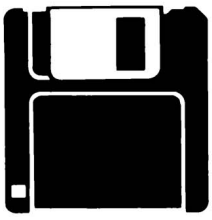
AMIGA
tecniche di programmazione

Robert A. Peck

• EXEC • LIBRARY • DEVICE • INTUITION • GRAFICA
• ANIMAZIONE • SUONO • MULTITASKING
• PROGETTARE E COSTRUIRE UN PROGRAMMA DI PAINT

CONTIENE DISCO 3½"



GRUPPO EDITORIALE
JACKSON

AMIGA
tecniche di programmazione

Robert A. Peck

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Titolo originale:
Programmer's Guide to the AMIGA

Traduzione autorizzata dall'edizione in lingua inglese

TRADUZIONE: Alessandro Sicoli
REVISIONE TECNICA: Daniele Cassanelli – Sergio Ruocco
IMPAGINAZIONE ELETTRONICA: Marco Santini
REDATTORE DI COLLANA: Luigi Cerabolini
COPERTINA: Emiliano Bernasconi

RINGRAZIAMENTI

Vorrei ringraziare i dirigenti e lo tutto staff della SYBEX per gli sforzi compiuti nel produrre questo libro. Vorrei in particolare ringraziare il Dr. Rudolph Langer e il Dr Karl Ray, che mi hanno dato la possibilità di dare ancora più informazioni ai programmatori di Amiga. Il mio curatore Valerie Robbins che è stata provvidenziale come aiuto e come guida. E' per merito suo che finalmente è stato realizzato questo progetto.

Ringrazio anche tutto il gruppo della Hewlett-Packard per il suo aiuto e incoraggiamento durante i mesi scorsi, e per avermi offerto l'occasione di lavorare a un progetto così interessante e stimolante.

E infine, saluto tutti i Busy Guys che hanno creato Amiga e tutti i programmatori che hanno lavorato sul sistema iniziale. I vostri sforzi hanno creato una macchina meravigliosa per il divertimento di tutti noi.

Alla mia favolosa moglie, Andrea.
Non avrei potuto fare nulla di tutto ciò senza di te.

INDICE

PREFAZIONE

Nello scrivere il Manuale del ROM Kernel di AMIGA dividemmo il sistema in varie aree funzionali. Una sezione riguardava il Sistema Operativo, un'altra riguardava la grafica, un'altra il suono, l'animazione, la matematica, le periferiche, e così via. Questa partizione in aree funzionali portò ad avere una certa visione del sistema. Nella parte introduttiva cercammo di rendere chiaro come le varie parti fossero connesse. Questo tipo di approccio iniziava di solito con una parte descrittiva dei dati usati dalle singole funzioni, e terminava con un esempio operativo che il lettore poteva introdurre e provare.

Quando iniziai questo libro capii che avrei dovuto usare un altro punto di vista nei confronti di Amiga. La mia esperienza di utilizzatore di Network mi ha convinto che un esempio è il modo migliore per arrivare alla meta. Se un esempio esaurientemente commentato mostra come una particolare struttura software possa essere usata, esso costituisce il massimo chiarimento. Così è nato questo libro, con l'intenzione di utilizzare il maggior numero di programmi brevi, ma esplicativi.

Ogniqualvolta è stato possibile l'esempio è stato listato completo di per sé. Comunque, ogni tanto, per semplificare un po' le cose, alcune parti di esempi importanti o di certe routine vengono spiegate specificatamente. A volte si rende quindi necessario compilare il singolo segmento e poi linkarlo con la routine iniziale per completare l'esempio.

Il libro inizia spiegando come far fare ad Amiga cose comuni a tutti i computer. Più avanti, procedendo su questo terreno, sono state incluse anche delle funzioni specifiche di Amiga, come il Multitasking e altre. Man mano che si avanza in questo libro si trovano programmi di esempio che possono validamente essere di supplemento all'Amiga ROM Kernel Manual, all'AmigaDos Developers Manual, e all'Intuition Manual. Tutti questi volumi trattano parti del software di sistema di Amiga e questo è proprio ciò che cerca di spiegare questo libro. E procedendo, poiché si troverà agevole compilare i programmi in Amiga C, si incontreranno alcune delle funzioni dello stesso Amiga C.

Il compilatore C spesso si presenta con la forma di una libreria di interfaccia che provvede, per esempio, alle routine di input-output dei file e dei caratteri. Questo libro usa queste interfacce specifiche il meno possibile. Un lettore che utilizza un compilatore diverso da quello dell'Amiga C avrà qualche difficoltà ad

adattare gli esempi al proprio compilatore. In fondo al libro, per rendere tale adattamento il più facile possibile, nell'Appendice B è inclusa una descrizione del modo con cui l'Amiga C si interfaccia con il Software di sistema.

I progettisti del software di sistema di Amiga provvedettero a creare varie routine per rendere meno gravoso possibile il compito del programmatore. Per esempio la maggior parte dell'hardware di Amiga è controllato in modo efficiente da varie routine di sistema. Nella maggioranza dei casi non è necessario memorizzare direttamente i dati nei registri perché avvengano certe cose. Ne è necessario consultare le profondità del manuale hardware per scoprire come utilizzare una certa funzione. Usando il software di sistema si può dire ad Amiga di svolgere determinati compiti complessi in un determinato modo e sarà il software di sistema stesso a manipolare l'hardware e i bit nella memoria per ottenere ciò che si desidera.

Per non complicare le cose, ovunque è stato possibile, ho cercato di concentrare l'attenzione sull'effetto finale piuttosto che soffermarmi sui singoli registri di controllo o sulla struttura dei dati, a meno che questi non avessero una correlazione immediata con ciò che veniva fatto. Spero che questo metodo di approccio vi sarà utile.

Per evitare che tutto il discorso non si riducesse a una spiegazione dei singoli dettagli, questo libro punta a far sì che il programmatore aumenti velocemente le proprie possibilità di esecuzione di programmi su Amiga. Invece di cercare di descrive le singole opzioni di ogni routine e i vari insiemi di dati, ho mantenuto il discorso più semplice possibile cercando di presentare ciò che più probabilmente un programmatore dovrà affrontare usando Amiga.

Tutti gli esempi di questo libro sono stati compilati con il compilatore dell'Amiga C versione 1.1 e usando il Kickstart 1.1 o 1.2 Notate che ogniqualvolta negli esempi si usa: include "myintuition.h" si potrebbe usare in alternativa: include "exec-types.h" e include "intuition/intuition.h". Se non si desidera digitare personalmente i programmi di esempio si possono utilizzare i programmi contenuti nel dischetto allegato.

Amiga possiede delle strutture Hardware specializzate per aiutare la CPU a svolgere i compiti primari. Alcune di queste strutture governano il trasferimento da e al disk drive, altre controllano le porte giochi e altre ancora leggono la tastiera e prmettono la visualizzazione sul monitor.

Sarebbe possibile agire direttamente sull'hardware per comandare Amiga ma risulta molto più facile utilizzare delle routine software (chiamate anche funzioni). Queste routine (che sono oltre 300) consentono facilmente di controllare tutte le varie parti della macchina.

Capitolo 1

Sguardo generale

Gerarchia del software e dell'hardware di Amiga

La figura 1.1 mostra un diagramma a blocchi del software di sistema di Amiga, conosciuto come Kickstart. Come si può vedere i vari livelli del software di sistema sono costruiti l'uno sull'altro. In cima vi è un programma applicativo, che è poi ciò che di fatto interagisce con l'utente. All'ultimo livello (livello primario) troviamo l'hardware della macchina. Un programmatore trova dei punti di inserimento nel software di sistema per ottenere ciò che vuole. Quali di questi punti egli utilizzi dipende da ciò che si prefigge e dalla complessità del suo progetto.

Il livello primario

L'interfacciamento con l'hardware del sistema avviene tramite alcune routine specifiche.

L'insieme delle routine che controllano l'uso del 68000 è chiamato Exec. L'Exec ripartisce il 68000 tra programmi differenti (task) che possono trovarsi nella memoria contemporaneamente; così Amiga è in grado di operare in multitaskig. L'Exec gestisce anche l'allocazione della memoria per i singoli programmi e controlla le richieste di Interrupt che sono generate dall'hardware o dalle varie applicazioni software. L'Exec inoltre mantiene le liste dei programmi che sono in attesa di determinati eventi, le liste delle aree di memoria disponibile, le liste dei messaggi relativi a ogni applicazione e le liste degli Input, come i movimenti del mouse, impulsi di timer, i tasti che vengono premuti e così via.

Unità software dette devices controllano l'accesso al drive, alla tastiera, alle porte giochi (mouse compreso) e alle porte seriale e parallela. Il software grafico di sistema controlla direttamente l'hardware corrispondente e contiene routine per il tracciamento e il controllo dei disegni, per la visualizzazione delle varie aree grafiche, per la selezione dei colori e dei pattern per la generazione delle varie risoluzioni e per la manipolazione degli oggetti grafici.

Il secondo livello

Nel livello superiore troviamo i device di input e quelle funzioni dette di Layer. I device di input girano come dei programmi indipendenti nel sistema e provvedono a raccogliere tutte le informazioni in input dalla tastiera e dalle porte giochi (normalmente connesse a un mouse), e le convoglia in unico canale input.
Come si può vedere questo canale porta le informazioni all'Intuition o a un device di Console.

