dipende dalla grandezza e dalla disposizione degli altri schermi. Qualsiasi finestra dello schermo Workbench che sia rimasta visibile può comunque essere resa attiva dall'utente.

WBenchToFront

Sintassi di chiamata della funzione

```
wasopen = WBenchToFront ()
DØ
```

Scopo della funzione

Questa funzione porta lo schermo Workbench davanti a tutti gli altri schermi della presentazione video. L'operazione interessa soltanto il livello di profondità dello schermo: la funzione WBenchToFront non lo sposta dalla sua posizione. Se lo schermo Workbench è aperto al momento della chiamata, la funzione restituisce il valore TRUE; altrimenti restituisce FALSE.

Argomenti della funzione

Questa funzione non ha argomenti.

Discussione

Nella libreria Intuition ci sono quattro funzioni che agiscono direttamente con lo schermo Workbench: OpenWorkBench, CloseWorkBench, WBenchToBack e WBenchToFront.

Una schermata può essere composta da diversi schermi sovrapposti. Innanzitutto lo schermo Workbench, l'unico disponibile per fornire un'interfaccia utente comune alle finestre dei programmi; poi possono essere presenti anche diversi schermi personalizzati. La funzione WBenchToFront porta lo schermo Workbench davanti a tutti gli altri. Una volta che questo schermo è davanti, l'utente può accedere a tutte le finestre in esso contenute.

WindowLimits

Sintassi di chiamata della funzione

```
limitsOK = WindowLimits (window, min_width, min_height,
                         AØ      DØ         D1
                         max_width, max_height)
                         D2         D3
```

Scopo della funzione

Questa funzione ridefinisce le dimensioni minime e massime per una finestra. WindowLimits va chiamata se si vogliono cambiare i limiti di ridimensionamento di una finestra agendo sulla sua struttura Window. Se tutti gli argomenti della funzione sono validi, la funzione WindowLimits restituisce il valore booleano TRUE nella variabile limitsOK; se qualche argomento è al di fuori dell'estensione ammessa, la funzione restituisce FALSE.

Argomenti della funzione

window	Indirizzo della struttura Window.
min_width	La nuova larghezza minima per la finestra.
min_height	La nuova altezza minima per la finestra.
max_width	La nuova larghezza massima per la finestra.
max_height	La nuova altezza massima per la finestra.

Discussione

Ci sono dodici funzioni della libreria Intuition che operano direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

Quando si crea una finestra (adoperando la struttura NewWindow e chiamando la funzione OpenWindow), la sua misura iniziale è quella definita dai parametri LeftEdge, TopEdge, Width e Height, contenuti nella struttura NewWindow. Inoltre i parametri MinWidth, MinHeight, MaxWidth e MaxHeight impostano i limiti di ridimensionamento per la finestra. Quando OpenWindow restituisce il controllo, questi valori sono memorizzati nella struttura Window relativa alla finestra considerata.

La funzione WindowLimits consente di cambiare i valori dei parametri MinWidth, MinHeight, MaxWidth e MaxHeight della struttura Window. Si può poi adoperare la funzione SizeWindow per ridimensionare la finestra in uno degli schermi, in accordo con i nuovi limiti.

Se si vuole lasciare inalterato uno dei parametri di misura della finestra. bisogna impostare a 0 l'argomento corrispondente nella chiamata della funzione WindowLimits.

Se qualche argomento della funzione WindowLimits è al di fuori della gamma di valori consentiti (per esempio il minimo è più grande dell'attuale misura della finestra, oppure il massimo è più piccolo), quel particolare limite viene ignorato, mentre vengono accettati gli altri. Se l'utente sta effettuando un ridimensionamento della finestra, i nuovi limiti non hanno effetto fino a quando l'operazione non è terminata.

WindowToBack

Sintassi di chiamata della funzione

WindowToBack (window)
A0

Scopo della funzione

Questa funzione fa arretrare la finestra al livello di maggior profondità possibile, ovvero sposta la finestra dietro a tutte le altre dello stesso schermo. WindowToBack non restituisce alcun valore.

Argomenti della funzione

window	Indirizzo della struttura Window.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

In uno schermo possono esserci diverse finestre sovrapposte. Quando la funzione WindowToBack restituisce il controllo, la finestra indicata come argomento si troverà all'ultimo livello di profondità rispetto alle altre dello stesso schermo. Questa funzione non cambia lo stato di attività della finestra; se era attiva, lo rimane.

WindowToFront

Sintassi di chiamata della funzione

WindowToFront (window)
A0

Scopo della funzione

Questa funzione dispone una finestra davanti a tutte le altre finestre dello stesso schermo. WindowToBack non restituisce alcun valore.

Argomenti della funzione

window	Indirizzo della struttura Window.

Discussione

Ci sono dodici funzioni della libreria Intuition che agiscono direttamente con le finestre: BeginRefresh, CloseWindow, EndRefresh, EndRequest, MoveWindow, OpenWindow, ReportMouse, SetWindowTitles, SizeWindow, WindowLimits, WindowToFront e WindowToBack. Si vedano anche le spiegazioni relative a queste funzioni.

WindowToFront e WindowToBack servono per cambiare l'ordine di sovrapposizione delle finestre di Intuition. Si ricordi che tutte le operazioni di ordinamento hanno luogo soltanto sullo schermo in cui le finestre sono contenute. Ogni finestra è costituita da un layer.

La finestra non viene spostata in primo piano subito dopo l'esecuzione della funzione WindowToFront, ma soltanto quando Intuition riceve un evento di input (cosa che accade con una frequenza che va da un minimo di dieci a un massimo di cinquanta volte al secondo).

Le funzioni della libreria Icon

Introduzione

Questo capitolo tratta le funzioni della libreria Icon, quelle che consentono di progettare programmi dotati di icone e capaci di aprire finestre sullo schermo Workbench dell'Amiga. Le funzioni della libreria Icon rientrano in sette categorie:

- le generiche funzioni per la gestione della memoria: AddFreeList e FreeFreeList, per aggiungere blocchi di memoria di dimensioni prefissate a liste di memoria libera, e per liberare tutti i blocchi di memoria in una particolare lista di memoria libera.

- La funzione per liberare la memoria precedentemente assegnata dal sistema a un oggetto-disco quando il suo file .info è stato caricato in memoria dal disco: FreeDiskObject.

- Le funzioni per leggere e scrivere su disco i file .info degli oggetti-disco: GetDiskObject e PutDiskObject.

- Le funzioni per la gestione dei nomi dei programmi applicativi (detti anche tool): FindToolType e MatchToolValue, rispettivamente per rintracciare la stringa che indica il tipo di tool, e per verificare la corrispondenza con una particolare sotto-stringa che indica il tipo di tool.

- La funzione per assegnare un nuovo nome a un file ottenuto come copia diretta di un altro: BumpRevision.

Il termine Workbench si riferisce tanto al programma *Workbench* quanto all'omonimo schermo di default di Intuition. Il programma *Workbench* risiede per lo più in ROM, ma non viene attivato per default dal sistema. Perché diventi operativo dev'essere mandato in esecuzione il comando LOADWB del CLI, cosa che normalmente accade durante la procedura di startup se il comando appare all'interno del file s:startup-sequence.

Il programma *Workbench* contiene un insieme di routine che interagiscono con i programmi applicativi al fine di fornire un'interfaccia utente comune. Inoltre, fornisce una procedura alternativa per avviare e mandare in esecuzione i programmi, permettendo loro di presentare icone e aprire finestre sullo schermo Workbench. L'utente può poi selezionare le icone per mandare in esecuzione il programma o svolgere altre operazioni.

Il programma *Workbench* è sostanzialmente un sistema user-friendly che consente agli utenti d'interagire con il sistema dei dischi utilizzando icone per i dischi, per le directory dei dischi, per i programmi applicativi e per i file da essi generati. Le funzioni della libreria Icon, che interagiscono con il programma *Workbench*, offrono facilitazioni per la manipolazione delle icone e

delle finestre dei programmi. Tra l'altro, il *Workbench* è predisposto per l'esecuzione automatica dei programmi (attraverso il meccanismo del tool di default) e per l'apertura automatica delle relative finestre. Ciò consente di dirigere le operazioni di input/output attraverso determinate finestre anche se il programma non viene mandato in esecuzione da CLI. Tutte queste caratteristiche dell'interfaccia utente possono essere sfruttate per semplificare il lavoro del programmatore.

Il *Workbench* opera con un insieme di file .info relativi a oggetti disco. Tali file rappresentano le icone associate ai corrispondenti programmi, file dati o directory. Ognuno di essi contiene una definizione dell'immagine destinata all'icona e altre informazioni necessarie affinché l'oggetto disco possa interagire con il *Workbench*. La struttura DiskObject fa parte del file .info dell'oggetto disco, e contiene informazioni che riguardano il programma rappresentato dal file. In particolare, ogni struttura DiskObject contiene una struttura Gadget per definire l'icona del programma e un puntatore a una struttura dati che definisce la finestra cassetto nella quale l'icona è destinata ad apparire.

Un cassetto è una finestra che mostra l'insieme di icone relative ai file che si trovano in una directory sul disco. La struttura DiskObject viene illustrata nel corso della descrizione della funzione FreeDiskObject.

I file oggetto e i file disco del Workbench vengono chiamati file .info per due ragioni: (1) contengono tutte le informazioni necessarie per presentare un'icona associata ai vari tipi di oggetti che abbiamo citato, e per mettere in relazione tale icona con le finestre e il software dei relativi programmi; (2) appaiono nelle directory del disco con i caratteri .info aggiunti in coda ai loro nomi di file.

Il *Workbench* consente all'utente di eseguire nello schermo Workbench le seguenti operazioni:

- selezionare un'icona per azioni da intraprendere successivamente, come la copia del corrispondente file, l'alterazione del nome, o la richiesta di ulteriori informazioni.
- Avviare un'applicazione.
- Aprire una finestra cassetto, cioè una directory del disco.
- Creare una finestra per un programma applicativo.
- Aprire un file dati di un programma applicativo.

Un'icona viene selezionata per operazioni successive semplicemente premendo il pulsante sinistro del mouse quando il puntatore vi si trova sopra; tenendolo premuto è possibile spostare attraverso lo schermo Workbench la maggior parte delle icone.

Un programma viene avviato attraverso la doppia pressione del pulsante sinistro del mouse quando il puntatore si trova sopra la corrispondente icona o sull'icona di un file di dati generato dal programma. Il *Workbench* rileva i

messaggi del mouse e invia un messaggio per avviare l'esecuzione del programma, aprire una finestra destinata all'I/O e così via. Il sistema *Workbench* prevede l'attivazione automatica dei programmi di default attraverso un insieme di parametri presenti nella struttura DiskObject.

Se l'utente seleziona un'icona di tipo project, il tool di default è il programma applicativo che ha originariamente prodotto quel file. Per esempio, se l'utente seleziona l'icona di un file prodotto da un programma di elaborazione testi, quel programma (che costituisce il tool) viene automaticamente mandato in esecuzione. Ovviamente, perché questo procedimento funzioni occorre che l'applicazione che ha generato il file dati selezionato dall'utente si trovi nella directory indicata per default nel file .info.

I cassetti si aprono selezionando due volte le loro icone, come per mandare in esecuzione un file. Un cassetto è in pratica una directory del disco, e come tale può contenere altri file e altre directory; quando il cassetto viene aperto risultano visibili soltanto le directory e i file dotati di file .info e quindi di icone. Un cassetto è definito da una struttura DrawerData.

Ciascun programma fornisce diversi modi per aprire finestre e file di dati. In genere le finestre diventano attive quando l'utente preme il pulsante sinistro del mouse mentre il puntatore video si trova al loro interno. I file dati dei programmi vengono generalmente aperti selezionando la relativa icona e la voce Open del menu file, oppure selezionandoli una seconda volta.

Le icone usate dal Workbench

Il *Workbench* riconosce i seguenti tipi di icone:

- WBDISK. Questa icona rappresenta la directory radice del disco Workbench. È colorata in bianco, arancio e nero e appare nell'angolo superiore destro. Sotto all'icona compare in bianco la scritta "Workbench", sullo sfondo blu dello schermo Workbench.

- WBDRAWER. Questa icona rappresenta la directory di un disco. Selezionandola due volte compare una finestra cassetto che contiene un'icona per ogni programma di sistema, programma applicativo e utility presenti in quella directory e dotati di file .info.

- WBTOOL. Questa icona rappresenta un programma eseguibile, cioè un tool. Un esempio è il programma *Preferences*: se la sua icona viene selezionata con la doppia pressione, il programma *Preferences* viene mandato in esecuzione.

- WBPROJECT. Questa icona rappresenta un file dati di un programma applicativo. Selezionandola due volte, si fa in modo che il *Workbench* mandi in esecuzione il relativo programma, quello che ha creato il file. Inoltre, il *Workbench* indica al programma che deve avviare l'elaborazione di quel particolare file dati.

- WBGARBAGE. Questa icona rappresenta il cassetto Trashcan; si tratta della directory nella quale l'utente deposita tutti i file di cui si vuole liberare. Se si seleziona con la doppia pressione l'icona WBGARBAGE, si apre una finestra cassetto che contiene l'insieme delle icone WBDRAWER, WBTOOL e WBPROJECT di cui l'utente si è sbarazzato.

- WBKICK. Questa icona rappresenta un disco non DOS, come il disco *Kickstart* per l'Amiga 1000, sul quale non sono incise tracce di sistema.

La struttura Gadget di un'icona

Il *Workbench* agisce con la struttura Gadget di Intuition per definire le immagini grafiche da impiegare per le icone. Il programma Icon Editor serve per cambiare le immagini delle icone. Vi si può accedere aprendo l'icona del disco Extras e selezionando successivamente con la doppia pressione l'icona Tools. Nella nuova finestra che si apre, si avvia quindi l'esecuzione del programma Icon Editor selezionandone l'icona con la doppia pressione.

Ogni icona con la quale il *Workbench* agisce è associata a una struttura Gadget di Intuition. Quando l'utente seleziona l'icona, viene inviato al *Workbench* un messaggio.

La struttura Gadget usata dal *Workbench* rappresenta sempre un gadget TRUE/FALSE. Non tutti gli elementi della struttura Gadget vengono impiegati dal *Workbench*; quelli inutilizzati vanno impostati a zero. Questa è la lista dei parametri della struttura Gadget che vengono impiegati:

- i parametri Width e Height contengono la larghezza e l'altezza in pixel della regione attiva dell'icona. Quando l'utente preme il pulsante sinistro del mouse in quella regione, il *Workbench* interpreta l'evento come una selezione.

- Il parametro Flags contiene alcuni flag che definiscono l'immagine del gadget. Attualmente il flag dell'immagine di gadget dev'essere impostato a GADGIMAGE. Ciò assicura che il gadget venga presentato con un'immagine piena anziché con un contorno. Inoltre il *Workbench* prevede tre modi di evidenziazione che definiscono l'aspetto che assumono le icone quando vengono selezionate: GADGHIMAGE, GADGBACKFILL e GADGHCOMP. Il modo GADGHIMAGE utilizza un'immagine alternativa. Il modo GADGHCOMP effettua il complemento dei pixel. GADGBACKFILL agisce come GADGHCOMP, ma in più assicura che non si produca un contorno color arancio intorno all'immagine. Tutti gli altri bit di questo parametro devono essere impostati a zero.

- Del parametro Activation devono essere impostati soltanto i flag RELVERIFY e GADGIMMEDIATE. Queste impostazioni consentono al

Workbench di rilevare la selezione del gadget, i movimenti del mouse mentre il gadget è selezionato e il rilascio del pulsante del mouse quando il puntatore si trova sul gadget.

- Il parametro Type contiene il tipo di gadget; per indicare un semplice gadget TRUE/FALSE questo parametro va impostato a BOOLGADGET.

- Il parametro SelectRender dev'essere impostato soltanto se il modo di evidenziazione dell'icona è GADGHIMAGE.

Le strutture della libreria Icon

DiskObject è la principale struttura adoperata dal *Workbench* per operare con le icone dei programmi applicativi sullo schermo Workbench. Tali icone consentono al *Workbench* di avviare l'esecuzione dei programmi quando l'utente le seleziona, o seleziona i relativi file project. La struttura DiskObject fa parte del file .info di oggetto-disco; viene illustrata nel corso della spiegazione della funzione FreeDiskObject.