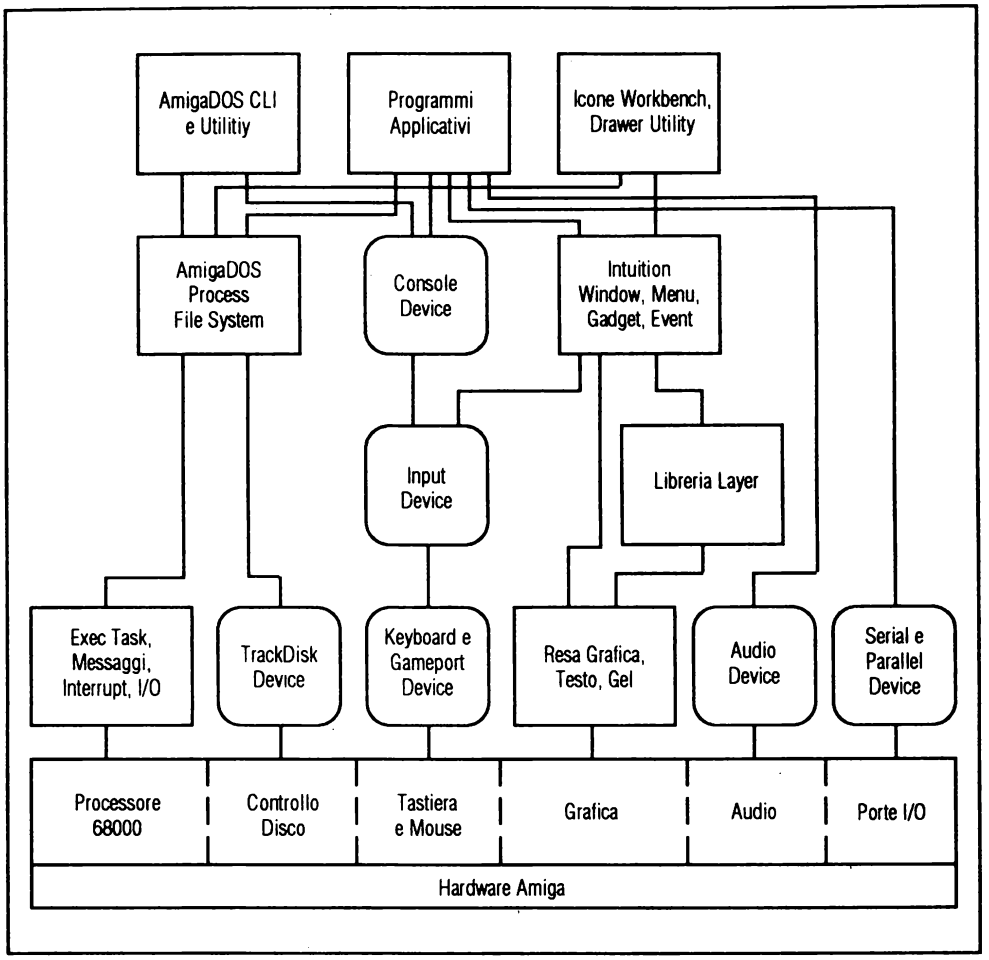


Fig. 1.1 La gerarchia del sistema software Amiga

Le funzioni di Layer sono costruite in cima al sistema grafico e provvedono a delle routine che dividono le normali aree grafiche in aree multiple, riunendo quelle aree che sono alla base della costruzione delle window. Tra le funzioni di Layer si trovano routine per la creazione, la manipolazione, lo scambio fronte-retro delle aree.

Il terzo livello

Al terzo livello troviamo l'AmigaDos, l'Intuition e i device di Console. L'AmigaDos è un sistema operativo con capacità di operare in modo multifunzionale. Esso opera con l'Exec al fine di ripartire correttamente tutte le risorse del sistema, inclusa la ripartizione della CPU tra i programmi. L AmigaDos provvede al trattamento dei file e contiene varie utility per manipolare i file stessi e per lanciare nuovi programmi.

L'Intuition è l'interfaccia multiscreen e multiwindow di Amiga. Usando le sue Routine è possibile creare un visualizzatore nel quale coesistono vari schermi con dati diversi. Ogni schermo può avere risoluzioni che vanno da 320 a 640 punti verticali e da 200 a 400 (da 256 a 512 per la versione PAL) punti orizzontali. Ogni schermo con formato 640-punti può avere 16 colori diversi visualizzati contemporaneamente (scelti da una palette di 4096) mentre uno schermo con formato 320-punti può visualizzare contemporaneamente 32 colori (scelti da una palette di 4096).

Intuition è di fatto una libreria di funzioni. (Vedi il par. Funzioni di libreria). Si possono richiamare queste funzioni per creare una interfaccia grafica per l'utente, spezzando gli schermi in sottounità dette window, creando menu dai quali l'utente può selezionare le opzioni, e creando dei requester (una specie di microwindow che richiede un input da parte dell'utente).

I device di input convogliano le informazioni di input nella Intuition. L'Intuition a sua volta traduce queste in eventi di tipo grafico ai quali si può reagire come si vuole con facilità. Inoltre l'Intuition filtra le varie informazioni in modo che nel singolo programma agiscano solo quelle che servono al programma stesso.

Il device di Console è un opzione applicabile alle window che le fa operare come se fossero dei terminali, simili a quelli usati all'inizio dell'era dei computer. Il device di Console si comporta proprio come un terminale hardware leggendo dalla tastiera le informazioni che sono inviate a quella particolare window e generando dei messaggi in risposta.

Intuition agisce come un vigile stradale tenendo attiva una sola window per volta. Così ciò che è digitato viene inviato a una window (e al suo device di Console se esiste) solo quando questa è attivata da una pressione del tasto di selezione del mouse.

Il livello massimo

In cima alla figura 1.1 si notano tre voci distinte: un programma applicativo, il Workbench e il CLI (Command Line Interface). Ognuna di queste ha una propria interfaccia per dialogare con l'operatore.
Il CLI si presenta con ciò che può essere chiamata la tradizionale interfaccia di un computer, qualcosa che assomiglia a un terminale. Il CLI è di fatto una applicazione che traduce ciò che viene inserito da tastiera in forma di comandi che si suppone il computer sappia eseguire.

Il Workbench sostituisce con un interfaccia a finestre e icone le normali operazioni di lavoro tradizionalmente svolte digitando i comandi sulla tastiera. Essa esegue molti dei comandi che possono essere eseguiti dal CLI. Ciò che perde in versatilità, lo guadagna senz'altro in facilità d'uso.
Un programma applicativo presenta il tipo di interfaccia desiderata dal programmatore, basata probabilmente su molte delle possibilità messe a disposizione dal Software del sistema.

Cosa è necessario conoscere per programmare Amiga

Ciò che è necessario conoscere riguardo la programmazione di Amiga dipendono largamente dalla complessità del programma che si cerca di realizzare. Per esempio, si potrebbe voler progettare un programma che trae vantaggio delle fa-

volose capacità grafiche e musicali di Amiga. In questo caso è necessario sapere come organizzare le strutture dati per la grafica e il suono, come aprire le relative funzioni di libreria, e quali funzioni usare per ottenere determinate cose.

Oppure si vorrebbe sviluppare un programma che utilizza le possibilità di multitaskig, per esempio stampando un documento mentre se ne sta preparando un altro, mentre contemporaneamente si sta leggendo una Mail elettronica da una banca dati lontana. In questo caso è necessario conoscere tutto ciò che riguarda il multitaskig e sapere come ordinare al Dos di aprire un'applicazione separata per i nostri scopi.

In generale si dovrebbero conoscere le operazioni riguardanti il CLI e il sistema Exec in aggiunta alle caratteristiche della parte specifica che si desidera usare per le applicazioni nei propri programmi, poiché è necessaria una certa padronanza dell'insieme della macchina prima di poter usare direttamente la grafica o qualsiasi altra funzione specifica di Amiga.

Funzioni di libreria

Nella descrizione dei vari livelli di Amiga si è usata la parola libreria. Una libreria è una raccolta di funzioni speciali che sono correlate le une alle altre. I progettisti di Amiga raccolsero diverse funzioni in libreria per far si che fosse garantita il più possibile la compatibilità tra i programmi e la varie versioni del Kickstart. Per permettere questa compatibilità, essi dovevano essere sicuri che i programmi potessero sempre trovare con sicurezza le funzioni di sistema, a prescindere dal cambiamento del sistema stesso da una versione all'altra.

Per favorire questo, i progettisti generarono una struttura di dati nella libreria che contiene, tra le altre cose, una tabella con le istruzioni di Jump e gli indirizzi delle funzioni, in questo modo

```
        JUMP Function N
        .... ....
        JUMP Function 3
        JUMP Function 2
        JUMP Function 1
LibBase <inizio della struttura dati della libreria in memoria>
        <resto della struttura dati della libreria>
```

Quando si fa partire a freddo Amiga, i vari gruppi di librerie di sistema vengono cercate nella memoria del Kickstart e vengono copiate nella RAM dove possono essere modificate in seguito se fosse necessario. Ognuna delle librerie trovate è in seguito aggiunta alla lista delle librerie di sistema.

Quando il sistema viene acceso, una libreria può trovarsi ovunque nella memoria centrale. Così per far si che i programmi possano accedere alle routine in essa contenute, il programma stesso deve definire una variabile specifica - l'indirizzo base della libreria. Per esempio, ecco il codice richiesto per accedere alle funzioni di libreria dell'Intuition, dove per IntuitionBase si intende l'indirizzo base della libreria stessa:

```
#include "exec/types.h"
#include "intuition/intuition.h"
#include "intuiton/intuitionbase.h"

struct IntuitionBase *IntuitionBase;   /* deve essere di tipo global*/
extern struct Library *OpenLibrary();

main()
{
IntuitionBase = (stuct IntuitionBase *)
               Open Library("intuition.library",0);
if(IntuitionBase = = 0)
{
printf("L'Intuition non si apre!\n")
}
/* continuazione del programma */
```

Quando viene compilato (vedi appendice A) un programma è necessario linkarlo con un file chiamato amiga.lib, che aggiunge uno speciale codice di interfaccia per le funzioni di libreria.

Per una data funzione di libreria, amiga.lib fa le seguenti cose:

- Salva i valori di tutti i registri così che il programma possa continuare quando termina l'esecuzione della funzione
- Carica un registro con l'indirizzo base della libreria nella quale si trova
- Fornisce il registro con i parametri di questa funzione
- Salta a un indirizzo noto di offset rispettando l'indirizzo base della Library e esegue la funzione

- Ricarica nei registri i valori che essi avevano prima della chiamata della funzione
- Restituisce un valore quando è necessario

Nel caso si sbagli a definire la libreria alla quale si riferisce la funzione, il programma di Link segnalerà:

```
-<nomeLibBase> undefined
```

Se si richiama una funzione di una libreria di base ma non si apre la libreria il programma, quasi sicuramente, si impianterà.

Se si richiama una funzione di libreria e si apre correttamente la libreria corrispondente, si potrà accedere alle routine con successo. L'appendice A del Manuale della ROM Kernel di Amiga riporta i nomi delle variabili di indirizzo base così come le librerie alle quali esse sono associate. Di nuovo, se viene dimenticato, il programma di Link segnalerà quale libreria dovrà essere richiamata e aperta.

La dos.library sarà automaticamente aperta dal codice di Startup (partenza) che viene linkato col programma da eseguire (AStartup.obj o Lstartup.obj). Di conseguenza si potrà accedere alle funzioni della dos.library senza aprire la libreria alle quali esse sono associate. Si possono trovare altre informazioni sulle librerie nel terzo capitolo.

Per chi programma in C

Tutti gli esempi in questo libro usano il linguaggio C, in parte perché attualmente ci sono già due compilatori C per Amiga, ognuno con un supporto completo per la definizione di file di Amiga (conosciuti come file Include). Inoltre tutte le strutture dati trattate sono facilmente esprimibili, e una grossa percentuale del sistema operativo di Amiga è stato sviluppato proprio in questo linguaggio.

Quando si utilizza il C o il Pascal o qualsiasi altro linguaggio ad alto livello, il programma finale codificato ad alto livello, deve essere linkato con una libreria di routine che adattano i parametri che passano le convenzioni del nostro codice complesso ai parametri che passano le convenzioni del codice di sistema di Amiga.

Per chi programma in Assembler

Se si usa l'Assembler per richiamare le routine di sistema, bisogna conoscere molto bene l'uso che Amiga fa dei registri del 68000. Innanzi tutto, i registri D0, D1, A0 e A1 sono sempre trattati come registri di Scratch. Essi possono essere usati per maneggiare dati in Input. Però, non è garantito il fatto che il loro contenuto venga sempre salvato o ripristinato dopo l'uso di una qualsiasi routine. Il sistema salva o ripristina sempre i valori degli altri registri.

Le funzioni che restituiscono dei valori, li memorizzano nel registro D0. Se una funzione restituisce più valori bisogna assicurarsi di ottenere sempre l'indirizzo del vettore o della struttura dati che la funzione restituisce.

Un solo registro ha un trattamento speciale nel sistema di Amiga: A6. Esso non è mai usato per passare dei parametri. Il nome di questo registro è SysBase. Esso contiene l'indirizzo della tabelle di vettori funzione che, a turno, contengono l'indirizzo corrente delle varie funzioni di sistema. Quando è richiamata una funzione, l'Exec salta agli indirizzi che trova nella tabella per richiamare una determinata routine. Così un programmatore può modificare tale tabella per far sì che le funzioni di sistema facciano operazioni differenti o, per inserire un codice di debug o di profilo in linea con le funzioni di sistema per meglio analizzare i risultati dell'esecuzione del programma.

I file Include

I file di questo tipo, che si trovano sul disco del compilatore C o dell'Assembler, provvedono a definire le costanti di sistema e le strutture di dati. Si possono trovare i dettagli riguardanti questi file - che hanno nomi terminanti in .h (per il C)

o in .i (per l'Assembler) - leggendo le varie parti di questo libro. Questi file non sono listati nel libro, perché il lettore potrà facilmente trovare l'Amiga ROM Kernel Manual, dove questi file sono completamente listati. Lo scopo di questo volume è di supportare e non di soppiantare quelle informazioni.

Si potranno trovare le descrizioni di molte delle routine di sistema nelle pagine che seguono, organizzate in un tour guidato del software di sistema. Questo libro mostra cosa debba avvenire prima che una determinata routine possa di fatto girare e suggerisce perché sia utile o necessario usare una determinata routine. Spero che questo approccio di tipo tutorial sia utile. Si noti che, nonostante la maggior parte dei programmi di esempio siano completi, sarà utile avere sottomano l'Amiga ROM Kernel Manual per poterli modificare.

Capitolo 2

L'AmigaDOS

Nonostante Amiga abbia un'interfaccia basata su un sistema a icone (il Workbench), possiamo anche trovarvi implementate tutte quelle funzioni che tradizionalmente si trovano su un normale terminale. Questo capitolo descrive le caratteristiche, tipiche di un terminale, dell'AmigaDos. Si potrebbe trovare questo capitolo estremamente familiare come contenuti. Esso mostra come fare ciò che segue:

- Scrivere sul CLI
- Trasferire i valori dei parametri attraverso l'AmigaDos
- Ottenere una stringa dal CLI
- Aprire e chiudere i file
- Leggere e scrivere i file
- Altre operazioni riguardanti i file
- Aprire una propria Window CON: per l'input-output
- Aprire una propria Window RAW: per l'input-output
- Utilizzare i comandi e le Utility dell'AmigaDos

Tutti i programmi di questo capitolo devono essere aperti dal CLI perché ognuno di essi invia i dati che lo riguardano direttamente alla Window CLI dalla quale è stato lanciato. Ci sono essenzialmente due modi per lanciare un programma su Amiga: o iniziarne l'esecuzione dal CLI, o selezionando la sua icona (se esiste) dal Workbench.

Su Amiga per accedere a una qualsiasi routine del software di sistema è necessario prima aprire la libreria nella quale essa risiede. Si noteranno le chiamate di OpenLibrary usate in tutti i capitoli successivi. Le chiamate di OpenLibrary non saranno invece usate in questo capitolo perché il codice di Startup (Astartup.obj o Lstartup.obj), che viene linkato col programma, apre la libreria del Dos automaticamente. Praticamente i programmi di Startup aprono per noi questa libreria di funzioni. Così il codice di interfaccia di libreria non deve essere aperto in questi esempi. Per gli esempi dei capitoli che seguiranno invece, si troverà sempre il codice di interfaccia (descritto inizialmente nel terzo capitolo). Inoltre, se in futuro si vorranno usare dei propri codici di Startup, sarà necessario includere una chiamata di apertura della libreria del Dos per accedere a qualunque funzione dell'AmigaDos.

Stampare sul CLI

Tradizionalmente capita che il primo programma che si trova in un libro riguardante la programmazione in C o in Pascal sia uno che scrive sullo schermo le parole "Hello world". Piuttosto che rompere questa tradizione, ecco lo stesso programma per Amiga. Come si può notare non è molto diverso o più complicato di altri programmi simili.

```
/* hello.c */

main()
{
   printf("Hello world\n");
}
```

Le istruzioni per compilare questo programma e molti altri programmi del libro si possono trovare espresse in breve nella appendice A. Si usi la versione standard del programma makesimple (sul disco dell'Amiga C nella directory degli esempi) per compilare e lanciare questo programma. Una volta compilato e lanciato, questo programma stampa sullo schermo la frase "Hello world" all'interno della Window del CLI.