La struttura DrawerData gestisce la presentazione di una finestra cassetto per un file .info di oggetto-disco. Contiene una sotto-struttura NewWindow e specifica la posizione della finestra cassetto sullo schermo Workbench. La struttura DrawerData viene illustrata nel corso di questa stessa introduzione.

La struttura FreeList gestisce la memoria libera disponibile per gli oggetti-disco, quando vengono portati sullo schermo Workbench. Ogni struttura FreeList contiene la quantità di memoria libera e una sotto-struttura MemList dell'Exec. La struttura FreeList viene illustrata nel corso della spiegazione della funzione AddFreeList.

La struttura WBStartup contiene tutte le informazioni necessarie per avviare un programma tramite il *Workbench*: prima di tutto una sotto-struttura Message e una sotto-struttura MsgPort, che consentono al *Workbench* di scambiare messaggi con i programmi. Vi sono inoltre due puntatori: il primo a una finestra del programma e il secondo a una struttura WBArg, che contiene la lista argomenti passata all'applicazione dal *Workbench*. Le strutture WBStartup e WBArg vengono illustrate nel corso di questa stessa presentazione.

Tutte queste strutture sono definite nei file INCLUDE workbench.h, startup.h e icon.h (per la programmazione in C) e workbench.i, startup.i e icon.i (per la programmazione in Assembly).

Le operazioni in ambiente Workbench

Il sistema *Workbench* segue sempre una generica sequenza operativa, benché in realtà le sue operazioni si basino sulle azioni dell'utente e

sull'organizzazione dei vari programmi.

Quando l'Amiga viene avviato, nell'angolo superiore destro dello schermo appare l'icona del disco Workbench. Quando l'utente preme due volte il pulsante del mouse per selezionare l'icona, compaiono tutte le icone presenti nella directory radice del disco: programmi, file dati e directory.

L'utente può avviare qualsiasi programma applicativo si trovi sul disco Workbench con una nuova doppia pressione sull'icona relativa. Quando lo fa, il gadget relativo all'immagine dell'icona invia un messaggio TRUE (che corrisponde all'avvenuta selezione) al *Workbench*; quest'ultimo verifica la struttura DiskObject associata all'icona in questione e individua il programma applicativo (tool) che rappresenta. Il *Workbench* allora carica il programma e lo manda in esecuzione, inviandogli anche un messaggio. Il programma deve restituire quel messaggio al *Workbench* per indicare che l'operazione è stata svolta.

Se l'icona selezionata rappresenta un file project, il *Workbench* controlla la struttura DiskObject per sapere a quale programma appartiene; invia quindi un messaggio al programma per avviarne l'esecuzione. Ancora una volta, il programma restituisce al *Workbench* un messaggio di risposta al termine della sua esecuzione.

La struttura WBStartup

La sequenza di operazioni appena descritta viene gestita attraverso la struttura WBStartup, che ha la seguente forma:

```
struct WBStartup {
        struct Message sm_Message;
        struct MsgPort *sm_Process;
        BPTR sm_Segment;
        LONG sm_NumArgs;
        char *sm_ToolWindow;
        struct WBArg *sm_ArgList;
};
```

Ecco il significato di ciascun parametro della struttura WBStartup.

- Il parametro sm_Message contiene il nome della sotto-struttura Message standard dell'Exec assegnata all'applicazione. Tale struttura gestisce i messaggi scambiati dall'applicazione con il *Workbench* e viceversa. Il *Workbench* possiede una message port per ricevere i messaggi che provengono dalle applicazioni.

- Il parametro sm_Process è un puntatore alla struttura Process standard dell'Exec per il task del programma. La relativa message port serve come reply port per i messaggi provenienti dal *Workbench*.

- Il parametro sm_Segment è un puntatore a una lista di segmenti, restituita dalla funzione LoadSeg del DOS, che identifica la posizione in memoria del programma e dei suoi dati.

- Il parametro sm_NumArgs contiene il numero di argomenti passati al programma applicativo dal *Workbench*.

- Il parametro sm_ToolWindow è un puntatore alla descrizione della finestra da aprire per il programma. Questa descrizione coincide con la stringa presente nel parametro do_ToolWindow nella struttura DiskObject. Questo puntatore permette al programma di aprire subito una finestra quando avvia l'esecuzione. Se vale zero, non viene aperta nessuna finestra di default.

- Il parametro sm_ArgList è un puntatore a una lista sm_NumArgs strutture WBArg che gestiscono gli argomenti passati dal *Workbench* al programma applicativo.

La struttura WBArg

La struttura WBArg ha la forma seguente:

```
struct WBArg {
    BPTR wa_Lock;
    BYTE *wa_Name;
};
```

Il parametro wa_Lock è un puntatore che individua il lock relativo al parametro in questione, se questo è di un tipo (es. directory, file) che supporta i lock.

Il parametro wa_Name è un puntatore al nome dell'argomento stesso.

La struttura DrawerData

Ogni volta che viene aperta una nuova finestra cassetto sullo schermo video dell'Amiga, il *Workbench* impiega una struttura DrawerData per definire il contenuto del cassetto. Le finestre cassetto sono una rappresentazione visiva di una parte di una directory del disco. La struttura DrawerData ha la forma seguente:

```
struct DrawerData {
    struct NewWindow dd_NewWindow;
    LONG dd_CurrentX;
    LONG dd_CurrentY;
};
```

I parametri hanno il seguente significato:

- il parametro dd_NewWindow contiene una sotto-struttura NewWindow di Intuition usata per definire la finestra cassetto.
- I parametri dd_CurrentX e dd_CurrentY contengono le coordinate x e y per l'origine della finestra cassetto sullo schermo Workbench.

AddFreeList

Sintassi di chiamata della funzione

```
status = AddFreeList (freeList, memBlock, length)
DØ                   AØ       A1        A2
```

Scopo della funzione

Questa funzione aggiunge un blocco di memoria a una lista di memoria libera. Se la chiamata alla funzione non ha successo, viene restituito il valore zero.

Argomenti della funzione

freeList	Indirizzo di una struttura FreeList.
memBlock	Indirizzo del primo byte del blocco di memoria che dev'essere aggiunto alla lista della memoria libera.
length	Lunghezza in byte del blocco di memoria da aggiungere alla lista della memoria libera.

Discussione

Nella libreria Icon ci sono due funzioni che gestiscono la memoria: AddFreeList e FreeFreeList. La prima aggiunge un blocco di memoria di grandezza predeterminata a una lista di memoria libera, mentre la seconda libera da una lista di memoria libera tutti i blocchi indicati.

La libreria Icon può mantenere diverse liste di memoria libera. Per

ciascuna di esse serve una specifica struttura FreeList. Ogni volta che un programma non deve più usare un blocco di memoria, si può aggiungerlo a una di queste liste di memoria libera, usando la funzione AddFreeList. Quando la funzione AddFreeList restituisce il controllo, la lista risulta estesa. Se non è disponibile abbastanza memoria per portare a termine la chiamata, viene restituito il valore NULL. Si noti che la funzione AddFreeList non alloca la memoria richiesta, ma aggiunge a una determinata lista di memoria libera un blocco di memoria allocato (che quindi dovrà essere liberato).

La struttura FreeList

La funzione AddFreeList opera con la struttura FreeList, che ha la forma seguente:

```
struct FreeList {
    WORD fl_NumFree;
    struct List fl_MemList;
};
```

Il parametro fl_NumFree contiene il numero di blocchi di memoria libera da otto byte disponibili per essere allocati. Il parametro fl_MemList è la sotto-struttura List che tiene sotto controllo la memoria nel sistema *Workbench* per questa particolare struttura FreeList.

BumpRevision

Sintassi di chiamata della funzione

```
result = BumpRevision (newBuf, oldName)
D0                    A0      A1
```

Scopo della funzione

Questa funzione, aggiunge la stringa "copy of" al nome di un file dotato di icona. Quando l'utente produce copie da altre copie, BumpRevision genera i relativi nomi con lo stesso criterio, ma riportando il numero di copia aggiornato nel nome del file. La funzione tronca inoltre il nuovo nome alla misura massima consentita dall'AmigaDOS, che è di 30 caratteri. Il buffer in cui viene riversato il nuovo nome dev'essere lungo almeno 31 caratteri, dato che un carattere viene azzerato per indicare la chiusura del buffer. BumpRevision restituisce un puntatore al nuovo nome.

Argomenti della funzione

newBuf — Indirizzo del buffer destinato a ricevere la stringa caratteri del nuovo nome.

oldName — Indirizzo della stringa caratteri contenente il nome originale.

Discussione

BumpRevision genera un nuovo nome per un file che è la copia di un altro. Si può usare la funzione BumpRevision per creare una serie di nomi diversi per file che sono la copia diretta di un solo file originale. Questa funzione si limita a creare il nuovo nome per il file di copia; non produce effettivamente la nuova copia.

BumpRevision viene chiamata indirettamente ogni volta che un utente effettua la copia di un file in ambiente *Workbench* e dispone la relativa icona sullo schermo. BumpRevision genera il nome che appare al di sotto della nuova icona. Per esempio, se si copia un file denominato Testo, apparirà un'icona al di sotto della quale verrà mostrato il nome: "copy of Testo". Se si effettua da questa una seconda copia, il suo nome sarà: "copy 2 of Testo".

FindToolType

Sintassi di chiamata della funzione

```
stringPointer = FindToolType (toolTypes, typeName)
D0                            A0         A1
```

Scopo della funzione

Questa funzione cerca un elemento typeName in un array ToolTypes. Restituisce l'indirizzo del primo carattere dopo il segno di uguale, quello che appare di seguito alla stringa typeName. Se nell'array ToolTypes non viene ritrovato nessun valore che corrisponda all'argomento typeName, viene restituito il valore NULL.

Argomenti della funzione

toolTypes Indirizzo dell'array ToolTypes in RAM.

typeName Indirizzo della stringa di caratteri che dev'essere cercata nell'array ToolTypes.

Discussione

Nella libreria Icon ci sono due funzioni che riguardano le stringhe dell'array ToolTypes: FindToolType e MatchToolValue. FindToolType consente di ottenere l'indirizzo di un elemento stringa nell'array ToolTypes; MatchToolValue permette di cercare una determinata sotto-stringa in un elemento stringa dell'array ToolTypes.

Queste due funzioni servono a impostare dei parametri operativi per il tool o il project corrispondente tramite i valori inseriti dall'utente nell'icona stessa per mezzo della funzione Info presente nei menu del programma *Workbench*.

Per usare FindToolType, si deve prima definire in RAM un array ToolTypes, ovvero un insieme di elementi costituiti da stringhe di caratteri. Ogni elemento contiene informazioni su uno specifico file .info.

Ciascuna stringa consiste in due sotto-stringhe principali, separate da un segno di uguale. A sinistra del segno di uguale si trova il nome del tipo di tool; a destra c'è una lista di valori stringa associati al nome. Ognuna di queste sotto-stringhe è separata dalla seguente da una barra verticale (il segno |).

Il formato generale per ciascun elemento stringa nell'array ToolTypes è:

typeName = sotto-stringa1 [|sotto-stringa2][|sotto-stringa3]...
[|sotto-stringaN]

dove typeName è il nome del tipo di tool, mentre le sotto-stringhe associate a questo nome iniziano dalla sotto-stringa1 e arrivano alla sotto-stringaN. In genere le sotto-stringhe sono nomi di programmi o nomi di categorie di file. Per esempio, un elemento di array ToolTypes per il codice sorgente di un programma Basic potrebbe essere:

FILETYPE=ABASIC.program|text

Qualsiasi programma applicativo che lavori utilizzando file di testo (come un programma di elaborazione testi oppure un programma Basic) sarà in grado di usare un file .info di oggetto-disco che contenga un simile elemento nel suo array ToolTypes. Ciò significa che quando questa icona viene selezionata dall'utente, può essere mandato in esecuzione tanto un programma di elaborazione testi quanto un compilatore o un interprete ABASIC.

Se nella chiamata alla funzione FindToolType l'argomento typeName viene specificato con il valore letterale FILETYPE, viene restituito un puntatore al carattere A della stringa:

ABASIC.program | text

Si tratta di una stringa composta da due sotto-stringhe; ABASIC.program e text. Per confrontarle entrambe con la stringa il cui puntatore è stato restituito dalla funzione FindToolType, si può adoperare la funzione MatchToolValue. Se la funzione MatchToolValue indica che esiste una corrispondenza, il programma riconoscerà che il file può essere impiegato con uno specifico programma applicativo. È importante ricordare che la funzione FindToolType restituisce l'indirizzo della stringa tool e non la stringa in sé. Quando MatchToolValue effettua il confronto, il puntatore viene impiegato internamente per cercare un valore valido.

Una stringa dell'array ToolTypes può essere lunga fino a 32K. È tuttavia più opportuno che ciascuna stringa risulti inferiore a una riga (ovvero circa 60 caratteri). I caratteri per le stringhe si scelgono dal codice dei caratteri a otto bit ANSI (American National Standards Institute), che riproduce il codice a sette bit ASCII (American Standard Code for Information Exchange), con l'ottavo bit impiegato per l'estensione ai caratteri delle varie grafie nazionali. Questi caratteri appaiono, nella normale impostazione di tastiera, tenendo premuto il tasto Alt. Si possono utilizzare per esempio con il file Notepad. Nelle stringhe-caratteri l'uso delle maiuscole o delle minuscole è significativo; sono inoltre ammessi soltanto caratteri stampabili.

Una volta che gli elementi dell'array ToolTypes sono stati definiti, l'array in questione va collocato in RAM e il suo indirizzo va memorizzato nella struttura DiskObject, più precisamente nel puntatore do_ToolTypes. Quando si usa la funzione PutDiskObject per trasferire l'oggetto-disco in un file .info di oggetto-disco, anche l'array ToolTypes verrà scritto su disco. In seguito, quando si chiama la funzione GetDiskObject per riportare in RAM il file .info di oggetto-disco, anche l'array ToolTypes viene riportato in RAM.

FreeDiskObject

Sintassi di chiamata della funzione

FreeDiskObject (diskObject)
A0

Scopo della funzione

Questa funzione libera tutta la memoria associata a un file .info di oggetto-disco; libera inoltre la memoria assegnata alla struttura DiskObject. La memoria per il file .info di oggetto-disco era stata allocata dal sistema al momento del caricamento in RAM tramite la funzione GetDiskObject.

Argomenti della funzione

diskObject Indirizzo della struttura DiskObject.

Discussione

FreeDiskObject è la sola funzione della libreria Icon che gestisce direttamente la memoria per i file .info di oggetto-disco. Libera la memoria precedentemente assegnata a un file .info di oggetto-disco tramite la funzione GetDiskObject.

La funzione FreeDiskObject svolge il suo lavoro chiamando la funzione FreeFreeList. FreeDiskObject può essere usata soltanto per le strutture DiskObject assegnate attraverso la funzione GetDiskObject, la quale si prende cura d'inizializzare la struttura FreeList dell'oggetto.

La struttura DiskObject

La funzione FreeDiskObject agisce con la struttura DiskObject. La forma della struttura è la seguente:

```
struct DiskObject {
      UWORD do_Magic;
      UWORD do_Version;
      struct Gadget do_Gadget;
      UBYTE do_Type;
      char *do_DefaultTool;
      char **do_ToolTypes;
      LONG do_CurrentX;
      LONG do_CurrentY;
      struct DrawerData *do_DrawerData;
      char *do_ToolWindow;
      LONG do_StackSize;
 };
```

Ecco il significato dei parametri della struttura DiskObject:

- il parametro do_Magic contiene la costante WB_DISKMAGIC. La funzione PutDiskObject inserisce questo valore nella struttura DiskObject nel momento in cui la struttura viene salvata sul disco. Il parametro identifica il file in questione come file .info. La funzione GetDiskObject non carica alcun file, a meno che nella struttura DiskObject associata al file non sia presente questo valore.