Se hello.c sarà compilato in un programma eseguibile di nome hello, lo si potrà lanciare digitando sul CLI :

```
hello
```

Trasferire parametri ai programmi dal CLI

I file di Startup (Astartup.obj e Lstart.obj) provvedono al passaggio dei valori dei parametri nello stesso modo in cui vi provvedono Unix o l'MS-DOS nei quali un programma scritto in C può ottenere gli argomenti (i parametri) che appaiono nella frase che, viene usata per lanciare il programma, insieme al nome del programma stesso. Per esempio, se si è scritto un programma di nome argecho e si digita la seguente linea:

```
argecho primoarg secondoarg terzoarg
```

allora, all'avvio del programma si conteranno tre parametri, e gli Array degli argomenti (argv[0], argv[1], argv[2]) punteranno a particolari stringhe contenenti, ognuna, il valore di uno degli argomenti passati. Nel listato 2.1 è scritto un programma, di nome argecho che stampa il contenuto degli argomenti che riceve.

```
/* argecho */

#include "exec/types.h"
#include  "libraries/dosextens.h"

main(argc,argv)
int argc;
char **argv;
{
      int j;
      for (j=0; j<argc; j++)
            printf("Il numero dell'argomento %ld è %ls\n",j,argv[j]);
      }
}
```

Listato 2.1

Redirezione dell'input-output standard

Dal CLI si possono usare dei simboli di redirezione, simili a quelli di Unix, per chiedere all'AmigaDos di utilizzare una finestra diversa da quella corrente per visualizzare l'input-output standard. (standard è usato in senso di default, cioè ciò che è usato se non vi sono disposizioni diverse). Si può fare questo usando da CLI i simboli, < e >, di redirezione. Diversamente da Unix, tuttavia, essi devono essere usati dopo il comando primario prima di qualsiasi argomento si voglia passare al proprio programma.

Qui vi sono due esempi di redirezione, usando il programma argecho e un ipotetico programma chiamato mycopy. Questo esempio copia sull'output standard (stdout) ciò che vi è nell'input standard (stdin), assumendo che esista un programma chiamato mycopy che semplicemente legge i caratteri uno alla volta dall'input standard e li riscrive uno alla volta sull'output standard:

```
mycopy < filesorgente > filedestinazione
```

questo significa "mycopy (preleva stdin dal) filesorgente (metti lo stdout nel) file-destinazione.

L'esempio seguente trasferisce il risultato di argecho all'interno di un file detto echofile:

```
argecho > echofile arg1 arg2 arg3 arg4
```

Qui, il simbolo di redirezione e il file di destinazione sono eliminati dall'AmigaDos e il programma riceve solo:

```
argecho arg1 arg2 arg3 arg4
```

Questo significa che vi saranno quattro argomenti che saranno listati nel file di nome echofile.

Quando si usa la redirezione, l'AmigaDos di fatto apre un file handle che usa internamente per maneggiare e controllare il flusso di dati di input-output. Quando il comando viene eseguito questo file di uso interno viene chiuso automaticamente. Il nostro programma non deve necessariamente accorgersi del ridirezionamento, esso semplicemente si occupa di prelevare le informazioni dal suo input standard e di inviarle al suo output standard, sarà l'AmigaDos che si occuperà di ogni altra operazione. Le notazioni stdin e stdout sono di fatto definite nei file di Startup proprio per queste operazioni di manipolazione (file handle). Se non si ridefinisce un path per stdin e stdout, essi saranno considerati tra i file handle della window di Console corrente.

La sezione successiva tratta l'uso diretto degli operatori di file (file handle), specificando dove sia utile creare e usare deliberatamente questi operatori per ottenere un certo risultato.

I file handle

Un file handle è un puntatore che si ottiene dall'AmigaDos quando si apre un file per la lettura o la scrittura. Così esso è usato sostanzialmente per operazioni come la redirezione del comando Execute, la funzione Write, la funzione Read e la funzione Close che chiude le operazioni di accesso al file.

La funzione printf, una funzione standard nell'Amiga C, è usata molto comunemente per inviare dati di output standard al Path che si è assegnato. C'è però un secondo modo per inviare un output alla window corrente del CLI. Si può infatti aprire questa window all'input-output con un file handle.

Si noti che i file handle dell'AmigaC sono differenti dai descrittori di file dell'AmigaDOS. Infatti non si possono passare all'AmigaDOS dei file handle, ottenuti attraverso l'AmigaC, e non si possono neppure passare, all'AmigaC, i descrittori di file ottenuti attraverso la funzione Open dell'AmigaDOS.

Si deve scegliere se usare le funzioni di input-output dell'Amiga C o le funzioni di input-output dell'AmigaDos. Semplicemente ci si assicuri di usare il file handle corretto per ogni funzione.

Se si useranno le funzioni dell'Amiga C per le operazioni di input-output ovunque sia possibile, sarà molto più facile tradurre i propri programmi perché girino su altre macchine, nonostante, usando direttamente le operazioni di input-output dell'AmigaDos, si ottiene un programma che gira più efficientemente su Amiga.

Il manuale dell'Amiga C descrive le funzioni standard di input-output, come putchar o getchar. Questo libro mostra invece come usare le funzioni dell'AmigaDos come Read e Write.

Aprire un file

Per ottenere l'accesso a un file è necessario un file handle. Si può ottenere un file handle - un puntatore a una determinata struttura di dati che contiene informazioni circa il file - usando la funzione Open dell'AmigaDos.

La chiamata della funzione Open, per aprire un file, prende questa forma:

```
filehandle = Open(pathname,accessmode);

struct FileHandle *filehandle, *Open();
char *pathname;
int accessmode;
```

La funzione Open necessita di due parametri. Il primo parametro è un puntatore alla stringa che definisce il nome del file. L'identificatore del nome del file può contenere anche il nome completo del path della directory, come in questo esempio. Se si specifica solo il nome, la directory usata è quella corrente.

Il secondo parametro è o MODE_OLDFILE o MODE_NEWFILE.(Questi termini sono definiti nel file Include dos.h). MODE_OLDFILE apre un file già esistente per la lettura e la scrittura con il puntatore di file posizionato all'inizio del file stesso. MODE_NEWFILE crea un nuovo file con questo nome, sempre per la lettura e la scrittura, e sempre con il puntatore all'inizio del file stesso. Se si usa MODE_NEWFILE per un file già esistente, la versione precedente viene cancellata (se è cancellabile) e viene creato un file vuoto con questo nome. Se la versione già esistente del file non è cancellabile (vedi protezione di un file più avanti nel capitolo), la funzione Dos Open non funzionerà.

Chiusura di un file

Si può terminare l'attività di un file chiudendo il file stesso. La chiamata della funzione Close prende questa forma:

```
Close(filehandle);

struct FileHandle *filehandle;
```

La funzione Close richiede un solo parametro: il file handle restituito dalla funzione Open. Essa termina l'accesso al file e stampa ogni output che sia stato bufferizzato internamente dall'AmigaDos.

Lettura di un file

Si possono ottenere delle informazioni da un file usando la funzione Read. Una chiamata della funzione Read prende questa forma:

```
actual_count = Read(filehandle,buffer,count);
```

Si dice all'AmigaDos dove trovare il file aperto passandogli un file handle ottenuto dalla funzione Open. Si tiene inoltre il conto dei caratteri che devono essere letti e si specifica l'indirizzo del buffer di memoria nel quale i caratteri saranno memorizzati:

```
struct FileHandle *filehandle;
char *buffer
in count,actual_count;
```

L'AmigaDos restituisce il conteggio dei caratteri che vengono letti.

Se con l'uso della funzione Read si ottiene il numero di caratteri che ci si era prefissi, allora actual_count conterrà il numero dei caratteri letti. Se actual_count è invece 0, allora vuol dire che è stata raggiunta la fine del file e nessun carattere è stato letto. Se, poi, actual_count è -1, allora vuol dire che nella fase di lettura è avvenuto un errore. Si possono ottenere maggiori informazioni circa questo errore dalla funzione IoErr. La funzione di AmigaC, di nome getchar usa la funzione Read.

Scrittura di un file

Si possono aggiungere informazioni a un file scrivendoci sopra dei dati. Una chiamata della funzione Write prende questa forma:

```
actual_count = Write(filehandle,buffer,count);
```

Si dice all'AmigaDos dove locare il file aperto, passandogli un file handle ottenuto dalla funzione Open. Si specifica anche il numero di caratteri che saranno scritti, e l'indirizzo del buffer di memoria dal quale saranno prelevati questi caratteri:

```
struct FileHandle *filehandle;
char *buffer;
int count,actual_count;
```

L'AmigaDos restituisce, in actual_count, il numero di caratteri scritti.

Se actual_count è uguale a -1, allora è avvenuto un errore. Si possono allora avere più informazioni circa l'errore dall'IoErr. La funzione di biblioteca di Amiga, di nome putchar usa la funzione Write.