- Il parametro do_Version contiene il numero di versione del file icona. Al momento il suo valore è la costante WB_DISKVERSION. La funzione PutDiskObject imposta questo valore quando salva su disco la struttura DiskObject.

- Il parametro do_Gadget è la struttura Gadget di Intuition usata per mantenere le informazioni d'immagine per l'icona dell'oggetto-disco.

- Il parametro do_Type contiene il tipo di icona. Può essere WBDRAWER. WBDISK, WBTOOL, WBPROJECT, WBGARBAGE oppure WBKICK.

- Il parametro do_DefaultTool punta alla stringa del nome del programma corrispondente al tool di default, quello destinato a iniziare automaticamente l'esecuzione quando l'utente seleziona l'icona associata alla struttura DiskObject. Un tool di default è un programma applicativo che viene avviato quando l'utente seleziona la corrispondente icona project. tramite la doppia pressione.

- Il parametro do_ToolTypes punta a un array di stringhe a formato libero che definiscono eventuali parametri per i programmi applicativi in grado di lavorare con il file .info rappresentato dall'icona. Si vedano, a questo proposito, le spiegazioni delle funzioni FindToolType e MatchToolValue.

- I parametri do_CurrentX e do_CurrentY contengono le coordinate x e y per l'icona all'interno della finestra che la contiene. Ciascun cassetto costituisce la rappresentazione visiva di una directory o di una sotto-directory dei file presenti sul disco. Ogni icona nella finestra cassetto ha una posizione nel sistema di coordinate della finestra. L'utente può effettuare lo scroll della finestra usando gli appositi gadget. Per ognuno di questi parametri che abbia il valore NO_ICON_POSITION, è il *Workbench* che si preoccupa di calcolare un valore ragionevole per la posizione dell'icona corrispondente. Se nella finestra non esiste spazio, l'icona viene collocata a destra della regione visibile.

- Il parametro do_DrawerData punta a una struttura DrawerData. Le icone di tipo WBDISK, WBDRAWER e WBGARBAGE possono essere aperte come finestre cassetto. Parte della struttura DrawerData è una struttura NewWindow. Il *Workbench* impiega questa struttura NewWindow per mantenere le necessarie informazioni sulla grandezza e sulla posizione della finestra, allo scopo di riaprire la finestra nella stessa posizione in cui era stata chiusa. Le posizioni CurrentX e CurrentY dell'origine della finestra vengono conservate anche nella struttura NewWindow.

- Il parametro do_ToolWindow punta a un file che contiene la definizione di una finestra. Viene utilizzato soltanto se la struttura DiskObject considerata rappresenta un programma di applicazione (se cioè

do_Type = WBTOOL). Se questo parametro è impostato, la finestra diventa la finestra standard per l'input e l'output relativo al programma applicativo. Per default, il *Workbench* avvia un programma senza assegnargli nessuna finestra di I/O.

- Il parametro do_StackSize contiene la misura dello stack che dev'essere impiegato dal tool. Questo parametro viene utilizzato soltanto se la struttura DiskObject rappresenta un programma di applicazione (se cioè do_Type = WBTOOL). Se la misura è uguale a NULL, il *Workbench* imposta per default una ragionevola misura di stack (in genere 4K).

FreeFreeList

Sintassi di chiamata della funzione

FreeFreeList (freeList)
AØ

Scopo della funzione

Questa funzione libera tutti i blocchi di memoria indicati nella lista individuata dalla struttura FreeList, e anche la memoria assegnata alla struttura stessa.

Argomenti della funzione

freeList	Indirizzo della struttura FreeList.

Discussione

Nella libreria Icon ci sono due funzioni che gestiscono la memoria: AddFreeList e FreeFreeList. La prima aggiunge un blocco di memoria di grandezza predeterminata a una lista di memoria libera, mentre la seconda libera da una lista di memoria libera tutti i blocchi indicati.

La libreria Icon può mantenere diverse liste di memoria libera. Per ciascuna di esse serve una specifica struttura FreeList. Per la definizione della struttura FreeList si veda la spiegazione della funzione AddFreeList.

GetDiskObject

Sintassi di chiamata della funzione

```
diskObject = GetDiskObject (fileName)
DØ                          AØ
```

Scopo della funzione

Questa funzione legge dal disco un file .info di oggetto-disco e lo trasferisce in memoria. L'inizio del file .info è una struttura DiskObject. Se GetDiskObject ha successo, restituisce l'indirizzo in RAM della struttura DiskObject. Se invece la chiamata non ha successo, viene restituito il valore zero. La ragione del fallimento si può individuare impiegando la funzione di sistema IoErr.

Argomenti della funzione

fileName	Indirizzo che individua in memoria la stringa nome del file .info di oggetto-disco da caricare.

Discussione

Nella libreria Icon ci sono due funzioni che riguardano le operazioni di trasferimento dei file .info di oggetto-disco: GetDiskObject e PutDiskObject. La funzione GetDiskObject legge dal disco un file .info di oggetto-disco e lo trasferisce in memoria; la funzione PutDiskObject salva sul disco un file .info, prelevandolo dalla memoria.

Quando GetDiskObject restituisce il controllo, il sistema alloca una struttura DiskObject; viene anche allocata una struttura FreeList associata alla struttura DiskObject (che però non ne fa parte). In seguito, si può usare la funzione FreeDiskObject per rilasciare la memoria precedentemente assegnata con la funzione GetDiskObject.

Le strutture DiskObject e FreeList sono illustrate rispettivamente nel corso delle discussioni relative alle funzioni FreeDiskObject e AddFreeList. Si ricordi che il file .info ha lo stesso nome del file corrispondente con l'automatica aggiunta dei caratteri .info; sul disco viene quindi cercato un file con questa estensione.

Per esempio, se il nome del file .info di oggetto-disco era MioTesto (che immaginiamo sia un file testo di tipo project, prodotto da un word processor, cioè con do_Type = WBPROJECT), la funzione GetDiskObject cercherà nella directory del disco un file denominato MioTesto.info. Tale file conterrà la struttura DiskObject e altre informazioni che riguardano l'oggetto-disco.

MatchToolValue

Sintassi di chiamata della funzione

```
stringfound = MatchToolValue (stringPointer, subString)
D0                           A0            A1
```

Scopo della funzione

Questa funzione cerca una specifica sotto-stringa nella stringa specificata. Se la sotto-stringa viene trovata, la funzione restituisce il valore 1, altrimenti il valore 0.

Argomenti della funzione

stringPointer	Indirizzo della stringa, restituito dalla funzione FindToolType.
subString	Indirizzo della sotto-stringa cercata.

Discussione

Nella libreria Icon ci sono due funzioni che operano con gli array di stringhe ToolTypes: FindToolType e MatchToolValue. FindToolType consente di determinare un puntatore a uno specifico elemento stringa nell'array ToolTypes; MatchToolValue permette di cercare una sotto-stringa specifica in uno di questi elementi.

Per confrontare una specifica sotto-stringa con una o più stringhe i cui puntatori sono stati restituiti dalla funzione FindToolType, si adopera la funzione MatchToolValue.

MatchToolValue sa come deve analizzare la sintassi della stringa

appartenente all'array ToolTypes per cercare una specifica sotto-stringa. In particolare, riconosce il carattere barra verticale (|) come separatore di sotto-stringhe tra gli elementi della stringa.

PutDiskObject

Sintassi di chiamata della funzione

```
status = PutDiskObject (fileName, diskObject)
D0                     A0         A1
```

Scopo della funzione

Questa funzione preleva dalla memoria un file .info di oggetto-disco che contiene una struttura DiskObject e lo salva su disco. Se il valore restituito dalla funzione è diverso da zero, significa che la chiamata alla funzione ha avuto successo, altrimenti che non ha avuto successo. La ragione del fallimento può essere individuata ricorrendo alla funzione di sistema IoErr.

Argomenti della funzione

fileName — Indirizzo del nome da assegnare al file .info di oggetto-disco. L'estensione ".info" viene aggiunta dalla funzione stessa.

diskObject — Indirizzo della struttura DiskObject da includere nel file .info al momento della scrittura su disco.

Discussione

Nella libreria Icon ci sono due funzioni che riguardano le operazioni di trasferimento dei file .info di oggetti-disco: GetDiskObject e PutDiskObject. La funzione GetDiskObject legge dal disco un file .info e lo trasferisce in memoria; la funzione PutDiskObject preleva dalla memoria un file .info e lo salva sul disco. Il parametro fileName della chiamata alla funzione PutDiskObject viene utilizzato per il nome del file su disco.

Glossario

Allocare

Allocare memoria significa appropriarsi di un'area di memoria da un pool di memoria libera, al fine di poterla utilizzare per memorizzarvi dati. Per svolgere questa operazione si possono utilizzare le funzioni AllocMem, AllocEntry e Allocate della libreria Exec o AllocRemember della libreria Intuition. In ogni caso, la maggior parte delle librerie possiede proprie funzioni di allocazione e disallocazione della memoria. Si noti che nelle allocazioni è possibile specificare gli attributi che l'area di memoria allocata dovrà possedere.

Allocare un bit di segnale

Ogni task dispone di un pool di 32 bit di segnale, di cui 16 sono riservati al sistema. Ogni elemento del sistema, compreso lo stesso task interessato, può attivare uno di questi segnali con l'intento di avvisare un task di un evento. Quando il segnale viene attivato, il task riceve il controllo, ma soltanto se si era messo in attesa; in caso contrario non può rilevarne l'arrivo. Comunque, se il task si mette in attesa di un segnale che è già stato attivato, riottiene subito il controllo. La funzione necessaria per entrare in attesa dell'attivazione di uno o più segnali è Wait. Un task può allocare uno dei suoi segnali (o bit di segnale) per ricordare a se stesso che quel segnale è già in uso, e può poi assegnarlo a una message port o a un altro task. L'allocazione non è una fase obbligatoria nell'uso dei segnali, ma è molto comoda per non dover aggiungere nei codici del proprio task istruzioni che tengano conto dei segnali in uso. Si noti che un segnale si dice attivo quando il suo bit, contenuto nel parametro tc_SigRecvd della struttura Task, è a 1. Quando si dice che una message port o un task inviano un segnale a un altro task, si vuole intendere che quel segnale viene attivato. Si veda la voce *Assegnare un segnale*.

Alta risoluzione

Questo modo video prevede 640 pixel di risoluzione orizzontale sullo schermo video; non influenza la risoluzione verticale, che dipende soltanto dall'eventuale attivazione dell'interlace.

AmigaDOS

È la parte del sistema operativo che controlla le operazioni con il sistema dei dischi e dei file. È una libreria condivisa nella quale sono raccolte tutte le funzioni di accesso al sistema dei file. Dal punto di vista dell'utente, l'AmigaDOS offre un insieme di comandi, come COPY, DISKCOPY e DELETE, impartibili da CLI (Command Line Interface, interfaccia linea comando). Tali comandi rappresentano il livello più alto del sistema operativo. Le funzioni dell'Exec costituiscono il livello immediatamente inferiore. Il livello di controllo più basso si raggiunge quando si interagisce direttamente con l'hardware.

Animazione a controllo di movimento

Costituisce uno dei due tipi di animazioni di alto livello dell'Amiga. Le animazioni a controllo di movimento impiegano sequenze di valori per assegnare la posizione, la velocità e l'accelerazione degli oggetti d'animazione sullo schermo video. L'animazione a disegni in sequenza è il secondo tipo di animazione di alto livello.

Animazione a disegni in sequenza

Costituisce uno dei due tipi di animazioni di alto livello dell'Amiga. Per ottenere l'animazione a disegni in sequenza, bisogna predisporre diverse immagini dell'oggetto da animare, e determinare quando e come queste immagini debbano essere presentate sullo schermo video. Si confronti con l'animazione a controllo di movimento.

Animazione di playfield

Costituisce uno dei due tipi di animazioni di basso livello dell'Amiga. Nell'animazione di playfield si sfruttano le caratteristiche di rapido movimento dati fornite dal canale DMA del Blitter, al fine di spostare singoli bob molto velocemente attraverso lo schermo video. Il Blitter sposta i dati in maniera abbastanza rapida da fornire l'illusione di un'effettiva animazione.

Assegnare un segnale

Ogni task dispone di un pool di 32 bit di segnale, di cui 16 sono riservati al sistema. Ogni elemento del sistema, compreso lo stesso task interessato, può attivare uno di questi segnali con l'intento di avvisare un task di un evento. Se il task è in attesa che quel segnale venga attivato, quando ciò accade riottiene il controllo. Se invece non è in attesa, quando il segnale viene attivato non se ne può rendere conto. Comunque, se il task entra in attesa di un segnale che è già stato attivato, riottiene subito il controllo. La funzione per entrare in attesa che uno o più segnali vengano attivati è Wait. I task possono allocare qualsiasi bit di segnale che non sia già in uso, e assegnarlo a una message port propria o altrui. Assegnare un segnale a una message port significa predisporla in modo che quando sopraggiunge un messaggio nella sua coda, il task che la possiede riceva quel segnale; se questo accade e il task era in attesa proprio di quel segnale, riottiene il controllo.

Attivare un segnale

Attivare un segnale di un task significa impostare a 1 il corrispondente bit di segnale nella struttura Task che definisce quel task. Se il task è in attesa che un segnale venga attivato, quando ciò accade riottiene il controllo. I segnali di un task possono essere attivati, o inviati, dalle message port del task, dalle sue routine di interrupt e di exception, da altri task o dal sistema. Si noti che dire "attivare un segnale" o "inviare un segnale" è esattamente lo stesso.

Attributo di testo

Sono le diverse caratteristiche di una fonte-carattere. Gli attributi di testo includono la larghezza, l'altezza e lo stile della fonte. Tali attributi sono contenuti nella struttura TextAttr, a cui ricorre il sistema per cercare nella memoria e nei dischi l'appropriata definizione della fonte.

Barra menu

Una barra menu di Intuition è la striscia di menu che appare sullo schermo quando, dopo aver selezionato una finestra dotata di menu, si preme il pulsante destro del mouse. Dal punto di vista software, è una raccolta di definizioni di strutture richiesta per la definizione completa dei menu, delle voci e sotto-voci dei menu. Il risultato delle definizioni delle strutture è una barra nella quale compaiono i nomi dei menu disponibili. Aprendoli, appaiono sullo schermo le rispettive voci, a ognuna delle quali può corrispondere un sotto-menu costituito da altre voci, dette sotto-voci. Il modo per collegare insieme le strutture della barra menu è fornito da Intuition.

Bassa risoluzione

Questo modo video prevede 320 pixel di risoluzione orizzontale sullo schermo video; non influenza la risoluzione verticale, che dipende soltanto dall'eventuale attivazione dell'interlace.

Bit di segnale del task

Ogni task può inviare messaggi e riceverne nelle sue message port. Ogni message port può essere predisposta dal task per avvisarlo con un particolare segnale ogni volta che nella sua coda sopraggiunge un messaggio. L'operazione d'invio di un messaggio corrisponde a impostare a 1 il corrispondente bit di segnale nella struttura Task che definisce il task. I bit di segnale consentono ai task di comunicare tra loro. Ogni struttura Task contiene una variabile a 32 bit (tc_SigAlloc) che indica quali segnali sono stati allocati dal task. Sedici di questi bit sono riservati al sistema, mentre gli altri sedici possono essere

allocati e assegnati a piacimento. Ben più importante è il parametro tc_SigRecvd, una long word composta da 32 bit di segnale. Quando una message port opportunamente predisposta riceve un messaggio nella sua coda. è in questo parametro che imposta il bit di segnale assegnatole dal task.

Bitmap

Una bitmap è un'area di memoria composta da bitplane, cioè da diversi piani di bit che l'hardware video considera sovrapposti nella composizione dell'immagine bitmap. Una bitmap può includere fino a sei bitplane; il numero di bitplane corrisponde al numero di bit associabili a ogni pixel della bitmap, e controlla quindi il numero di stati (colori) che ogni pixel può assumere. Per esempio, con due bitplane ogni pixel è definito da due bit e può dunque assumere quattro stati diversi, corrispondenti ai primi quattro registri di colore della tavolozza (palette) in uso. I bitplane che nel loro complesso costituiscono la bitmap possono risiedere in qualsiasi posizione nella chip RAM, ed è indifferente che siano adiacenti in memoria. Ogni bitplane, però, dev'essere rappresentato da un'area di memoria continua. In generale i termini bitmap, raster e raster bitmap, vengono considerati equivalenti, ma in questo volume abbiamo impiegato esclusivamente il termine bitmap per evitare inutili confusioni al lettore.