Input-Output di Console usando i file-handle dell'AmigaDos

Quello seguente è un programma che apre la Window corrente della CLI per l'Input_Output. Come hello.c, esso scrive "Hello world", ma questo programma lo fa con un metodo diverso. In questo esempio la notazione * significa che deve essere usata la Window corrente per locarvi il path standard di input-output. La notazione "%ls" è un descrittore del formato di stringa che dice alla funzione fprintf come stampare la stringa. Il simbolo "%" significa che quello è l'inizio dell'output formattato. Il simbolo "l" significa che il puntatore alla locazione dei dati della stringa è a 32 bit (cioè LONG); il simbolo "s" significa che i dati sono di tipo ASCII.

```
#include "libraries/dosextens.h"
/* definisce il file handle */

extern struct FileHandle *Open()
/* dichiara il tipo di funzione*/
main()

{
   struct FileHandle *dos_fh;
   dos_fh = Open("*",MODE_OLDFILE);
   /* apre la console */

   fprintf(dos_fh,"%ls","Hello world\n");
   /* funzione amiga.lib */
   Close(dos_fh);
}
```

Aprire una nuova window per l'output

Al posto della window corrente, si può aprire una nuova window di Console per l'input-output cambiando la stringa significativa del comando Open. Ecco la versione modificata del programma precedente; questa versione apre una nuova window:

```
#include "libraries/dosextens.h"
extern struct FileHandle *Open()
main()
{
   struct FileHandle *dos_fh;
   dos_fh = Open("CON:10/10/500/150/New Window",
   MODE_NEWFILE);

   Write(dos_fh,"Hello world\n",13);
   Delay(300)     /* 6 secondi di ritardo */
   Close(dos_fh);
}
```

La window è di tipo CON:, cioè essa opera esattamente come una Console. (Si può trovare di più sulle console nel prossimo paragrafo). Essa sarà posizionata in modo da avere il vertice superiore sinistro nel punto di coordinate dello schermo 10,10, e sarà larga 500 pixel e alta 150. Il suo nome è New Window. La funzione Delay dell'AmigaDos (dove i tempi sono espressi in cinquantesimi di secondo) è usata affinché sia possibile osservare chiaramente l'Output prima che la finestra si chiuda e scompaia. Questa volta il modo per l'Open è settato a MODE_NEWFILE, perché la finestra non è quella corrente.

Ricevere l'Input nella window di Console

La stessa window di Console che è stata usata per l'output, può essere utilizzata per l'input. La nuova finestra diventa attiva non appena viene aperta (a meno che, naturalmente, non si selezioni col mouse una window differente).

Così se si digitano dei caratteri sulla tastiera, essi verranno automaticamente inviati alla nuova window. Il listato 2.2 mostra una modificazione del programma precedente; questo programma richiede una linea in Input all'utente.

Se si fa girare questo programma, si vedrà che solo i caratteri stampabili sono accettati dalla window di Console, e nulla sarà stampato nella Window originale del CLI finché non verrà premuto il tasto return. I tasti funzione e i tasti per lo spostamento del cursore non avranno alcun effetto. In breve, l'AmigaDos sta accettando dei caratteri immessi da tastiera e li sta filtrando.

Poiché l'AmigaDos sta filtrando i dati immessi, si possono usare solo quei semplici comandi di editing o, come ad esempio la pressione del tasto Del per far arretrare il cursore e cancellare l'ultimo carattere immesso, oppure premere contemporaneamente il tasto Ctrl e la X per reinserire una frase completamente nuova. Quando si è finita la digitazione della frase, premendo il tasto Return si comunica all'AmigaDos di passare la linea stessa al programma.

Inserire dati senza l'uso del tasto Return

Nell'esempio precedente, la finestra CON accettava dati Input filtrati e aspettava la pressione del tasto Return prima di ritornare qualsiasi cosa dalla chiamata della funzione Read. Se si deve però tenere conto della pressione dei singoli tasti, come spesso accade, invece di una Window di tipo CON si può usare una Window RAW:.

Il listato 2.3 mostra un programma che riporta al CLI la pressione di ogni tasto così come avviene. Se si premono tasti che non siano strettamente della tastiera (tasti funzione, tasti cursore, o il tasto Help), si noterà che più di un valore è generato dalla pressione di uno di questi tasti. Quando viene premuta la Q il programma finisce.

```
#include "libraries/dosextens.h"
extern struct FileHandle *Open;

main()
{
      char userinput[256];
      int howmany;
      struct FileHandle *dos_fh;

      dos_fh = Open("CON:10/10/500/150/New Window",MODE_NEWFILE);
      write(dos_fh,
                  "Scrivi una frase e poi premi Return\n", 45);
      howmany = Read(dos_fh,userinput,255);
```

```
        userinput[howmany]= '\0';
        printf("Hai inserito %ld caratteri,\n"howmany);
        printf("eccoli qua:\n");
        printf("%ls\n",userinput);
        Close(dos_fh);
}
```

Listato 2.2

```
#include "exec/types.h"
#include "libraries/extens.h"
#define QUIT 0x51 /* la lettera Q shiftata */

extern struct FileFandle *Open();

main()
{
        char userinput[256];
        int howmany,j;
        struct FileHandle *dos_fh;

        dos_fh = Open("RAW:10/10/500/150/New Window",MODE_NEWFILE);

        for(;;)
        {
                howmany = Read(dos_fh,userinput,255);
                userinput[howmany] = '\0';
                        printf("ho appena letto %ld valori:\n",howmany);
                /* notare che se l'utente scrive velocemente, */
                        /* più  di un valore sarà riportato all'interno */
                        /* di uno stesso intervallo di tempo */
                printf("hai immesso questi dati: ");
                for(j=0; j<howmany; j++)
                {
                        printf("%lx ",userinput[j]);
                }
                printf("\n");
                if(userinput[0]==QUIT) break;
        }
        Close(dos_fh);
}
```

Listato 2.3

Si può anche rilevare quando l'utente preme o rilascia il singolo tasto. Comunque questa operazione non è effettuata dall'AmigaDos. Per ottenere un identificatore della singola pressione dei vari tasti, si usa un particolare meccanismo della Intuition chiamato IDCMP (Intuition Direct Communication Message Port). Questa Utility permette anche la lettura dei movimenti del mouse e della pressione dei tasti del mouse stesso (Click). L'IDCMP è descritto nel capitolo 5.

Inviare dati a una stampante

L'AmigaDos permette di inviare dati a una stampante che sia connessa alla porta seriale o a quella parallela. Ci sono tre differenti modi per trasmettere dati a una stampante:

SER:	per utilizzare l'interfaccia seriale attraverso il device serial
PAR:	per utilizzare l'interfaccia parallela attraverso il device parallel
PRT:	per utilizzare la porta di stampa, che può essere quella seriale o quella parallela a secondo di ciò che è specificato nelle Preferences; questo modo utilizza il device printer

Questi device possono essere aperti come un qualsiasi file dell'AmigaDos, e la funzione Open restituisce un file handle che permette di redirezionare o di ricevere dei dati nel programma corrente. Il listato 2.4 mostra un programma che apre il printer.device e invia un file alla stampante.

Il comando che è implementato in questo programma è simile alla combinazione dei seguenti comandi del CLI:

```
JOIN nomeFile1 nomeFile2 altrinomiFile AS nomeJoined
Type > PRT: nomeJoned
```

Si noti che la funzione Open può aver usato sia il device parallel che quello serial. In questo caso l'output sarà direzionato con le modalità che sono state scelte nelle preferences (numero di baud, parità, numero di Bit e così via).

L'AmigaDos provvede a definire i modi in cui una stampante dovrebbe reagire ai vari codici standard di controllo. Quando si utilizza il device printer, i codici di controllo che saranno inviati alla stampante saranno tradotti seguendo ciò che è stato impostato nelle preferences riguardo la stampante in uso. Quando si utilizza il device serial o quello parallel, non avverrà questa traduzione; La stampante in questo caso riceverà esattamente ciò che gli viene inviato. Si può trovare utile questa possibilità nel caso si voglia controllare una stampante che non è presente nella lista di quelle controllate direttamente dalle Preferences.

Il manuale della ROM Kernel di Amiga fornisce ulteriori informazioni su come si suppone che una stampante reagisca ai codici di controllo, e come raggiungere e utilizzare il device printer direttamente, invece che attraverso l'AmigaDos.

Ulteriori funzioni per la manipolazione dei File

Ecco alcune ulteriori funzioni che sono di solito utilizzate per gestire e modificare i file:

Seek	sposta la posizione del puntatore all'interno del file o semplicemente indaga circa la posizione corrente del puntatore nel file stesso.
IsInteractive	determina se il file handle è associato a una Console.
WaitForChar	attende un carattere dalla Console ma solo per un tempo limitato

```
/*printem.c*/
#include "libraries/dosextens.h"
extern struct FileHandle *Open();

main(argc,argv)
int argc;
char *argv[];
{
      struct Filehandle *fh, *fh2;
      int datasize;
      int n;
      char buffer [256];
      n = 1;
```

```
if(argc < 2)
{
      printf("Formato: printem <nomefile> [anche più file]\n");
      exit(0);
}
fh = Open("PRT:",MODE_OLDFILE);
if(fh == 0) exit(20);         /* problemi con la stampante */

while(argc < 1)
{
fh2 = Open(argv[n],MODE_OLDFILE);
if(fh2 == 0)
{
      printf("%ls: file not found\n",argv[n]);
}
else
for(;;)      /* per sempre (fino a un errore, magari a un EOF */
      {
      datasize = Read(fh2,buffer,256) /* legge il file */
      Write(fh,buffer,datasize);      /* scrive l'Output */
      if(datasize< 256) break;
      /* è stato letto meno di quanto richiesto? */
      /* deve cercare se c'è un IOerr() qui per vedere se EOF */
      }
      Close(fh2);
      n++; argc--;
}
Close(fh);
}
```

Listato 2.4

Ricercare una posizione all'interno di un file

Quando l'AmigaDos apre un file e restituisce un file handle, il file è posizionato al suo stesso inizio. Si può utilizzare la funzione Seek per trovare una posizione qualsiasi all'interno del file.

Si può specificare la posizione desiderata in relazione alla posizione corrente, in relazione alla posizione iniziale o in relazione alla posizione finale. La funzione Seek restituisce il valore della posizione corrente. La grandezza di un file è specificata in byte, così la posizione attuale è data come posizione di byte nel file.

Una chiamata della funzione Seek assume questa forma:

```
currentpositon = Seek(filehandle,position,relative_to_what);

struct FileHandle *filehandle;
int position;
int relative_to_what;
int currentposition;
```

Per muoversi alla fine del file per aggiungere qualcosa, si digiti:

```
currentposition = Seek(filehandle,0,OFFSET_END);
```

Per sapere la posizione corrente senza muoversi nel file, si digiti:

```
currentposition = Seek(filehandle,0,OFFSET_CURRENT);
```

Per spostarsi di dieci byte all'interno del file, si digiti:

```
currentposition = Seek(filehandle,10,OFFSET_CURRENT);
```

Per tornare all'inizio del file, si digiti:

```
currentposition = Seek(filehandle,0,OFFSET_BEGINNING);
```

Determinare se il proprio programma è connesso a un terminale

Si può determinare se l'Input standard del proprio programma è connesso alla CLI usando la funzione IsInteractive. Questa funzione restituisce un valore non nullo se il programma è stato lanciato dal CLI, ritorna invece il valore zero se si è lanciato il programma dal Workbench o con delle funzioni dell'Exec. Quando si lancia un programma dal CLI, si provvede a costruire una Window nella quale l'utente può scrivere e rispondere. Questo significa che il programma è interattivo (cioè può scambiare dati con l'utente). Se il programma è stato lanciato da un'altra funzione, allora non ci sarà in generale una finestra di input e quindi l'input standard non è interattivo. Questo permette di decidere di non mostrare all'utente certe istruzioni in output, attendendo ad esempio la pressione di un tasto che potrebbe non avvenire mai.

La chiamata della funzione ha questa forma:

```
status = IsInteractive();


int status;
```

Attendere un carattere per un tempo determinato

Se (e solo se) un file handle è connesso a un terminale interattivo (una finestra del CLI), allora la funzione WaitForChar può essere usata per attendere per un tempo determinato che un certo carattere sia inserito. Magari si vuole far lampeggiare lo schermo o fare un beep e ripetere un messaggio se l'utente non risponde. O magari si vuole uscire dal programma se nessun tasto viene premuto in un tempo determinato.

Si possono specificare i file handle sia per il valore in Input, sia per il tempo di attesa del programma prima che esso ricominci a eseguire le istruzioni successive. Il valore restituito dalla funzione WaitForChar è un parametro Booleano (Vero <> da zero, falso = zero) che avverte se c'è un carattere che attende di essere letto. Si può decidere cosa fare di questo. La chiamata è di questo tipo:

```
status = WaitForChar(filehandle,timeout);
BOOL status;
struct FileHandle *filehandle;
int timeout;
```

Si noti che si deve specificare il valore di timeout in cinquantesimi di secondo.

Questo è un uso semplice del multitaskig di Amiga, nel quale il proprio programma (task) si blocca finché non viene premuto un certo tasto o scade il tempo di attesa. Nel frattempo tutti gli altri programmi possono tranquillamente girare mentre il proprio rimane inattivo in attesa.

Struttura della directory dell'AmigaDos

L'AmigaDos implementa un sistema gerarchico di archiviazione dei dati. Si può pensare al sistema di archiviazione immaginando che il disco sia un registro con delle cartellette e dei fogli. Le cartelle con i file sono le sottodirectory del disco. I singoli fogli sono i vari file dati o i vari programmi. Le cartellette possono contenere i fogli o altre sottocartellette.

Quando si richiede un listing del disco, con il comando List, l'AmigaDos mostra tutti i nomi presenti sul disco stesso, specificando quali appartengano a file e quali alle directory, usando per queste la notazione (dir) posposta al nome. La directory primaria (root) è quella che si trova in cima alla gerarchia del disco. Ogni altro file è contenuto in essa, o contenuto in sottodirectory che sono a loro volta contenute in essa.

In AmigaDos si specifica la directory primaria, usando i due punti. Se si digita:

```
CD:
```

significa "rendi la directory corrente (CD) la directory primaria di questo disco". Se il path della directory corrente è df0:test/mystuff, allora il comando "CD:" rende df0: la directory corrente. Se la directory corrente è df1:a/b/c, allora "CD:" rende df1: la directory corrente. Questo comando sposta al livello massimo, o alla root del corrente path di directory.Dalla directory primaria si può discendere nella gerarchia assumendo come directory corrente una delle sottodirectory elencate in quella primaria.

Per esempio, se un comando dir eseguito dalla directory primaria mostra :

```
tests(dir)
```

si può rendere questa la directory corrente digitando:

```
CD tests
```

o si può raggiungerla direttamente, da qualsiasi altra directory specificando esattamente il path per accedere alla directory corrente:

```
CD df0:tests
```

Si può fare riferimento a un preciso disco AmigaDos usando il suo nome (per esempio, mydisk, testfiles/firsttry, discoprova) invece di usare semplicemente la designazione di un disco (per esempio df0:xxx o df1:xxx). L'AmigaDos, comunque, richiede che il disco in questione sia inserito in un drive qualunque prima di continuare le varie operazioni. Il nome di un disco viene definito quando lo si formatta, o quando si utilizzano le funzioni DISKCOPY o RENAME dal Workbench. Si ricordi questa possibilità quando si esamineranno i paragrafi successivi.

Se si è già un programmatore di Amiga, le informazioni appena descritte riguardo i nomi dei Path e dei dischi sono sicuramente familiari. Comunque, si vuole semplicemente ricordare che può essere necessario, nell'esecuzione di un programma, muoversi lungo la gerarchia di archiviazione dati di Amiga per accedere a particolari file. Il paragrafo seguente spiega come si può permettere al proprio programma di muoversi in tale gerarchia.

Muoversi all'interno dell'AmigaDos

Si trova molto utile poter digitare i comandi dalle Window del CLI e ottenerne l'esecuzione attraverso l'AmigaDos. Quando si programma utilizzando l'AmigaDos, si possono trovare e utilizzare direttamente le funzioni per cancellare e rinominare i file. Inoltre, si possono direttamente usare tutti quelle utility che permettono, ad esempio di copiare un file o cambiare directory.

In aggiunta alle funzioni di accesso al disco, si possono utilizzare dei comandi tipo CLI direttamente dal programma usando la funzione Execute dell'AmigaDos. Esso, infatti, accetta una stringa di comando come se fosse stata scritta su una Window del CLI, e la esegue come se fosse presente il CLI.

Execute ha due restrizioni:

- Il comando RUN deve essere presente nella directory definita come C dall'istruzione ASSIGN.
- Il comando che si esegue deve essere o nella directory corrente, o nella directory C.

Quello seguente è un programma che utilizza il comando Execute. Si noti che la stringa comando contiene un operatore di redirezione per inviare l'output del comando su di un file.

```
/* execute.demo.c */

#include "libraries/dosextens.h"

main()
{
   int success;
   success = Execute("dir > df0:dir.file",0,0);
   if(success = = 0)printf("I/O error %ld",IoErr());
}
```

Il programma carica la lista della directory corrente all'interno di un file chiamato dir.file sul Drive df0. Per vedere il risultato di questa operazione si digiti:

```
TYPE df0:dir.file
```

La funzione Execute richiede tre parametri. Il primo parametro è la stringa comando che può contenere anche operatori per la redirezione, che sono un simbolo di minore (<) per specificare da dove proviene l'Input standard, e un simbolo di maggiore (>) per specificare la direzione che deve assumere l'output standard.

Il secondo e il terzo parametro sono dei file handle di redirezione; essi specificano come devono essere direzionati l'input e output standard se non ci sono operatori per ridirezionamento contenuti nella stringa comando passata come primo parametro. Un valore zero per questi parametri fa si che l'AmigaDos assuma che l'input e l'output standard, per il comando Execute, devono essere direzionati come l'input e l'output del processo primario che richiama il comando stesso. Così, un semplice comando DIR stampa il proprio output direttamente

nella Window del CLI se il programma execute.demo è stato lanciato dal CLI stesso.

Usare un comando o chiamare la funzione Execute

Come tutte le routine finora trattate, chiamare un comando direttamente invece di usare la funzione Execute ha sia vantaggi sia svantaggi. Se si usa il comando in se stesso, senza interpellare la funzione Execute, non si devono rispettare le limitazioni della funzione Execute (cioè la presenza obbligatoria nella directory C del comando invocato dall'Execute e del comando RUN). D'Altronde, usando il comando direttamente non si usufruisce di tutte quelle caratteristiche tipiche di Execute.