Bitplane

Si tratta di un'area di memoria continua, che viene trattata come un'area fisica rettangolare. Le bitmap vengono create come insiemi di bitplane, o piani di bit, sovrapposti (si veda la voce *Bitmap*). Ogni bitplane, quindi, occupa in memoria lo stesso numero di bit della bitmap a cui appartiene. Per definire una bitmap si possono creare in memoria fino a sei bitplane. Si noti che i bitplane che costituiscono una bitmap possono trovarsi in qualsiasi posizione all'interno dello spazio indirizzabile dai chip dedicati dell'Amiga (la chip RAM).

Blitter

È uno speciale coprocessore contenuto nel chip dedicato alle animazioni grafiche; gestisce i dati in modo molto veloce. Il Blitter controlla un canale DMA (Direct Memory Access, accesso diretto alla memoria) dell'Amiga e può trasferire milioni di pixel da un'area a all'altra della memoria in un secondo. Per via della sua elevata velocità, il canale DMA del Blitter viene impiegato per copiare dati e per tracciare linee. Il Blitter viene usato anche per gestire i bob (oggetti del Blitter) adoperati nelle animazioni grafiche. Il dispositivo TrackDisk impiega il Blitter per codificare e decodificare i dati delle tracce dei dischi.

Bob

Bob è l'acronimo di "Blitter object" (oggetto del Blitter), ovvero un oggetto disegnato attraverso il canale DMA del Blitter. Ogni parte di un componente d'animazione, e quindi ogni parte di un oggetto d'animazione, è un bob. Ciascun bob è definito da una struttura Bob.

Canale DMA

Un canale DMA (Direct Memory Access, accesso diretto alla memoria) fornisce un modo rapido per spostare dati in memoria all'interno del sistema senza impegnare la CPU. Il Blitter costituisce il miglior esempio di come un canale DMA possa essere utilizzato per rendere più veloce lo spostamento di dati tra due blocchi di memoria. L'Amiga fornisce fino a 25 canali DMA, a cui i programmi possono ricorrere per migliorare le loro prestazioni.

Caricare una view

Caricare una view significa trasferire le istruzioni Copper che la definiscono nell'hardware che si occupa dell'aggiornamento del video. Questa operazione produce l'apparizione della view sullo schermo video. Viene svolta tramite la funzione LoadView della libreria Graphics, la quale dice al sistema qual è la definizione di bitmap da utilizzare per il quadro video successivo e quali sono le istruzioni Copper.

Catena di server di un interrupt

È una catena di routine di interrupt (server), che al limite può essere formata anche da una sola routine. Ogni server (routine di servizio) in tale catena ha una propria priorità. Quando si verifica l'interrupt, l'ordine di esecuzione delle routine di interrupt è determinato dalla priorità che ognuna ha indicato nella propria struttura Interrupt. Lungo la catena, ogni server ha la facoltà di consentire o meno l'esecuzione delle routine di interrupt che seguono, cioè che possiedono priorità inferiore.

Clip

Si veda la voce *Delimitare*.

Coda delle richieste di I/O

Questo tipo di coda interviene nello scambio di messaggi fra task e dispositivi, e accoda messaggi costituiti da strutture IORequest. Questi

particolari messaggi, detti richieste di I/O, vengono inoltrati dal task al dispositivo per effettuare una richiesta di I/O. Il messaggio viene inserito in fondo alla coda delle richieste di I/O che fa capo all'unità del dispositivo a cui il messaggio è rivolto. Le richieste di I/O vengono poi elaborate secondo l'ordine d'arrivo (First In First Out) nel momento in cui raggiungono la sommità della coda (cioè della lista) relativa al dispositivo considerato.

Coda di attesa

Nell'Amiga i task hanno la possibilità di scambiare messaggi fra loro o con i dispositivi di I/O. Lo scambio dei messaggi avviene attraverso le message port, "porti" predisposti per accodare tutti i messaggi che ricevono e per offrirli a chi li richiede. Ogni message port dispone quindi di una "coda di attesa", nella quale i messaggi rimangono fino a quando non giungono alla sommità e vengono rimossi (si può anche rimuovere un messaggio non ancora giunto alla sommità della coda). Queste code sono in pratica liste a doppia concatenazione gestite in modo FIFO. Le code sono fondamentali nello scambio dei messaggi, in quanto non sempre il ricevente è pronto per elaborare i messaggi alla stessa velocità con cui gli giungono. L'accodamento dei messaggi viene gestito automaticamente dal sistema.

Collisione

Una collisione consiste in una sovrapposizione tra due elementi grafici o tra un elemento grafico e un confine di playfield. Le rilevazioni vengono effettuate dal sistema, che è in grado d'individuare collisioni tra qualsiasi combinazione di bob, sprite virtuali e confini di schermo.

Colore di penna

È il valore contenuto nel registro di colore assegnato a una data penna di disegno. Il sistema grafico utilizza tre penne: quella per il disegno di primo piano, determinata attraverso FgPen (oppure FrontPen, in ambiente Intuition); quella per il disegno di fondo, determinata attraverso BgPen (oppure BackPen, in ambiente Intuition); quella per contornare le aree, determinata attraverso AOlPen. Inoltre Intuition utilizza le penne DetailPen e BlockPen.

Comandi dei dispositivi

Sono comandi impiegati per manipolare dati provenienti dai dispositivi o a essi diretti. I comandi standard includono CMD_FLUSH, CMD_READ, CMD_WRITE, CMD_RESET, CMD_UPDATE, CMD_CLEAR, CMD_START e CMD_STOP. Ogni dispositivo possiede inoltre alcuni comandi specifici.

COMPLEMENT

COMPLEMENT è uno dei quattro modi di disegno impiegati dalle funzioni di disegno. Per ogni bit impostato a 1 nella matrice grafica (per i riempimenti di aree), o di continuità (per le linee), della penna di primo piano in uso (FgPen), viene eseguita un'operazione di complemento. Lo stato del bit viene perciò invertito (i bit impostati a 0 passano a 1 e viceversa).

Componente d'animazione

È il secondo livello, partendo dall'alto, nella gerarchia degli elementi d'animazione. I componenti d'animazione sono formati da bob. Un gruppo di componenti d'animazione può definire un oggetto d'animazione.

Composizione del colore

La composizione del colore è il processo tramite il quale l'hardware video definisce il colore di un pixel di schermo basandosi su una serie di bit a esso relativi. Il colore che assume il pixel è quello indicato dal registro colore a cui il pixel è associato. L'associazione avviene nel modo seguente: ogni bitmap è composta da una serie di bitplane; a ogni pixel della bitmap corrisponde un valore binario ottenuto componendo i bit dei bitplane la cui posizione nell'area di memoria corrisponde alla posizione del pixel sul video. Il numero dei bit che costituiscono quel valore binario dipende quindi dal numero di bitplane che costituiscono la bitmap. Per esempio, con una bitmap a quattro bitplane a ogni pixel può essere associato un valore numerico compreso fra 0 e 16. Questo valore numerico indica il numero del registro di colore dal quale dev'essere estratto il colore da attribuire a quel pixel.

Copper

È un coprocessore per scopi speciali contenuto nel chip dedicato all'animazione; le sue istruzioni controllano l'hardware interessato alla presentazione video. Ogni parte di un quadro video risulta definita da una lista di istruzioni Copper. Le istruzioni Copper sono inserite in RAM ed eseguite dal Copper, parallelamente alle istruzioni del 68000, per produrre la presentazione video. L'azione del coprocessore Copper è sincronizzata con il movimento del pennello elettronico sulle linee di scansione dello schermo.

Coppia di coordinate

È una coppia di numeri nella forma "x, y"; dove x e y sono le distanze orizzontali e verticali rispetto all'origine del sistema di coordinate impiegato (bitmap, view, viewport, schermo, finestra...). Coppie di coordinate vengono

impiegate da diverse funzioni delle librerie Intuition e Graphics per inserire immagini in una bitmap o in un elemento video di Intuition.

Delimitare (Clip)

Delimitare significa troncare un'operazione di disegno nel momento in cui il disegno si spinge al di fuori di una particolare area associata a una bitmap. L'area di delimitazione può essere composta da un unico rettangolo o da un insieme di rettangoli che formano una "regione di delimitazione".

Dispositivo

Dal punto di vista dell'hardware, con il termine dispositivo ci si riferisce sempre a uno strumento hardware predisposto per compiti specifici. Per esempio i disk drive, la tastiera, il mouse e ogni altro congegno collegabile all'Amiga. In termini software, un dispositivo è un particolare tipo di libreria necessaria per interfacciare il sistema con un particolare dispositivo hardware. Questa libreria viene definita "attiva" in quanto, oltre a raccogliere una serie di funzioni, è anche in grado di ricevere comandi dai task e d'indirizzarli verso specifiche unità logiche e fisiche del dispositivo hardware da essa controllato. Un dispositivo di I/O è definito da una struttura Device, in tutto uguale alla struttura Library che definisce le normali librerie condivise dell'Amiga. Per questa loro somiglianza con le librerie condivise, anche i dispositivi di I/O vengono installati nel sistema tramite la funzione MakeLibrary.

Dispositivo di input

I dispositivi di input dell'Amiga includono la tastiera, i disk drive dei dischi, il controller dell'hard disk, il mouse e qualsiasi altro dispositivo collegato ai connettori di espansione della macchina.

Dispositivo di output

I dispositivi di output dell'Amiga includono il monitor, i disk drive dei dischi, il controller dell'hard disk, e qualsiasi altro dispositivo collegato ai connettori di espansione della macchina.

Double-buffer

La tecnica double-buffer consiste nell'impiego di due distinte aree di memoria per una stessa immagine, in modo che mentre una è sotto il controllo dell'hardware video che preleva da essa le necessarie definizioni (liste di istruzioni Copper, bitplane...), l'altra può essere nel frattempo predisposta per

il quadro successivo. Questo sistema permette l'aggiornamento di un'area di memoria mentre l'altra è occupata a visualizzare la schermata che appare effettivamente sul video. Sebbene tale tecnica impegni il doppio della normale memoria video, consente di ottenere una maggiore omogeneità e di evitare effetti di sfarfallio dello schermo.

Dual-playfield

Questo modo video consente di gestire due separate memorie video, fornendo contemporaneamente due playfield indipendenti l'uno dall'altro. Si può regolare la priorità video di ogni playfield per determinare quale dei due è destinato ad apparire davanti e quale dietro. Il playfield anteriore possiede una parte "trasparente", che lascia intravedere il secondo playfield. Si possono fare scorrere i due playfield indipendentemente, producendo un effetto tridimensionale dinamico.

Elemento d'animazione

Può trattarsi di uno sprite hardware, di uno sprite virtuale, di un bob (oggetto del Blitter), di un componente d'animazione o di un oggetto d'animazione. Si tenga conto che soltanto i bob possono essere uniti per formare componenti d'animazione e oggetti d'animazione.

Evento di input

Ogni dispositivo hardware collegato all'Amiga (tanto internamente quanto esternamente) genera eventi di input. I segnali hardware relativi a questi eventi vengono rilevati dai dispositivi software predisposti per il controllo dei dispositivi hardware, e sottoposti all'attenzione del sistema.

Exception

Un'exception è una condizione anomala che si produce nel corso dell'attività di un task, e che ne provoca la sospensione per lasciare il posto a uno speciale segmento di codice predisposto per la gestione dell'exception.

Finestra

Una finestra è un'area rettangolare che può essere aperta su uno schermo di Intuition. Ciascuna finestra può essere dotata di un insieme di gadget di sistema o definiti dal programmatore; tali gadget permettono all'utente di operare sulla finestra. Ogni presentazione video di Intuition può consistere di uno o più schermi, e su ogni schermo può essere aperto un qualsivoglia numero

di finestre, compatibilmente con la memoria disponibile. In ogni istante può essere attiva al più una sola finestra: la finestra che riceve gli input dall'utente.

Finestra attiva

In ambiente Intuition, la finestra attiva è quella che riceve l'input generato dall'utente attraverso la tastiera e il mouse. In ogni istante non può esserci più di una finestra attiva; tutte le altre sono inattive. L'utente può cambiare la finestra attiva premendo il pulsante sinistro del mouse quando il puntatore si trova all'interno di una finestra inattiva.

Finestra backdrop

Questo tipo di finestra di Intuition rimane ancorata in fondo allo schermo. Tutte le altre finestre si sovrappongono a quella backdrop. In tal senso, le finestre backdrop posseggono la priorità video più bassa rispetto a qualsiasi altra finestra di Intuition.

Finestra superbitmap

In ambiente Intuition, è il tipo di finestra che usa una propria bitmap separata (cioè diversa da quella di schermo) per contenere le informazioni che appaiono sullo schermo e per effettuare autonomamente il refresh delle parti cancellate.

Gadget

Un gadget è un dispositivo grafico multiuso di input, utilizzato da Intuition. La maggior parte dell'input destinato a un'applicazione giunge attraverso i gadget presenti nelle finestre e nei requester di cui l'applicazione si serve. Ogni gadget di Intuition è definito da una struttura Gadget. Esistono i gadget standard, che Intuition associa a richiesta alle finestre dei programmi, e i gadget personalizzati definiti dai programmatori.

Gel

È l'acronimo di Graphics element (elemento grafico). Nel sistema Amiga ci sono due tipi di gel: gli sprite virtuali e i bob. Tutti gli elementi grafici vengono collocati in una lista, alla quale il sistema ricorre per eseguire il disegno nell'ordine opportuno.

Gimmezerozero

È un tipo speciale di finestra di Intuition che consiste praticamente in due diverse finestre: una interna e una esterna. La finestra esterna comprende (oltre alla finestra interna) la barra titolo, i gadget e i bordi. Quella interna contiene ogni altra informazione di disegno. Il vantaggio di questo tipo di finestra è che l'origine del sistema di coordinate è il vertice in alto a sinistra della finestra interna, così che non è possibile disegnare per sbaglio sulla barra titolo, sui gadget e su tutte le aree della finestra sulle quali non si dovrebbe disegnare.

Handler di un interrupt

Questo tipo di routine gestisce sempre e soltanto un particolare interrupt. Non appartiene a una catena di server, e quindi è l'unica responsabile dell'elaborazione dell'interrupt. Una volta che ha terminato di svolgere i propri compiti, esegue l'istruzione RTS (ritorno da subroutine) della CPU. Talvolta questa routine viene adoperata per inviare un segnale verso un task in attesa, in modo da riattivarlo.

Hold And Modify

Questo modo video consente un'estesa selezione di colori. In questa modalità è possibile presentare contemporaneamente sullo schermo fino a 4096 colori.

Icona

Un'icona è un'immagine grafica sullo schermo dell'Amiga. Il programma *Workbench* impiega le icone per mostrare una piccola immagine associata a un disco, a un programma, a un file project generato da un programma, a una directory o al Trashcan. L'utente può avviare l'esecuzione di un programma attraverso la doppia selezione della corrispondente icona.

IDCMP

È l'acronimo di Intuition Direct Communications Message Port, message port per le comunicazioni dirette con Intuition. È il percorso fondamentale per l'input dell'utente, attraverso la finestra attiva, nei programmi che funzionano in ambiente Intuition. Questo meccanismo prevede che la finestra in uso abbia associate due particolari message port. La prima, la user port, è quella nella quale il task deve prepararsi a ricevere i messaggi di Intuition relativi agli eventi che il task stesso ha indicato tramite il flag IDCMP. La seconda, la window port, è la reply port alla quale il task deve restituire i messaggi che riceve da Intuition.

Inizializzare

Significa stabilire i dati iniziali in una struttura o più in generale in un blocco di memoria. La maggior parte delle librerie software contengono un insieme di funzioni destinate all'inizializzazione delle strutture.

Interlace

Accorgimento dell'hardware tramite il quale si riescono a visualizzare schermi video composti da 512 linee di scansione con quadri da 256 linee. Il sistema prevede che ogni schermo in interlace sia ottenuto visualizzando due quadri da 256 linee, ma con il secondo sfalsato di mezza linea rispetto al primo.

Interrupt hardware

È un'interruzione della CPU che viene effettuata su esplicita richiesta di un dispositivo hardware. I dispositivi hardware inoltrano richieste di interrupt per spostare l'attenzione della CPU su un evento. Per esempio, il sistema risponde agli interrupt che provengono dal disco, dall'hard disk, dal mouse... Quando la CPU può sospendere quanto sta facendo per soddisfare la richiesta di interrupt, cede il controllo alla routine di gestione dell'interrupt corrispondente alla richiesta sopraggiunta. Questo accade quando la CPU sta funzionando a un livello di interrupt inferiore a quello della richiesta pervenuta (il livello 0 indica che la CPU non sta elaborando alcun interrupt).

Interrupt software

È un interrupt della CPU prodotto dal software sistema o da un programma.

Intervallo di vertical-blanking

È il periodo di tempo durante il quale il pennello elettronico del tubo a raggi catodici, dopo essere giunto alla fine di un quadro video, si spegne e torna alla posizione di partenza per essere pronto a disegnare il quadro successivo. Esiste sia un intervallo di blank verticale che un intervallo di blank orizzontale. Quello verticale viene adoperato dalle routine di sistema e dal coprocessore Copper per definire il quadro video successivo.

INVERSVID

È uno dei quattro modi di disegno disponibili. Quando è specificato INVERSVID, tutti i bit dell'oggetto vengono invertiti (i bit impostati a 0 divengono 1 e viceversa) prima che la sua immagine sia trasferita nella bitmap

ricevente. INVERSVID può essere usato con JAM2 per produrre caratteri di testo in modo inverso.

Inviare un segnale

Si veda la voce *Attivare un segnale*.

JAM1

È uno dei quattro modi di disegno disponibili. Quando è impostato, nelle operazioni di disegno un solo colore viene sovrapposto nella bitmap, impiegando il colore della penna di primo piano (FgPen).

JAM2

È uno dei quattro modi di disegno disponibili. Quando è impostato, nelle operazioni di disegno due colori vengono inseriti nella bitmap, impiegando i colori della penna di primo piano (FgPen) e della penna di sfondo (BgPen).

Layer

Un layer è una sezione di bitmap che può essere portata sullo schermo oppure tolta mediante un insieme di funzioni della libreria Layers. I layer, detti anche strati, sono fra loro sovrapponibili, e vengono impiegati da Intuition per creare le finestre.

Liberare

Liberare o rilasciare memoria significa rendere disponibile un'area di memoria che era stata allocata e di cui non si ha più bisogno. La memoria viene così restituita al pool di memoria libera. Nell'Amiga è opportuno che i programmi liberino quanto prima la memoria che non impiegano più.

Libreria

Una libreria è fondamentalmente una raccolta di funzioni. Si compone di due parti principali: le funzioni, dislocate ovunque nella memoria, e il modulo di controllo, sempre in RAM. Il modulo di controllo, a sua volta, è composto rispettivamente da una tavola di vettori di salto (istruzioni jmp $XXXXXXXX che individuano tutte le funzioni della libreria) dalla struttura Library di definizione della libreria e dall'area dati. L'indirizzo di memoria dove inizia la struttura Library costituisce l'indirizzo base della libreria. È tramite questo

indirizzo assoluto che in modo relativo si riescono a individuare tutti i parametri della struttura Library (tramite offset negativi) e tutti i dati che la seguono nell'area dati (tramite offset positivi). Le funzioni delle librerie possono essere condivise da diversi programmi. In effetti, l'intero sistema operativo dell'Amiga è strutturato in librerie di funzioni, alle quali chiunque può aggiungerne di proprie. Ci sono due tipi di librerie, quelle residenti in ROM e quelle residenti su disco. Le librerie residenti in ROM sono quelle a cui il sistema ricorre più spesso, mentre quelle meno utilizzate si trovano sul disco al fine di non ingombrare la ROM; vengono automaticamente trasferite in memoria non appena un task richiede di accedervi.

Lista

Una lista è una concatenazione di nodi disposti ovunque in memoria. Si dice "semplice" quando in ogni nodo è contenuto soltanto l'indirizzo del nodo successivo, mentre si dice "doppia", o a doppia concatenazione, quando ogni nodo possiede anche l'indirizzo del precedente. Una lista semplice è percorribile soltanto in un senso, mentre una lista doppia in entrambi.

Lista a doppia concatenazione

Una lista a doppia concatenazione è una lista all'interno della quale ogni nodo possiede un puntatore al nodo successivo e un puntatore al nodo precedente. Come effetto di questa disposizione, i nodi possono essere aggiunti o rimossi senza distruggere la continuità della lista, e la lista può essere percorsa in entrambi i sensi.

Lista dei danni

Una lista dei danni è un insieme di rettangoli di delimitazione che definiscono la porzione nascosta di una finestra a refresh semplice. Ogni lista dei danni è definita da una struttura DamageList. In ambiente Intuition, il sistema impiega la struttura DamageList per ricostruire le parti di una finestra a refresh semplice tornate visibili, in seguito a spostamenti o ridimensionamenti effettuati dall'utente.

Lista di istruzioni Copper

Una lista di istruzioni Copper è un insieme di istruzioni MOVE, WAIT e SKIP che indica al coprocessore come controllare il pennello elettronico del tubo a raggi catodici nella definizione di ogni pixel dello schermo video. La lista viene ordinata dal sistema in ordine crescente di coordinate (y, x). Ciò permette al Copper, a un certo punto della presentazione video, di cambiare colori e risoluzione.

Lista di sistema della memoria libera

La lista di sistema della memoria libera tiene conto di tutti i blocchi di memoria libera presenti nel sistema; l'insieme di questi blocchi, che possono essere sparsi ovunque, viene anche chiamato "pool di memoria libera del sistema". Ciascun task può inoltre allocare dal pool di memoria libera del sistema un'area di memoria sufficiente per tutte le sue esigenze, e creare una propria lista di memoria libera per gestirla in modo esclusivo.

Lista FIFO

Si tratta di una lista a doppia concatenazione in cui il primo elemento inserito è il primo a uscire (FIFO è l'acronimo di First In First Out). Una lista gestita in questo modo viene definita "coda", e nell'Amiga viene per lo più utilizzata per accodare i messaggi che giungono alle message port. In contrapposizione a questo tipo di lista, esiste la lista LIFO (Last In First Out), che viene anche definita stack (o pila).

Lista semplice

Si tratta di una lista di nodi nella quale ogni nodo contiene soltanto l'indirizzo del nodo successivo. In questo modo il legame che si instaura fra i nodi viene detto "semplice", e consente di percorrere la lista in un solo senso.

Maschera dei segnali da attendere

È un parametro da 32 bit che specifica quali segnali un task può accettare ed elaborare. Questo parametro è definito per ogni task nella relativa struttura Task.

Memoria chip

La memoria chip è costituita dal primo megabyte dello spazio indirizzabile. Si tratta della sola area di memoria alla quale i chip dedicati dell'Amiga possono accedere per prelevare e memorizzare informazioni. Questi chip talvolta entrano in concorrenza con il bus della CPU, producendo quindi un rallentamento nell'accesso alla memoria da parte della CPU. Gli altri tipi principali di memoria sono la fast e la public.

Memoria fast

La memoria fast (veloce) è quella che si trova al di là del limite costituito dal primo megabyte dello spazio indirizzabile. Include tutta la memoria indirizzabile, fino al limite massimo per l'Amiga (8 MB). Dal momento che i chip dedicati dell'Amiga non possono accedere a questa regione, la CPU vi accede più rapidamente, ed è per questo che viene definita veloce.

Memoria libera

La memoria libera è memoria di cui nessuno nel sistema si è impossessato allocandola, e quindi rappresenta una risorsa disponibile. Questa memoria viene controllata attraverso la lista di sistema della memoria libera. Tanto i programmi quanto le routine di sistema possono allocarla secondo le proprie esigenze.

Memoria public

Questa memoria è condivisa da diversi task e dispositivi nel sistema. Un blocco di memoria public può essere assegnato in ogni punto dello spazio di memoria dell'Amiga (fino a 8MB).

Memoria video

La memoria video consiste in una o più aree di memoria tenute da parte per contenere le bitmap e i bitplane necessari per definire i vari schermi video dell'Amiga. Queste aree di memoria vengono assegnate dalla funzione AllocRaster della libreria Graphics. Tutti i bitplane e le bitmap devono trovarsi nella chip RAM.

Message port

È un meccanismo software gestito dalle funzioni dell'Exec e dedicato alle comunicazioni tra i task. Ogni message port è definita da una struttura MsgPort. Task e dispositivi possono ricevere messaggi attraverso le proprie message port, e inviare messaggi a qualunque message port. Diverse strutture posseggono puntatori a strutture MsgPort o le includono direttamente, al fine di consentire scambi di messaggi tra le diverse strutture.

Messaggi dei task

I task comunicano l'un l'altro utilizzando messaggi che vengono passati attraverso le message port definite nelle rispettive strutture MsgPort. Quando

un messaggio arriva in una message port di un task può causare l'invio di un segnale.

Messaggio di risposta

Si tratta di un messaggio inviato da un task a un altro, oppure da un dispositivo a un task. Quando un task, oppure un dispositivo, riceve un messaggio nella coda alla propria message port, viene generalmente avvisato. Quando poi entra in possesso di quel messaggio e lo elabora, per convenzione lo deve restituire al mittente, eventualmente modificandolo per formulare una risposta. La restituzione al mittente viene effettuata tramite la funzione ReplyMsg, la quale inserisce il messaggio nella coda alla reply port del mittente (indicata nel messaggio). L'azione di restituzione del messaggio è fondamentale per due motivi: serve per avvisare il mittente che il destinatario ha ricevuto il messaggio e l'ha elaborato, e riporta la struttura che definisce il messaggio sotto il dominio di chi l'ha allocata (in genere il task mittente).

Messaggi per i dispositivi

Sono particolari tipi di messaggi che i task inviano ai dispositivi di I/O per formulare richieste di I/O. L'ossatura di questi particolari messaggi è costituita da una struttura IORequest, che spesso è ampliata e quindi costituisce solo il primo elemento di strutture di I/O più complesse. Ogni dispositivo prevede una certa struttura di I/O che dev'essere inizializzata per formulare la richiesta.

Modo di disegno

Il modo di disegno è rappresentato dai parametri della struttura RastPort associata alla bitmap nella quale si desidera disegnare. Tali parametri indicano come devono essere inseriti i dati nella bitmap nel corso del disegno. Esistono quattro modi di disegno: JAM1, JAM2, INVERSVID e COMPLEMENT.

Modo trace

È uno dei tre modi caratteristici della CPU; gli altri due sono il modo user e il modo supervisor. Il modo trace viene utilizzato per il debug dei programmi. In tale modalità viene forzata un'exception dopo che ogni istruzione è stata eseguita. In tal modo è possibile correggere il programma controllando la sua esecuzione passo dopo passo.

Modo video

L'Amiga possiede diversi modi video, e può impiegarli in diverse combinazioni. Nel modo a bassa risoluzione vengono adoperati 320 pixel per ciascuna linea di scansione; nel modo ad alta risoluzione vengono usati 640 pixel per ogni linea di scansione. In modo senza interlace si hanno 256 linee di scansione per ogni schermo (standard PAL); in modo interlace si hanno 512 linee di scansione per ogni schermo (standard PAL).

Multitasking

Si dice di un sistema quando è in grado di eseguire diversi task "simultaneamente". Ogni task si comporta come terminale virtuale unico e ai propri occhi possiede un completo controllo sulla CPU e sull'hardware del sistema. L'Amiga è dotato di un sistema operativo multitasking, nel quale è possibile mandare in contemporanea esecuzione un numero qualsiasi di task, compatibilmente con i limiti della memoria. Ogni task è associato a una struttura Task e gli viene assegnata una priorità. I task comunicano tra loro impiegando segnali generati via software; tali segnali vengono rilevati tramite le strutture Task. In ogni istante, però, è attivo un solo task. Lo scambio di controllo fra i task viene condotto tenendo conto della priorità d'esecuzione, dell'ingresso di nuovi task nel sistema e degli eventi di interrupt.

Nodo predecessore

È il nodo della lista che precede quello di riferimento.

Nodo successore

È il nodo della lista che segue quello di riferimento.

Oggetto d'animazione

Rappresenta il più alto livello nella gerarchia degli elementi d'animazione. Il lavoro del programmatore è rivolto appunto a ottenere oggetti d'animazione, che verranno poi spostati sullo schermo per i previsti effetti d'animazione.

Overscan video

È costituito dalla parte delle informazioni video (pixel sulle linee di schermo) che si trova al di là dell'area normalmente visibile. L'overscan video viene impiegato per evitare la distorsione dell'immagine sui lati; tale problema deriva dalla curvatura della superficie del monitor. In genere ci si riferisce

all'overscan orizzontale, ma esiste tuttavia anche un certo overscan verticale. Nelle produzioni video si preferisce lavorare in overscan per disporre d'immagini a tutto schermo.

Pixel

Il pixel è il più piccolo elemento d'informazione video presente sullo schermo. Alla risoluzione massima sono controllati sullo schermo 327.680 pixel (640 x 512, nello standard PAL).

Playfield

Il playfield è uno dei due elementi fondamentali della grafica dell'Amiga. Costituisce lo sfondo per gli elementi della presentazione. Il playfield è la parte statica del video e forma la superficie sulla quale si spostano gli sprite. View e playfield sono termini equivalenti.

Preferences

È un tool del disco Workbench. Tale programma permette all'utente di modificare i parametri di sistema. Alcuni gadget consentono all'utente d'impostare i colori, la cadenza di ripetizione dei tasti e altri particolari del modo di operare dell'Amiga. Queste impostazioni possono essere usate solo durante una sessione di lavoro o anche salvate per la volta successiva.

Primitive

Questo termine si riferisce all'intero insieme delle funzioni di libreria dell'Amiga. Ciò include tutte le funzioni di tutte le librerie discusse in questo libro.

Priorità del task

Si tratta di un numero compreso tra -128 e +127, usato per assegnare la priorità di elaborazione al task. Dev'essere memorizzato nella relativa struttura Task prima che il task sia inserito nel sistema. Maggiore è la priorità, maggiore è la precedenza di elaborazione. La priorità di un task può essere eventualmente modificata anche quando il task è già stato inserito nel sistema.

Priorità video

Ogni oggetto di schermo possiede una priorità video. Questa priorità determina quali oggetti (playfield, screen, finestre, bob e sprite) appaiono in primo piano e quali sullo sfondo. Gli oggetti a priorità più alta appaiono davanti a quelli con priorità inferiore. Inoltre, le funzioni della libreria Layers permettono di definire una presentazione a strati, costituita da diverse parti di schermi e finestre, ciascuna delle quali si sovrappone all'altra.

Pulsante destro del mouse

Il pulsante destro del mouse serve per far apparire la barra menu associata alla finestra attiva. Tenendolo premuto e portando il puntatore sulle voci della barra menu, si può visitare l'intera struttura di menu. Se il pulsante viene rilasciato quando il puntatore sta evidenziando una voce, Intuition interpreta l'evento come la selezione della voce.

Pulsante sinistro del mouse

Il pulsante sinistro del mouse serve per selezionare icone e per trasferire informazioni al sistema Intuition. In ambiente Workbench, la doppia pressione di questo pulsante quando il puntatore si trova sull'icona di un programma ne avvia l'esecuzione; la doppia pressione del pulsante sull'icona di un file di dati avvia automaticamente il programma da cui è stato prodotto quel file.

Puntatore

Questo termine possiede due significati: il puntatore di Intuition è lo sprite che sul video segue i movimenti del mouse e che serve all'utente per selezionare oggetti sullo schermo; il puntatore considerato nel linguaggio C è invece un particolare tipo di variabile che contiene l'indirizzo di memoria di una struttura, di un array e così via.