Usare il file handle restituito dalla funzione Open

Come precedentemente menzionato, si può ottenere una manipolazione dei file handle da parte dell'AmigaDos mediante l'uso della redirezione sulla linea di comando, oppure si può aprire da soli un file e utilizzare i file handle che restituisce la funzione Open. Ecco qui un altro esempio che dimostra l'uso dei simboli di redirezione:

```
DIR > dir.list
```

Questo esegue il comando DIR e pone il suo Output in un file chiamato dir.list. Il listato 2.5 mostra un programma che fa esattamente la stessa cosa, ma non usa simboli si ridirezionamento all'interno della linea di comando; si ottiene, invece, un file handle aprendo un file di nome dir.list residente sul drive interno. L'effetto finale è identico, esso mostra semplicemente l'uso dei parametri dei file handle nella funzione Execute. Si farà ancora uso dei file handle in questo capitolo.

Muoversi lungo i rami delle directory ad albero

L'esempio di Execute precedente semplicemente eseguiva un comando che aveva effetto solo sulla directory corrente. Può essere necessario muoversi in un'altra directory, o, per esempio, chiedere all'AmigaDos di richiedere all'utente l'inserimento di un altro disco.

L'AmigaDos possiede un meccanismo che gli permette di muoversi all'interno del suo sistema di archiviazione dati. Questo meccanismo è chiamato lock (blocco). Caricare un lock in una directory significa semplicemente dire all'AmigaDos di riferire tutte le proprie richieste di accesso ai file a una particolare directory.

```
/* execute.demo2.c */

#include "libraries/dosextens.h"
extern struct FileHandle *Open();
main()
{
      int success;
      struct FileHandle *outhandle;

      outhandle = Open("df0:dir.list",MODE_NEWFILE);
      if(outhandle == 0)
      {
            printf("I/O error %ld\n",IoErr());
            exit (20);
      }
      success = Execute("dir",0,outhandle);
      if(success == 0)
            printf("i/O error %ld\n",IoErr());

      Close(outhandle);           /* chiude il file */
}
```

Listato 2.5

Questa directory può essere la directory primaria di un disco particolare o una qualsiasi subdirectory all'interno del disco iniziale. Una volta che si ha un lock su di una directory si possono richiedere a tale directory informazioni riguardanti la directory stessa o riguardanti i file all'interno della directory. Non è necessario usare un lock solo per aprire, chiudere o leggere un file. Il lock è un meccanismo

che permette a un programma di accedere e, possibilmente, manipolare i dati che sono gestiti dall'AmigaDos.

Esso è del tutto necessario in operazioni che comprendano il multitaskig. Se ad esempio un certo programma utilizza una directory con un lock, allora l'AmigaDos impedirà che qualsiasi altro programma agente contemporaneamente possa distruggere tale directory o altre directory ad essa legate. Poiché il corretto uso dell'AmigaDos dipende da un utilizzo appropriato di questo meccanismo di blocco, è necessario sbloccare tutte le directory contenenti lock prima di uscire dal programma.

Il listato 2.6 è un programma che utilizza un lock per settare la directory di uso corrente. Questo programma equivale all'uso del comando cd <alcunedirectory> dal CLI, ma è invece è attivo per il proprio programma. Il programma non cambia la directory nella quale lavora il CLI, ma cambia la directory nella quale lavorerà il programma stesso. Invece di listare la directory nella quale sta operando il CLI, il programma my.cdir sposta la directory di lavoro alla directory C del disco. La lista è generata sulla Window del CLI.

Se si vuole provare il programma, lo si compili e si salvi il file eseguibile come my.cdir. Si digiti il comando DIR per ottener un lista della directory all'interno della quale ci si trova. Si digiti poi my.cdir e si otterrà invece un listato della directory df0:c. Si scriva poi CD e si noti che ci si trova nella directory dalla quale si era partiti, mentre solo il programma si è spostato nella directory C per compiere il proprio lavoro.

Risalire lungo le directory ad albero

Ci sono varie funzioni necessarie per muoversi lungo le directory dell'AmigaDos:

IoErr	restituisce il valore dell'errore avvenuto nella più recente operazione Dos
Lock	garantisce il controllo di un particolare Path di directory

UnLock	rilascia il controllo del Path bloccato Lock
CurrentDir	si muove all'interno di una determinata directory

```
/* my.cdir.c */
#include "libraries/dosextens.h"
extern struct FileLock *Lock(),*CurrentDir();
main()
{
      int success;
      struct FileLock *lock,*oldlock;

      /* preleva il Pointer per una determinata directory /
      lock = Lock("df0:c",ACCESS.READ);
      if (lock == 0)
      {
            printf("\nNon posso operare un lock!");
            exit(20);
      }
      /* si sposta nella directory se il Pointer è valido */
      Oldlock = CurrentDir(oldlock);
      success = Execute("dir",0,0);
      if(success == 0)
      {
            printf("errore di execute:  %ld\n",IoErr());
            /* ritorna alla directory originale */
            lock = CurrentDir(oldlock);
            Unlock(lock);
            exit(40);
      }
      /* ritorna alla directory originale */
      Oldlock = CurrentDir(oldlock);

      /* e cancella con un comando di unlock tutti i lock */
      /* ottenuti con Lock(). */
      /* si noti: non va cancellato nulla che si sia */
      /* ottenuto dalla  Current Dir... */



      /* altrimenti si fa dimenticare all'AmigaDos */
      /* l'esistenza del disco. Solo i lock ottenuti */
      /* con Lock() o DupLock() */
      /* possono essere cancellati con UnLock. */
      Unlock(lock);
}
```

Listato 2.6

Examine	riempe una struttura dati con le informazioni riguardanti una certa directory
ExNext	riempe una struttura dati con informazioni riguardanti i file di una certa directory
ParentDir	sposta in una directory superiore

Lettura dei codici di errore dell'AmigaDos. Quando si usano delle funzioni dell'AmigAdos, nella sequenza di chiamata spesso si specifica:

```
success = Funzione(parametri);
```

oppure

```
pointer = Funzione(parametri);
```

dove un valore zero significa che è avvenuto un errore e la funzione non ha funzionato.Si può ricercare il motivo per il quale la funzione non ha portato a termine il suo compito, leggendo il codice di errore dell'AmigaDos. Questo codice è accessibile attraverso la funzione IoErr. Una chiamata di questa funzione assume questa forma:

```
error = IoErr();
```

I possibili errori che si possono presentare sono elencati nel file Include dell'Amiga C chiamato libraries/dos.h.

Bloccare i file e le directory. Una chiamata della funzione Lock assume questa forma:

```
mylock = Lock(pathname_string,access_mode);
```

La funzione Lock richiede due parametri. Il primo consiste in una stringa che contiene il path della directory che si vuole bloccare. La stringa può essere un path completo, contenente il nome del disco, il nome di una directory all'interno della directory corrente o una stringa nulla (""). Si vedrà più avanti nel capitolo un uso frequente della stringa nulla come parametro.

Il secondo parametro è il modo di accesso, specificato come ACCESS_READ, anche chiamato SHARED_LOCK. Se si specifica uno shared lock, allora gli altri processi possono anche leggere e scrivere all'interno di quella directory o in quel file. L'altro modo di accesso è chiamato ACCESS_WRITE o EXCLUSIVE_LOCK. Se lo si usa allora nessun altro processo avrà accesso a quel file o a quella directory.

La funzione Lock restituisce un pointer a una struttura dati FileLock, che contiene informazioni che serviranno poi, all'AmigaDos, per accedere a un file (se si blocca un file) o per accedere al contenuto di una directory (se si blocca una directory).

La funzione Lock ha inoltre un accesso più veloce a un file della funzione Open, così per vedere se un file esiste si può tentare di bloccarlo. Se la funzione ritorna con un valore valido (cioè non nullo), allora il file esiste e si può decidere di aprirlo, Se Lock ritorna un valore nullo, allora il file non è stato trovato.

Sbloccare i file e le directory. Per richiamare la funzione Unlock si opera così:

```
UnLock(mylock);
```

Il parametro è un puntatore a una struttura dati FileLock. Si deve sbloccare qualsiasi cosa sia stata precedentemente bloccata, per permettere all'AmigaDos di operare correttamente.

Spostarsi da una directory ad un'altra. Si usa CurrentDir per spostarsi da una directory ad un'altra. Il valore di oldlock serve in futuro per rispostarsi nella directory da dove si proviene. Una chiamata della funzione CurrenDir ha questa forma:

```
oldlock = CurrentDir(mylock);
```

dove mylock è un puntatore a un bloccaggio, ottenuto chiamando la funzione Lock o un'altra funzione che operi un bloccaggio. Ha lo stesso effetto del comando AmigaDos CD, che cambia la directory di lavoro. Quelle seguenti sono sequenze di chiamata equivalenti:

```
success = Execute("cd df0:c",0,0);
```

e

```
mylock = Lock("df0:c",ACCESS_READ);
oldlock = CurrentDir(mylock);
```

Il vantaggio della seconda forma è che non richiede la presenza nella directory C del disco corrente del comando RUN.

Solo i bloccaggi che sono stati ottenuti con la funzione Lock possono essere disattivati più tardi. Non si usi mai UnLock con un valore ottenuto dalla funzione CurrentDir. Infatti, l'AmigaDos crea i propri blocchi che riguardano la directory corrente; il valore che ritorna è semplicemente un puntatore a un suo blocco privato. Se il proprio programma lo sbloccasse, l'AmigaDos non sarebbe più in grado di accedere a quel file o a quella directory. Così, si usi UnLock solo per blocchi ottenuti attraverso la funzione Lock.