Quadro video

È costituito da una serie di 256 linee di scansione (nello standard PAL, 200 in NTSC). Ogni quadro viene prodotto in 1/50 di secondo. Senza interlace, un quadro video è equivalente allo schermo video. Con l'interlace, invece, occorrono due quadri video per visualizzare uno schermo (per creare 512 linee di scansione, infatti, l'hardware video visualizza due quadri da 256 linee sfalsandoli di mezza linea). Si noti quindi che la frequenza di ripetizione di uno schermo video in interlace è di 25 schermi al secondo, anche se la frequenza di ripetizione dei quadri video continua a essere di 50 quadri al secondo.

QuickIO

È una forma di input/output tra dispositivo e task che procede senza creare attese. Viene usato quando si desidera che una richiesta di I/O indirizzata a un dispositivo venga elaborata il più rapidamente possibile.

Raster

In generale i termini raster, bitmap, e raster bitmap, vengono considerati equivalenti, ma in questo volume abbiamo impiegato esclusivamente il termine bitmap per evitare inutili confusioni al lettore. Si veda la voce *Bitmap*.

Refresh

L'operazione di refresh consiste in generale nel ripristino della presentazione video, sulla base d'informazioni mantenute nella memoria video dell'Amiga. Molto spesso quest'operazione viene limitata alle aree di finestre che tornano visibili dopo essere state coperte.

Refresh avanzato

È un metodo per ripristinare le parti danneggiate delle finestre. Il sistema impiega diversi piccoli buffer per memorizzare le bitmap delle aree non visibili, aggiornando automaticamente tali buffer a mano a mano che la situazione varia. Si confronti con il refresh semplice e con il refresh superbitmap.

Refresh semplice

È un metodo per ripristinare le parti danneggiate delle finestre. Delega la ricostruzione al task che ha creato le finestre. Si confronti con il refresh avanzato e con il refresh superbitmap.

Refresh superbitmap

È un metodo per ripristinare le parti danneggiate delle finestre. Con il refresh superbitmap, le aree nascoste non devono essere ridisegnate; vengono semplicemente copiate da un'area bitmap (detta superbitmap) che contiene sempre e comunque l'intera immagine della finestra. Tale superbitmap è estesa almeno quanto quella per cui dev'essere effettuato il refresh, anche se può avere dimensioni superiori. Si confronti con il refresh semplice e con quello avanzato.

Regione

Una regione è una delimitazione geometrica impiegata per indicare alla libreria Graphics quali parti di una bitmap possono essere modificate da operazioni di disegno e quali invece devono rimanere intatte. Qualsiasi operazione di disegno che tenti di sconfinare al di fuori della regione viene bloccata. Una regione è costituita da un insieme di rettangoli di delimitazione.

Registro di colore

Un registro di colore mantiene la definizione di una delle 4096 tonalità disponibili. Ci sono 32 registri di colore da 16 bit. I quattro bit più significativi di ogni registro non vengono sfruttati. I dodici bit meno significativi definiscono l'intensità dei colori rosso, verde e blu: i colori fondamentali della sintesi additiva. Ciò permette 16 livelli d'intensità per ogni colore fondamentale, e quindi 4096 colori componendo i 16 livelli di ogni colore fondamentale in tutte le combinazioni possibili.

Render

È la parola inglese con la quale si indica tutto il complesso di operazioni necessarie a visualizzare un'immagine sullo schermo. Un'operazione di render si verifica quando viene chiamata la funzione LoadView per presentare le bitmap e le liste di istruzioni Copper all'hardware video dell'Amiga.

Reply port

È la message port attraverso la quale un task o un dispositivo ricevono le risposte ai messaggi che inoltrano.

Requester

Un requester è un box di dialogo di Intuition, utilizzabile per chiedere all'utente un intervento o una risposta, al fine di poter procedere con le successive azioni previste dal programma.

Scambio di controllo fra i task

È il processo di ripartizione del tempo di elaborazione della CPU tra i diversi task presenti nel sistema. Per esempio, si verifica uno scambio di controllo fra i task ogni volta che viene inserito nel sistema un task con priorità maggiore di quello in esecuzione, oppure quando il task che detiene il controllo entra in stato di attesa per ottenere una particolare risorsa.

Schermo

Questo termine indica uno schermo di Intuition, standard o personalizzato. Gli schermi di Intuition consentono ai programmi di aprire e chiudere finestre sul video senza limitazioni.

Schermo video

Questo termine si riferisce a tutto ciò che appare sullo schermo del monitor. Lo schermo video è costituito da alcuni dei seguenti elementi (non necessariamente tutti): playfield, schermi, finestre, bob, sprite.

Schermo Workbench

È lo schermo che viene aperto per default da Intuition. Presenta caratteristiche particolari, ed è indipendente dall'attivazione del programma *Workbench*. Comunque, quando il *Workbench* viene attivato tramite il comando LOADWB del CLI, utilizza l'omonimo schermo per aprire le proprie finestre e disporre le icone.

Scorrimento del playfield

Lo scorrimento del playfield consiste nello spostamento della presentazione video in senso verticale od orizzontale sullo schermo.

Segmento dati di libreria

È la terza parte del modulo di controllo di una libreria, quella che segue la struttura Library. In essa, le funzioni della libreria immagazzinano i dati di cui hanno bisogno.

Server di un interrupt

Si veda la voce *Catena di server di interrupt*.

Sprite

Sprite è il termine generico che identifica un oggetto creato per essere collocato sul campo video. Gli sprite sono larghi di norma 16 pixel, tuttavia alcuni pixel possono essere trasparenti e produrre quindi uno sprite apparentemente più stretto. Gli sprite possono assumere qualsiasi altezza, fino a occupare l'intera altezza dello schermo. Si vedano anche le definizioni di sprite hardware e sprite virtuale.

Sprite hardware

Il sistema Amiga possiede otto canali DMA per gli sprite. Gli sprite hardware sono oggetti mobili larghi 16 bit e di altezza che può raggiungere quella dell'intero schermo. Lo sprite hardware è limitato a tre colori, oltre alla trasparenza. Ciascuno degli otto sprite hardware può essere assegnato a uno sprite virtuale per accrescere il numero degli oggetti sprite mobili sullo schermo.

Sprite virtuali

Uno sprite virtuale è uno sprite definito da una struttura VSprite. Viene adoperato il termine "virtuale" perché lo sprite non è controllato direttamente dall'hardware. Ciò permette a un numero indefinito di sprite virtuali d'impiegare gli otto canali DMA destinati agli sprite hardware dell'Amiga. Le restrizioni su colori e dimensioni valide per gli sprite hardware si applicano anche agli sprite virtuali. Gli sprite virtuali possono essere usati nel rilevamento delle collisioni ma non possono essere utilizzati per definire componenti oppure oggetti d'animazione.

Struttura

È un tipo di variabile prevista dal linguaggio C e dai compilatori Assembly evoluti, che serve per definire un insieme di dati in memoria. Nell'Amiga ne viene fatto amplissimo uso, soprattutto nello scambio di dati con il sistema operativo. Viene impiegata per le librerie, i dispositivi, gli elementi di presentazione grafica e così via. Nella programmazione in C, a ogni parametro di struttura viene in genere assegnato un nome simbolico, facile da ricordare. Le strutture standard previste dal sistema sono definite nei file INCLUDE.

Struttura BitMap

Questa struttura contiene informazioni rivolte a definire la bitmap associata a una view, a una viewport, o un layer. Localizza in memoria i bitplane che costituiscono l'immagine e ne indica il numero.

Struttura bltnode

È la struttura adoperata dal sistema per porre in coda le richieste Blitter provenienti dai vari task di disegno. Inoltre, la struttura bltnode fornisce il modo per chiamare una routine Blitter definita dal programmatore, nel momento in cui viene eseguita la particolare richiesta Blitter. Le richieste Blitter vengono collocate in una coda di tipo FIFO. Tale sistema consente a più di un task di sfruttare il Blitter attraverso l'accodamento delle richieste.

Struttura IORequest

È una struttura dell'Exec destinata a contenere le informazioni necessarie per inviare messaggi ai dispositivi o ricevere le risposte. Questi messaggi vengono formulati sempre come comandi impartiti al dispositivo per effettuare uno scambio di dati. La struttura IORequest è in pratica un messaggio standard di comunicazione con i dispositivi.

Struttura Library

La struttura Library serve per definire una libreria, e concatenarla nella lista di sistema LibList delle librerie disponibili.

Struttura List

Si tratta di una struttura che viene impiegata per intestare liste a doppia concatenazione. Le liste sono costituite da strutture Node, e possiedono almeno un nodo di testa e un nodo di coda. Ogni struttura Node della struttura List contiene un puntatore alla struttura Node predecessore e alla struttura Node successore.

Struttura RastPort

È la struttura di dati impiegata per controllare il disegno all'interno di una bitmap. La struttura RastPort è probabilmente la più importante dell'intero sistema grafico. Contiene le informazioni per il controllo del disegno che definiscono come dev'essere disegnata ciascuna porzione di un quadro video. La struttura RastPort gioca un ruolo rilevante nelle funzioni grafiche con effetto statico, in quelle per l'animazione e in quelle destinate ad agire con i testi.

Struttura Task

È la struttura dati che definisce un task. Contiene informazioni per puntare a tutte le strutture necessarie per gestire un task. Contiene inoltre una struttura Node per la definizione della priorità del task, il suo nome, e per legare la struttura alle altre presenti nelle liste di sistema dei task.

Strutture pubbliche

Sono tutte le strutture che, contenendo una sotto-struttura Node, possono essere inserite in una lista ed essere identificate tramite il nome indicato nel parametro ln_Name della struttura Node. Una volta che il nome è stato assegnato, qualsiasi altro task può riferirsi a quella sotto-struttura Node tramite

il suo nome, utilizzando per esempio la funzione FindName. Esistono parecchi casi di strutture pubbliche, come la struttura Task per esempio. Conoscendone il nome, tramite la funzione FindTask si può ottenere l'indirizzo della struttura. Lo stesso vale per le strutture MsgPort che definiscono le message port. Il task che crea una message port ha la facoltà di scegliere se definirla come pubblica oppure no. In caso affermativo, la struttura MsgPort (tramite la sua sotto-struttura Node) viene inserita nella lista di sistema PortList, nella quale qualsiasi task la può rintracciare conoscendone il nome e chiamando la funzione FindPort. La funzione che aggiunge una struttura Task nel sistema rendendola pubblica è AddTask, mentre per la struttura MsgPort è AddPort.

Supervisor stack

È lo stack di sistema, utilizzato dalla CPU durante il funzionamento in modo supervisor. Per definizione, ce n'è uno solo in tutto il sistema.

Task

È costituito da un unico insieme di istruzioni software associate a una particolare struttura Task. Un task può essere una semplice subroutine, una routine di gestione degli interrupt, una routine di servizio di un interrupt, oppure parte di un programma applicativo più vasto.

Tavola dei colori

La tavola dei colori è un array che contiene l'insieme dei colori disponibili per una determinata viewport dello schermo video; si tratta dei colori che vengono memorizzati nei registri di colore dell'hardware video. A ogni pixel della viewport è associato un numero che lo fa corrispondere a uno dei 32 registri di colore. Ciascun registro rappresenta uno dei 4096 possibili colori. In questo modo, la tavola dei colori costituisce la palette dalla quale si può attingere per colorare i pixel di una viewport (che in Intuition è in realtà uno schermo). La palette è unica per tutto quanto appare all'interno della viewport, ma cambia nel passaggio a un'altra viewport, quando lo schermo video è suddiviso in diverse viewport (o schermi di Intuition).

Terminale virtuale

Questo termine possiede due significati. Nel primo è una finestra di Intuition destinata a ricevere input dall'utente e a fornire output. Nel secondo è un qualsiasi task presente nel sistema Amiga (non necessariamente dotato di finestra). Ogni task viene definito "terminale virtuale" in quanto pur coesistendo nel sistema con tutti gli altri, può ritenere di essere l'unico, purché venga creato sotto la completa osservanza di alcune fondamentali regole che

garantiscono la salvaguardia del sistema multitasking. Ciascun task del sistema agisce, nel momento in cui diviene attivo, come unico terminale virtuale; detiene il completo controllo della macchina (della sua memoria, del video, dei disk drive).

UART

È l'acronimo di Universal Asynchronous Receiver/Transmitter (trasmettitore/ricevitore asincrono universale). È costituito da un circuito e da un chip che controlla il collegamento seriale con i dispositivi periferici.

User port

È la message port creata quando s'impiega il metodo di comunicazione IDCMP di Intuition. Il programma utilizza questa message port per ricevere i messaggi IDCMP che Intuition gli invia quando l'utente genera eventi di input agendo da tastiera o da mouse e contemporaneamente è attiva la finestra del programma.

User stack

A ogni task viene assegnato uno stack detto user stack. Tale stack è adoperato per salvare e ripristinare i contenuti dei registri della CPU oltre che per conservare le variabili locali e gli indirizzi di ritorno delle subroutine.

Vettore

Questo termine ha due significati. Nel primo caso il vettore è un'istruzione di salto per la CPU 68000, usata per individuare particolari sezioni di codici nel sistema. Le funzioni delle librerie vengono individuate da vettori disposti in opportune tavole di salto. Lo stesso termine è adoperato anche per riferirsi a segmenti di linea sullo schermo video.

View

Caratterizza tutto quanto è visibile sullo schermo, dal momento che non è possibile visualizzare due o più view contemporaneamente. Ogni view può consistere in una o più viewport, le quali quindi possono spartirsi l'altezza complessiva della view. Ciascuna view è definita da una struttura View e possiede una propria risoluzione orizzontale e verticale. Tutte le viewport di una view vengono visualizzate con la risoluzione assegnata alla view di cui fanno parte.

Viewport

Una viewport è una sezione rettangolare dello schermo. Ciascuna viewport è in genere larga quanto l'intero schermo video, anche se può essere più stretta. La viewport costituisce un sotto-rettangolo, nel più esteso rettangolo costituito dalla view. Gli schermi di Intuition, che possono spartirsi l'altezza dell'intero schermo, sono tipici esempi di viewport, e pertanto ne seguono le regole. Si ricordi che ogni viewport possiede una propria tavola dei colori, e che all'interno di ogni viewport può essere presente un numero qualsiasi di layer (che in ambiente Intuition sono le finestre).

Window port

È la message port di una finestra nella quale Intuition si aspetta di ricevere in risposta i messaggi che ha inviato alla user port di quella stessa finestra. In questo senso, la window port costituisce la reply port di Intuition.

Word per la descrizione del colore

Sono coppie di word che definiscono l'immagine pixel per ogni linea di sprite hardware o virtuale. Entrambi i tipi di sprite sono larghi 16 pixel e possono essere alti come l'intero schermo in modo video non interlace.

Workbench

Questo programma serve per consentire all'utente d'interagire molto intuitivamente con il sistema dei dischi e dei file. L'interfaccia che usa è prettamente grafica: si serve di finestre, menu, icone, requester, allo scopo di creare un'interfaccia utente molto user-friendly. Le icone possono rappresentare dischi, directory, programmi applicativi, dispositivi o file generati da programmi. Le funzioni della libreria Icon forniscono un modo standard per trattare queste icone sullo schermo Workbench.

I modi grafici dell'Amiga

I modi grafici dell'Amiga

Oltre al modo single-playfield, trattato nel capitolo 2, ci sono altri quattro modi grafici. Si tratta dei modi dual-playfield, double-buffer, Hold And Modify e Extra Half-Brite.

Il modo dual-playfield

Il modo grafico dual-playfield consente di usare due playfield nello schema delle view. Questa modalità non va confusa con il modo double-buffer, che tratteremo più avanti nel corso del capitolo. Nel modo dual-playfield, i due playfield sono indipendenti l'uno dall'altro; le loro viewport seguono la maggior parte delle regole che interessano gli schermi single-playfield. È possibile impostare la priorità video relativa dei playfield, in modo da far apparire l'uno o l'altro in primo piano, indifferentemente.

Ecco le fasi per definire il modo grafico dual-playfield:

1. Si allocano due strutture BitMap che descrivono due bitmap indipendenti. Gli indirizzi restituiti dalle funzioni di allocazione della memoria devono essere conservati: verranno impiegati nelle strutture RasInfo e RastPort per puntare alle due strutture BitMap.

2. Si usa la funzione AllocRaster per allocare la memoria richiesta per le due bitmap. Una verrà adoperata per definire il playfield di primo piano (chiamato playfield 1); l'altra per definire il playfield di sfondo (chiamato playfield 2). Si possono predisporre fino a tre bitplane per ogni bitmap. Le due bitmap possono avere larghezza e altezza diverse.

3. Si alloca la memoria per due strutture RasInfo, che vengono collegate tramite un parametro puntatore Next. Esse definiscono i confini dei due playfield relativamente alle due bitmap.

4. Si alloca la memoria per due strutture RastPort, che verranno impiegate per controllare i disegni nelle due bitmap.

5. Si alloca la memoria necessaria per ciascuna struttura ViewPort prevista nella view. Per ogni viewport è necessaria una sola struttura ViewPort, dal momento che si provvede a impostare a DUALPF il parametro Modes nella struttura ViewPort per indicare il modo grafico dual-playfield. Una sola struttura ViewPort gestisce quindi entrambi i playfield di ciascuna viewport.

6. Si alloca la memoria per una struttura View. Entrambi i playfield vengono gestiti dalla stessa struttura View, dal momento che il parametro Modes nella struttura ViewPort è impostato a DUALPF per indicare il modo grafico dual-playfield.

7. Si adopera la funzione InitBitMap per inizializzare le due strutture BitMap. Gli indirizzi precedentemente restituiti dalle funzioni di allocazione della memoria vanno impiegati come puntatori alle due strutture BitMap.

8. Per inizializzare le strutture RastPort si usa la funzione InitRastPort. Gli indirizzi precedentemente restituiti dalle funzioni di allocazione della memoria vanno impiegati come puntatori alle due strutture RastPort. Il puntatore di bitmap nella prima struttura RastPort va inizializzato con l'indirizzo della prima struttura BitMap; il puntatore di bitmap nella seconda struttura RastPort va inizializzato con l'indirizzo della seconda struttura BitMap.

9. Si impiegano quattro istruzioni di assegnazione dei parametri di struttura per inizializzare due strutture RasInfo per le due bitmap. Il playfield 1 risulta definito dalla prima struttura RasInfo; il playfield 2 dalla seconda. Nella prima struttura RasInfo, il puntatore BitMap va impostato per puntare alla prima delle due strutture BitMap; il puntatore BitMap nella seconda struttura RasInfo va impostato per puntare alla seconda struttura BitMap. Inoltre, il puntatore Next nella prima struttura RasInfo, sarà impostato in modo da puntare alla seconda struttura RasInfo, collegando così le due strutture.

10. Si chiama la funzione InitVPort per inizializzare la struttura ViewPort precedentemente allocata. Si imposta a DUALPF il parametro Modes della struttura ViewPort (per indicare il modo grafico dual-playfield). Inoltre, se si vuole che il playfield 1 appaia in primo piano, il flag PFBA del parametro Modes va lasciato a zero; se lo si imposta, compare in primo piano il playfield 2. Si ricordi che ogni struttura ViewPort possiede un puntatore a una struttura RasInfo. Da parte sua, la struttura RasInfo possiede un puntatore a una struttura BitMap. La struttura ViewPort controlla quindi le due strutture RasInfo e BitMap. Il puntatore RasInfo della struttura ViewPort va impostato per puntare alla prima struttura RasInfo.

11. Si chiama la funzione InitView per inizializzare la struttura View precedentemente allocata. Quindi si inizializza il suo parametro ViewPort con l'indirizzo della viewport allocata.

12. Per definire le liste di istruzioni Copper relative a ciascuna viewport, si deve impiegare la funzione MakeVPort, la quale genera la lista Copper intermedia per la viewport indicata. Con questo risulta definita l'organizzazione delle diverse viewport della view.

13. Per unire insieme tutte le istruzioni Copper necessarie a definire la view, si chiama la funzione MrgCop. Ciò fonderà tutte le liste Copper intermedie nella lista Copper hardware della view.

14. Si chiama la funzione LoadView per caricare le istruzioni Copper nell'hardware video, al fine di produrre il modo grafico dual-playfield.

Ogni playfield del modo grafico dual-playfield è formato da uno, due o tre bitplane. I colori dei playfield (fino a sette, oltre a quello trasparente) vengono prelevati da differenti insiemi di registri colore. Le Tavole B.1, B.2 e B.3 definiscono con maggior precisione questa situazione. Si noti che il modo a bassa risoluzione può usare fino a tre bitplane in ogni playfield, mentre il modo ad alta risoluzione può usarne al massimo due. La Tavola B.3 mostra come vengono assegnati i bitplane ai due playfield. I bit provenienti dai bitplane identificati dal numero più alto risultano i più significativi nel numero binario che determina il registro di colore associato a un pixel.

Un modello di programmazione per il modo dual-playfield

Il listato che segue mostra le istruzioni necessarie per definire una presentazione dual-playfield. Questa sezione di programma può essere impiegata come modello per qualsiasi programma che faccia uso del modo dual-playfield.

L'esempio illustra una view costituita da una viewport. XOffset1, XOffset2, YOffset1, YOffset2 sono gli offset per le bitmap delle viewport, e devono essere scelti in base alla misura della bitmap e alla posizione della viewport nella bitmap. Per fissare la posizione della viewport nella view che costituisce lo

Tavola B.1: *Assegnazioni dei registri di colore nel modo grafico dual-playfield a bassa risoluzione*

Combinazione di bit	Playfield 1	Playfield 2
000	Registro di colore 0 (trasparenza)	Registro di colore 8 (trasparenza)
001	Registro di colore 1	Registro di colore 9
010	Registro di colore 2	Registro di colore 10
011	Registro di colore 3	Registro di colore 11
100	Registro di colore 4	Registro di colore 12
101	Registro di colore 5	Registro di colore 13
110	Registro di colore 6	Registro di colore 14
111	Registro di colore 7	Registro di colore 15

Tavola B.2: *Assegnazioni dei registri di colore nel modo grafico dual-playfield ad alta risoluzione*

Combinazione di bit	Playfield 1	Playfield 2
00	Registro di colore 0 (trasparenza)	Registro di colore 8 (trasparenza)
01	Registro di colore 1	Registro di colore 9
10	Registro di colore 2	Registro di colore 10
11	Registro di colore 3	Registro di colore 11

Tavola B.3: *Assegnazioni dei bitplane nel modo grafico dual-playfield*

Numero di bitplane usati	Numeri dei bitplane per il playfield 1	Numeri dei bitplane per il playfield 2
1	1	Nessuno
2	1	2
3	3, 1	2
4	3, 1	4, 2
5	5, 3, 1	4, 2
6	5, 3, 1	6, 4, 2

schermo video dell'Amiga, è anche necessario specificare il valore di yourposition1. La viewport è profonda tre bitplane, in modo che possano essere contemporaneamente mostrati otto colori.

Le direttive di compilazione #define descrivono un video in bassa risoluzione senza interlace. Si noti che viene definita una sola view. Si noti inoltre che il flag PFBA del parametro Modes, presente nelle strutture ViewPort e View, viene lasciato indefinito. Il sistema dispone il playfield 1 al di sopra del playfield 2; se si vuole ottenere la situazione opposta, il flag PFBA nella struttura ViewPort va impostato. Si osservi inoltre che le strutture sono allocate staticamente; comunque la struttura di programmazione ora illustrata continuerebbe a essere valida anche se venissero allocate dinamicamente.

```
/* INCLUDE */

#include "exec/types.h"
#include "exec/memory.h"
#include "exec/exec.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
```

```
#include "graphics/view.h"
#include "graphics/copper.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "graphics/text.h"
#include "hardware/blit.h"
#include "hardware/custom.h"
#include "hardware/dmabits.h"

/* DEFINE */

#define DEPTH 3
#define HEIGHT 256 /* sistema PAL */
#define WIDTH 32Ø

/* dichiarazioni per strutture allocate staticamente */

struct BitMap BitMap1, BitMap2;
struct RasInfo RasInfo1, RasInfo2;
struct RastPort RastPort1, RastPort2;
struct ViewPort ViewPort1;
struct View MyView;

InitBitMap (&BitMap1, DEPTH, WIDTH, HEIGHT);
InitBitMap (&BitMap2, DEPTH, WIDTH, HEIGHT);

for (i=Ø;iDEPTH;i++){
      BitMap1.Planes[i] = (PLANEPTR) AllocRaster (WIDTH, HEIGHT);
      BitMap2.Planes[i] = (PLANEPTR) AllocRaster (WIDTH, HEIGHT);
};

InitRastPort (&RastPort1); InitRastPort (&RastPort2);
RastPort1.BitMap = &BitMap1; RastPort2.BitMap = &BitMap2;

RasInfo1.BitMap = &BitMap1;
RasInfo1.RxOffset = XOffset1; RasInfo1.RyOffset = YOffset1;
RasInfo1.Next = &RasInfo2;

RasInfo2.BitMap = &BitMap2;
RasInfo2.RxOffset = XOffset2; RasInfo2.RyOffset = YOffset2
RasInfo2.Next = NULL;

InitVPort (&ViewPort1);

ViewPort1.Modes = DUALPF;
ViewPort1.DWidth = WIDTH; ViewPort1.DHeight = HEIGHT;
ViewPort1.DxOffset = Ø.Ø; ViewPort1.DyOffset = yourposition1;
```

```
ViewPort1.RasInfo = &RasInfo1;
ViewPort1.Next = NULL;

InitView (&MyView);
MyView.ViewPort = &ViewPort1;

MakeVPort (&MyView, &ViewPort1);

MrgCop (&MyView);
LoadView (&MyView);
```

Il modo double-buffer

La tecnica del double-buffer consiste nell'usare due sezioni di RAM (considerate due buffer) di identiche dimensioni. Vengono così mantenute due versioni delle informazioni della bitmap. Ogni bitmap è in relazione con l'altra. In pratica, per la maggior parte del tempo, l'una costituisce la copia dell'altra. Questa situazione distingue il modo double-buffer dal modo dual-playfield.

Il modo double-buffer consente al programma di disegnare in una delle due bitmap mentre viene presentata l'altra. In tal modo è possibile produrre presentazioni grafiche in evoluzione senza scatti di schermo. Ciò risulta particolarmente utile quando si vuole produrre un'animazione. Tale metodo è un passo avanti rispetto al ridisegnare la bitmap dopo che il pennello elettronico del tubo a raggi catodici ha superato un determinato punto dello schermo video. Questo modo di procedere permette anche di evitare lo sfarfallio di schermo (si veda la spiegazione relativa alla funzione QBSBlit).

Quando si prende in considerazione l'ipotesi d'impiegare la tecnica del double-buffer, si deve ricordare che tutte le informazioni grafiche devono rientrare nei limiti della memoria chip. Per esempio, per una bitmap di 320 x 256 pixel con cinque bitplane, ogni bitplane è destinato a occupare 10240 byte; una bitmap completa richiede quindi 51200 byte. Ne consegue che una presentazione video basata sulla tecnica del double-buffer ha bisogno di 102400 byte, cioè di ben 100K. L'impegno di memoria non sarebbe ancora eccessivo, ma se si raddoppia la risoluzione in entrambe le direzioni (usando uno schermo di 640 x 512 pixel), le informazioni per la bitmap richiedono 409600 byte di chip RAM. Come si può osservare, una tale situazione impone gravi limiti.

La chiave della programmazione con il modo grafico double-buffer consiste nella corretta sequenza di esecuzione delle istruzioni. È necessario alternare il puntatore alla bitmap, così che il programma mostri una bitmap mentre disegna nell'altra; poi i ruoli si invertono.

Ecco una breve esposizione della sequenza di operazioni necessarie a definire una schermata con la tecnica double-buffer:

1. Si allocano due strutture BitMap. Ognuna è destinata a descrivere una bitmap diversa nella presentazione. Si devono conservare gli indirizzi restituiti dalle funzioni di allocazione della memoria, che verranno in seguito impiegati nella struttura RasInfo (e nella struttura RastPort) per puntare alternativamente alle due strutture BitMap.

2. Si impiega la funzione AllocRaster per allocare la memoria richiesta per le due bitmap separate. Si possono prevedere fino a sei bitplane per bitmap. Le bitmap devono avere larghezza e altezza identiche.

3. Si alloca la memoria necessaria per una struttura RasInfo. Ne basta solo una: quello che va continuamente cambiato è il puntatore alla struttura BitMap (contenuto nella struttura RasInfo).

4. Si alloca la memoria necessaria per una struttura RastPort. Ne basta solo una, in quanto si può modificare il puntatore alla struttura BitMap, contenuto nella struttura RastPort, in modo che punti alternativamente a ciascuna delle due bitmap.

5. Si alloca la memoria necessaria per una struttura ViewPort.

6. Si alloca la memoria necessaria per una struttura View. Ne basta solo una, perché la presentazione delle informazioni è destinata ad avvenire su una bitmap per volta.

7. Si impiega la funzione InitBitMap per inizializzare le due strutture BitMap. Gli indirizzi restituiti dalle funzioni di allocazione della memoria vengono ora usati per puntare a ciascuna struttura BitMap.

8. Si adoperano quattro istruzioni di assegnazione dei parametri di struttura per inizializzare la struttura RasInfo (già allocata). Il puntatore BitMap della struttura RasInfo va impostato per puntare alla struttura BitMap corrispondente alla prima bitmap nella quale s'intende disegnare.

9. Va chiamata la funzione InitRastPort per inizializzare la struttura RastPort (già allocata). Il puntatore BitMap nella struttura RastPort dev'essere impostato per puntare alla struttura BitMap che corrisponde alla prima bitmap nella quale s'intende disegnare.

10. Si chiama la funzione InitVPort per inizializzare la struttura ViewPort (già allocata). Si ricordi che ogni struttura ViewPort possiede un puntatore a una struttura RasInfo. La struttura RasInfo, peraltro, possiede un puntatore a una struttura BitMap. Perciò, cambiando il puntatore alla struttura BitMap presente nella struttura RasInfo, la struttura ViewPort risulta associata alla giusta bitmap. Bisogna impostare anche gli altri parametri della struttura ViewPort.

11. Si chiama la funzione InitView per inizializzare la struttura View (già allocata).

12. Si chiamano le funzioni di disegno della libreria Graphics per disegnare nella prima bitmap. Nel frattempo l'altra è inattiva.

13. Si chiama la funzione MakeVPort per la viewport. Ciò produce le liste di istruzioni Copper per la view basata sulla bitmap sulla quale abbiamo fatto il disegno.

14. Si chiama la funzione MrgCop per produrre la fusione di tutte le liste di istruzioni Copper di viewport, cioè delle liste relative alla prima bitmap. Quest'operazione informa il sistema di creare la lista Copper che definisce la presentazione della prima bitmap. Il sistema crea allora i puntatori per le liste di view LOFCprList e SHFCprList (long frame Copper list e short frame Copper list: la lista di istruzioni per il quadro video primario e per quello secondario, nel caso di schermo in interlace) per la prima delle due presentazioni alternate. Il sistema alloca automaticamente la memoria per tali liste. La SHFCprList viene usata soltanto per la presentazione video in interlace, mentre la LOFCprList viene adoperata con o senza interlace. I puntatori a queste due liste Copper vengono impostati dal sistema nella struttura View. Come risultato, il flusso di istruzioni Copper si riferisce alla prima struttura BitMap; si chiama quindi la funzione LoadView per caricare le istruzioni Copper della view nell'hardware video, al fine di veder apparire sullo schermo la prima bitmap.

15. Si salvano, in variabili di riserva, gli indirizzi delle liste LOFCprList e SHFCprList, e si azzerano i corrispondenti puntatori della struttura View.

16. A questo punto si è pronti per passare alla seconda bitmap. I puntatori alla struttura BitMap presenti nelle strutture RasInfo e RastPort vanno cambiati, in modo da puntare alla seconda struttura BitMap. Si usano quindi le funzioni di disegno della libreria Graphics per disegnare nella seconda bitmap.

17. Si chiama la funzione MakeVPort per le varie viewport della view. Ciò produce le liste di istruzioni Copper per la seconda bitmap.

18. Si chiama la funzione MrgCop. Ciò produce l'unione di tutte le liste di istruzioni Copper di viewport per definire la presentazione della seconda bitmap. Quando si chiama la funzione MrgCop, il sistema alloca automaticamente un'altra area di memoria per mantenere le nuove liste di istruzioni Copper. Ancora una volta, gli indirizzi di queste liste (le liste LOFCprList e SHFCprList) vengono inseriti nella struttura View. Si chiama LoadView per caricare nell'hardware le istruzioni Copper per la view, e si vede apparire sullo schermo la seconda bitmap.

Come risultato, in RAM coesistono due flussi di istruzioni Copper. Gli indirizzi di uno dei due vengono temporaneamente salvati mentre si creano gli indirizzi del secondo insieme. Si ricordi che LoadView agisce con la struttura View. Cambiando i puntatori delle liste Copper nella struttura View, ciascuna delle due bitmap può essere caricata e visualizzata.

Nel frattempo, il task grafico può essere impegnato a disegnare nella bitmap non visibile. Ci si accerti d'impiegare il corretto puntatore alla struttura BitMap nelle strutture RasInfo e RastPort, in ciascuna fase della definizione dei disegni. E si ricordi che, al termine, si dovrà effettuare una chiamata alla funzione FreeCopList per entrambe le liste Copper intermedie.

Un modello di programmazione per il modo double-buffer

Il listato che segue riporta le istruzioni di programma necessarie per definire una presentazione double-buffer. Questa sezione di programma può essere impiegata come base per qualsiasi programma che faccia uso della tecnica del double-buffer.

L'esempio illustra una viewport e una view. XOffset e YOffset sono gli offset della bitmap di viewport, e devono essere scelti in base alla grandezza della bitmap e alla posizione della viewport all'interno della bitmap. Per fissare la posizione della viewport nella view che costituisce lo schermo video dell'Amiga, è anche necessario specificare il valore di yourposition1. La viewport è profonda cinque bitplane, in modo da poter offrire contemporaneamente 32 colori.

Le direttive di compilazione #define descrivono le caratteristiche di un video in bassa risoluzione e senza interlace. Si noti che viene definita una sola view. Si noti inoltre che nelle strutture ViewPort e View non vengono impostati modi grafici particolari; il sistema non viene informato del fatto che si sta impiegando la tecnica double-buffer.

Questo esempio mostra come fare per:

1. Gestire i puntatori di struttura CprList per l'hardware (puntatori presenti nella struttura View).

2. Creare una struttura View.

3. Salvare in variabili temporanee i puntatori della struttura CprList.

4. Creare un'altra struttura View.

5. Salvare ancora una volta i puntatori della struttura CprList.

Si può poi procedere alternando le due bitmap e le liste Copper a esse associate; ciò si ottiene impiegando i due citati insiemi di puntatori. Si osservi inoltre che le strutture sono assegnate staticamente; lo schema di programmazione, comunque, vale anche nel caso in cui le strutture vengano create dinamicamente.

```
/* INCLUDE */

#include "exec/types.h"
#include "exec/memory.h"
#include "exec/exec.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "graphics/view.h"
#include "graphics/copper.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "graphics/text.h"
#include "hardware/blit.h"
#include "hardware/custom.h"
#include "hardware/dmabits.h"

/* DEFINE */

#define DEPTH 5
#define HEIGHT 256
#define WIDTH 320

/* dichiarazioni per strutture assegnate staticamente */

struct BitMap BitMap1, BitMap2;
struct RasInfo RasInfo;
struct RastPort RastPort;
struct ViewPort ViewPort1;
struct View MyView;

InitBitMap (&BitMap1, DEPTH, WIDTH, HEIGHT);
InitBitMap (&BitMap2, DEPTH, WIDTH, HEIGHT);

for (i=0;iDEPTH;i++){
      BitMap1.Planes[i] = (PLANEPTR) AllocRaster (WIDTH, HEIGHT);
      BitMap2.Planes[i] = (PLANEPTR) AllocRaster (WIDTH, HEIGHT);
};

RasInfo.BitMap = &BitMap1;
RasInfo.RxOffset = XOffset; RasInfo.RyOffset = YOffset
RasInfo.Next = NULL;

InitRastPort (&RastPort);

RastPort.BitMap = &BitMap1;
```

```
InitVPort (&ViewPort1);

ViewPort1.DWidth = WIDTH; ViewPort1.DHeight = HEIGHT;
ViewPort1.DxOffset = Ø.Ø; ViewPort1.DyOffset = yourposition1;
ViewPort1.RasInfo = &RasInfo;
ViewPort1.Next = NULL;

InitView (&MyView);
MyView.ViewPort = &ViewPort1;

/* Ora si disegna nella bitmap 1 */

MakeVPort (&MyView, &ViewPort1);

MrgCop (&MyView);

/* Ora si presenta la bitmap 1 */

LoadView (&MyView);

/* Ora ci si prepara a disegnare nella bitmap 2 */

BitMap1LOFCprList = MyView.LOFCprList;
BitMap1SHFCprList = MyView.SHFCprList; /* Non è necessario
   se non in modo LACE */

MyView.LOFCprList = Ø.Ø;
MyView.SHFCprList = Ø.Ø; /* Non è necessario se non in modo LACE */

RasInfo.BitMap = &BitMap2;
RastPort.BitMap = &BitMap2;

/* Ora si disegna nella bitmap 2 */

MakeVPort (&MyView, &ViewPort1);

MrgCop (&MyView);

/* Ora si presenta la bitmap 2 */

LoadView (&MyView);

/* Ora ci si prepara a disegnare nuovamente nella bitmap 1 */
```

```
BitMap2LOFCprList = MyView.LOFCprList;
BitMap2SHFCprList = MyView.SHFCprList; /* Non è necessario
   se non in modo LACE */

MyView.LOFCprList = BitMap1LOFCprList;
MyView.SHFCprList = BitMap1SHFCprList; /* Non è necessario
   se non in modo LACE */

/* Si continua a disegnare, scambiando alternativamente i puntatori CprList
   nella struttura View */
```

Il modo Hold And Modify

Il maggior vantaggio offerto dal modo grafico Hold And Modify rispetto agli altri è la possibilità di mostrare contemporaneamente sullo schermo fino a 4096 colori, contro capacità massime degli altri modi che arrivano a 64 colori in bassa risoluzione senza interlace, o addirittura solo 16 colori in alta risoluzione con interlace.

Per adoperare il modo Hold And Modify, si deve agire come segue:

1. Si impiegano sei bitplane per definire la bitmap. I bitplane 1, 2, 3 e 4 definiscono i 16 colori per i pixel, mentre i bitplane 5 e 6 sono impiegati per informare il sistema come deve mantenere (hold) o modificare (modify) quei colori.

2. Si inizializza una struttura ViewPort. Nel parametro Modes va impostato il flag HAM. Quando si disegna nella bitmap dalla quale ha origine ogni viewport, si possono scegliere diversi metodi di disegno. Se s'impiegano i numeri di colore compresi tra 0 e 15 (in relazione ai bitplane da 1 a 4), il pixel considerato risulterà disegnato con il colore specificato nel registro di colore corrispondente (si ricordi che i numeri di colore vengono impostati attraverso le funzioni SetAPen, SetBPen e con la macro SetOPen).

   Se si disegna con un valore di colore compreso tra 16 e 63 (in relazione ai bitplane da 1 a 6), il colore effettivamente impiegato per il pixel dipende dal colore del pixel immediatamente alla sua sinistra. Il colore del pixel risulterà basato su due dei tre colori primari (RGB, rosso verde blu) del pixel precedente. Per due di questi colori primari verrà mantenuta costante l'intensità (si ricorda che l'intensità può avere 16 diversi valori), mentre cambierà quella del terzo colore primario. Per sapere come comportarsi con il terzo colore, l'hardware video utilizza i bit dei pixel appartenenti ai quattro bitplane inferiori. Per esempio, è possibile mantenere l'intensità dei colori blu e rosso sulla base dei precedenti valori di pixel. L'intensità del verde dipenderà dai bit dei

bitplane 1/4. In tal caso, il rosso e il blu restano costanti, mentre il verde cambia. Ciò spiega l'origine del nome del modo grafico considerato.

I bitplane 5 e 6 vengono impiegati per determinare quali dei colori primari devono essere mantenuti, conservandoli dal precedente pixel. Per i pixel la cui combinazione bit 6-5 risulta 00 verrà impiegato il normale metodo di selezione dei colori, e quindi il colore definito dalla combinazione bit dei bitplane 1/4 (il pixel di sinistra non viene usato per modificare i colori di quello considerato).

Se la combinazione bit 6-5 per un dato pixel è 01, il colore del pixel di sinistra viene duplicato per essere poi modificato. In tal caso la combinazione bit dei bitplane 1/4 viene impiegata per determinare l'intensità del blu nella posizione pixel considerata. Restano quindi costanti il rosso e il verde, mentre si modifica il blu. Se la combinazione bit 6-5 per un pixel è 10, restano costanti il verde e il blu, mentre si modifica il rosso. Se la combinazione bit 6-5 per un pixel è 11, restano costanti il rosso e il blu, mentre si modifica il verde. In tutti i casi l'intensità del colore modificato si adegua alla definizione di colore descritta dai bit dei bitplane 1-4.

Da ciò si osservi che la decisione d'impiegare il modo Hold And Modify va presa fin dall'inizio della progettazione di un programma grafico.

Un modello di programmazione per il modo Hold And Modify

Il listato che segue mostra le istruzioni di programma necessarie per definire una presentazione in modo Hold And Modify.

Questa sezione di programma può essere impiegata come base per qualsiasi programma che faccia uso del modo Hold And Modify.

L'esempio illustra una viewport e una view. XOffset e YOffset sono gli offset per la bitmap di viewport che devono essere scelti in base alla grandezza della bitmap e alla posizione della viewport nella bitmap. Per fissare la posizione della viewport nella view che costituisce lo schermo video dell'Amiga, è anche necessario specificare il valore di yourposition1. La viewport è profonda sei bitplane, in modo che sia possibile attivare il modo Hold And Modify e ottenere i 4096 colori.

Le istruzioni #define descrivono un video in bassa risoluzione senza interlace. Si noti che viene definita una sola view. Si noti inoltre che nella struttura ViewPort il parametro Modes è impostato a HAM. Quindi, nel momento in cui si disegna nella bitmap, i bit in essa presenti risultano interpretati in modalità Hold And Modify per quanto riguarda la definizione dei colori. Si osservi inoltre che le strutture sono assegnate staticamente. Il modello di programmazione, comunque, rimarrebbe valido anche se venissero assegnate dinamicamente.

```
/* INCLUDE */

#include "exec/types.h"
#include "exec/memory.h"
#include "exec/exec.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "graphics/view.h"
#include "graphics/copper.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "graphics/text.h"
#include "hardware/blit.h"
#include "hardware/custom.h"
#include "hardware/dmabits.h"

/* DEFINE */

#define DEPTH 6
#define HEIGHT 256
#define WIDTH 32Ø

/* dichiarazioni per strutture assegnate staticamente */

struct BitMap BitMap;
struct RasInfo RasInfo;
struct RastPort RastPort;
struct ViewPort ViewPort1;
struct View MyView;

InitBitMap (&BitMap, DEPTH, WIDTH, HEIGHT);

for (i=Ø;iDEPTH;i++){
   BitMap.Planes[i] = (PLANEPTR) AllocRaster (WIDTH, HEIGHT);
};

InitRastPort (&RastPort);

RastPort.BitMap = &BitMap;

RasInfo.BitMap = &BitMap;
RasInfo.RxOffset = XOffset; RasInfo.RyOffset = YOffset
RasInfo.Next = NULL;

InitVPort (&ViewPort1);
```

```
ViewPort1.Modes = HAM;
ViewPort1.DWidth = WIDTH; ViewPort1.DHeight = HEIGHT;
ViewPort1.DxOffset = Ø.Ø; ViewPort1.DyOffset = yourposition1;
ViewPort1.RasInfo = &RasInfo;
ViewPort1.Next = NULL;

InitView (&MyView);
MyView.ViewPort = &ViewPort1;

MakeVPort (&MyView, &ViewPort1);

MrgCop (&MyView);
LoadView (&MyView);
```

Il modo Extra Half-Brite

Il modo Extra Half-Brite consente di definire viewport che mostrano 32 colori addizionali dotati di una luminosità dimezzata rispetto ai colori definiti attraverso i registri di colore 0/31.

Il modo Extra Half-Brite richiede sei bitplane e il modo grafico single-playfield; questa modalità non può essere utilizzata insieme al modo Hold And Modify. Per usare il modo Extra Half-Brite, si imposta a EXTRA_HALFBRITE il flag del parametro Modes della struttura ViewPort. Ogni pixel che abbia il bit del bitplane 6 impostato a 1, sarà presentato con colori di luminosità dimezzata rispetto alle normali condizioni. I colori sono definiti dai primi cinque bitplane.

Il listato presentato per il modo Hold And Modify, può essere facilmente modificato per ottenere un modello per il modo Extra Half-Brite. Basta cambiare in EXTRA_HALFBRITE la costante da memorizzare nel parametro Modes della struttura ViewPort, per fare in modo che le informazioni di bitmap nei sei bitplane vengano interpretate secondo la filosofia del modo Extra Half-Brite.

Indici

S

INDICE ALFABETICO DELLE FUNZIONI E DELLE MACRO

* Queste funzioni restituiscono un valore
** Macro-istruzioni

INDICE DELLE STRUTTURE

Questo volume è stato stampato nel mese di maggio 1990
presso gli stabilimenti della Grafica Editoriale Lodigiana S.r.l.
Stampato in Italia – Printed in Italy

PROGRAMMARE L'AMIGA

VOLUME I

Programmare l'Amiga Volume I è un completo manuale di programmazione, che espone in dettaglio le funzioni e le strutture che l'Amiga mette a disposizione per la grafica, l'animazione e la gestione multitasking del sistema. Aggiornato, conciso e strutturato in modo da venire incontro alle esigenze dei programmatori, costituisce una guida ideale e completa per la realizzazione di applicazioni che siano in grado di sfruttare al meglio le straordinarie capacità dell'Amiga. Il libro copre più di 300 funzioni di sistema, ordinate alfabeticamente e suddivise in sette argomenti principali:

- Le funzioni della libreria Exec
- Le funzioni di disegno della libreria Graphics
- Le funzioni di animazione della libreria Graphics
- Le funzioni di gestione del testo della libreria Graphics
- Le funzioni della libreria Layers
- Le funzioni della libreria Intuition
- Le funzioni della libreria Icon

Fornisce inoltre un'articolata serie d'informazioni su:

- Bitmap
- Finestre
- Gadget
- Gestione della memoria
- Gestione dello schermo
- Icone
- Layer
- Multitasking
- Requester
- Schermi
- ...e moltissimi altri argomenti

Il testo è anche corredato di numerose illustrazioni e diagrammi per facilitare la comprensione degli argomenti trattati e guidare il lettore nella consultazione.

Si rivolge ai programmatori in linguaggio C e Assembly, fornendo loro strumenti di base indispensabili (la descrizione di tutte le principali strutture, delle principali funzioni di libreria e dei loro argomenti) e illustrando tanto le tecniche di programmazione ad alto livello quanto l'interazione diretta con l'hardware.

«*Programmare l'Amiga Vol. I* è un imponente strumento di lavoro, strutturato con grande intelligenza: indispensabile per la vostra biblioteca tecnica. Nella trattazione, arricchita da un prezioso compendio di riferimenti incrociati sulle diverse funzioni analizzate, non mancano consigli ed esempi pratici, così rari nei consueti manuali di programmazione». (Dalla rivista *Byte*)

L'AUTORE

Eugene P. Mortimore è presidente della Micro Systems Analysis, Inc., un'azienda di consulenza informatica specializzata nei computer Amiga e IBM. È autore di numerosi libri sui microcomputer e vive a Bethel Park, in Pennsylvania (USA).

ISBN 88-7803-004-X

Lire 80.000

9 788878 030046

Eugene P.
Mortimore

Programmare l'Amiga Volume I

iHT
GRUPPO EDITORIALE