Ottenere informazioni su un file o una directory. Si usa la funzione Examine per ottenere informazioni circa un file o una directory. Una chiamata di Examine ha questa forma:

```
success = Examine(lock,address_of_FileInfoBlock);
```

Il primo parametro è un puntatore a un blocco, di solito ottenuto con la funzione Lock. Il secondo parametro è l'indirizzo di un FileInfoBlock. Quando viene chiamata la funzione Examine, un FileInfoBlock è riempito con le informazioni riguardanti una directory. Fra le voci presenti in questo FileInfoBlock vi sono la DirectoryType, il FileName, i bit Protection, il Size, i Comment, e così via. Il significato dei campi presenti nel FileInfoBlock è riassunto nell'esempio del listato 2.7

```
/ exam.example */

#include "libraries/dos.h"
#include "exec/memory.h"

long rmask = ((long)('r') < 24);
long brmask = ((long)(' ') < 24);
/* un blank nella stessa posizione */
long wmask = ((long)('w') < 16);
long bwmask = ((long)(' ') < 16);
long emask = ((long)('e') < 8);
```

```
long bemask = ((long)(' ') < 8);
long bdmask = (long)(' '); < long dmask = (long) ('d')

struct
{
      long pmask;
      char stringnull;
}
      maskout;
      /* setta la costruzione dei valori dei bit di protezione */

main()
{

      struct FileInfoBlock *fib;
      int success, p;
      struct FileLock *lock;
      extern struct FileLock *Lock();
      /* esaminiamo il 'dir' command file */
      fib = (struct FileInfoBlock *)AllocMem(sizeof(struct
            FileInfoBlock),MEMF_CLEAR);
      lock = Lock("df0:c/dir", ACCES_READ);
      if(lock)
      {

            success = Examine(lock,fib);
            if(success)
            {
            printf("\n file name: %ls,&fib->fib_FileName[0]);
                  if(fib->fib DirEntryType > 0)
                        printf("\nè una directory");
                  else
                        printf("\nè un file ");
                              /* ora calcola i Bit di protezione */
                  p = fib->fib Protection;
                  maskout.pmask = 0;
                  maskout.stringnull = '\0';
                  /* fine della stringa nulla */

                  if(p & FIBF_READ)
                        makout.pmask |= brmask;
                  else
                        maskout.pmak |= rmask;
                  if(p & FIBF_WRITE)
                        maskout.pmask |= bwmask;
                  else
                        maskout.pmask |= wmask;
                  if(p & FIBF_EXECUTE)
                        maskout.pmask |= bemask;
                  else
                        maskout.pmask |= emask;
                  if(p & FIBF_DELETE)
                        maskout.pmask |= bdmask;
                  else
                        maskout.pmask |= dmask;
```

```
            printf("\nha bit di protezione di valore: %ls",&maskout)
            printf("\nha un file lungo (in Bytes) %ld",fib->fib_Size);
            printf("\(%ld in blocchi)",fib->fib_NumBlocks);
            printf("\ncommento:\n%ls",fib->fib_Comment);
      /* c'è anche una datestamp per la data nel FileInfoBlock */
      /* che può essere interpretata dalla funzione ShowDate */
      /* (si veda Trovare la data corrente più@ avanti nel capitolo */
            printf("\nha come data più@ recente:");
            ShowDate(&(fib->fib_Date));

            }
      Unlock(lock);
}                       /* fine del programma exam.example.c */
```

Listato 2.7

Quando ci si trova nella directory primaria del disco, il campo FileName contiene il nome del disco stesso. Così, per ogni disco, quando si è nella directory primaria, si può leggere il nome del disco e la creazione della timestamp. Attraverso la timestamp infatti si possono distinguere due dischi aventi lo stesso nome.

Il listato 2.7 è un programma che chiama la funzione Examine. Il programma mostra solo quei campi che possono essere di interesse per il programmatore. Gli altri campi sono esclusivamente di uso interno dell'AmigaDos.

La ragione per cui si è fatto uso della funzione AllocMem per creare il FileInfoBlocks è che l'AmigaDos necessita di un allineamento per le longword. AllocMem allinea sempre i limiti delle longword quando alloca la memoria.

Ottenere informazioni su directory o file consecutivi. Per ottenere informazioni riguardanti file o directory dello stesso livello si usa la funzione ExNext. Una chiamata della funzione ExNext ha questa forma:

```
success = ExNext(lock,address_of_FileInfoBlock);
```

I parametri sono gli stessi richiesti dalla funzione Examine. La funzione ExNext usa il puntatore di lock e il contenuto corrente del FileInfoBlock per determinare quale sia (se ce ne sono) il file o la directory successivi allo stesso livello gerarchico. Essa poi riempie FileInfoBlock con le informazioni riguardanti il file o la directory successivi.

Il valore di success è nullo se non vi sono file o directory successivi.

Risalire l'albero gerarchico delle directory. Se si ha un lock in una directory, la funzione ParentDir ottiene un lock sulla directory superiore; cioè lo ottiene sulla directory della quale quella di lavoro è parte. Una chiamata alla funzione ParentDir ha questa forma:

```
parentlock = ParentDir(lock);
```

Si usa questa funzione per risalire l'albero gerarchico delle directory sino alla directory primaria. Quando la si è raggiunta non vi sono directory ad essa superiori così la funzione restituisce un valore nullo.

Programmi che utilizzano le funzioni sulle directory. Il listato 2.8 è un programma che implementa alcune delle funzioni della linea comando:

```
DIR opt a
```

Come richiamo, ricordiamo che questa linea comando ordina all'AmigaDos di listare tutti i file della directory corrente, tutte le directory contenute nella directory corrente e tutti i file e le directory al di sotto di queste. La differenza tra questo comando e la versione di sistema è che esso mostra i nomi così come li trova e non in un sequenza organizzata.

Si può far girare il programma opta da un qualsiasi Path di directory, ed esso mostrerà tutti i file e le directory sotto di sé.

Determinare la directory di lavoro corrente

Si può determinare quale sia la directory corrente grazie a un particolare caratteristica dell'AmigaDos. Usando una stringa nulla con la funzione Lock, l'AmigaDos ottiene un lock sulla directory corrente. Si possono quindi avere informazioni su questa directory e muoversi da essa a quella superiore.(Il programma seguente legge il nome della directory e si sposta in quella superiore). Di fatto spostandosi da una directory a quella superiore si può risalire tutto l'albero gerarchico sino alla directory primaria del sistema di file corrente.

Dopo aver ottenuto un lock, si possono avere informazioni circa questa directory usando la funzione Info, o si può semplicemente salvare il lock con le informazioni per un uso successivo. Si potrebbe decidere di andarsene dalla directory corrente e poi ritornarvici per una ragione o per l'altra.

Il listato 2.9 è un programma che fornisce informazioni riguardanti la directory corrente. Si noti che utilizza la funzione UnLock prima di terminare. Si devono sbloccare infatti tutti lock prima di proseguire così che l'AmigaDos possa proseguire tranquillamente. Questa precauzione è simile a quella di liberare tutta la memoria che si è allocata dopo avèrla usata nel proprio programma.

```
/* opta.c */

#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "exec/memory.h"
extern struct FileLock *Lock().*DupLock(),*CurrentDir();
main()
{

      struct FileLock *oldlock;
      oldlock = Lock("",ACCESS_READ);
      if (oldlock! = 0 )
      {
            followthread (oldlock, 0);
      }
      else
      {
            printf("non posso bloccare la Dir corrente\n");
      }
            printf("\n");
}
/* ora segui il filo... potresti trovare un file o una directory */
/* se trovi un directory, listala e poi prosegui più giù */
/*(in modo ricorsivo), se trovi un file */
/* listalo e prosegui fino alla fine */

int followthread(lock,tab_level)
struct FileLock *lock;
int tab_level;
{

      struct FileInfoBlock *m;
      struct FileLock *newlock, *oldlock, *ignoredlock;
      int success,i;

      /* se sei alla fine della strada non stampare niente */
```

Listato 2.8 (continua)

```
        if(!lock) return(0);
        /* alloca lo spazio per un FileInfoBlock */

        m = (struct FileInfoBlock *)
              AllocMem(sizeof(struct FileInfoBlock),MEMF_CLEAR);
        success - Examine(lock,m);
              /* La prima chiamata di Examine riempie il FileInfoBlock */
              /* con le informazioni riguardanti la directory, se lo è */
              /* chiamata al livello primario, essa contiene il nome */
              /* del disco. Così questo programma stampa solo l Output */
              /* di ExNext, invece che quelli ExNext ed Examine */
              /* Se ne stampasse  due, allora listerebbe  doppiamente */
              /* gli accessi alle directory */
while (success !=0)
{
        if(m->fib_DirEntryType > 0)
        {
              /* finché è una directory, poni un lock su essa e */
              /* entra dentro per listare il suo contenuto */
              /* così com'è */
              /* il suo nome è */

        newlock = Lock(m->fib_FileName[0],ACCESS_READ);

        /* se il lock risulta valido allora questa directory diventa */
        /* quella corrente, ma salva il valore del precedente lock */
        /* così che possiamo tornare in questo punto e continuare a */
        /* listare il resto del contenuto della directory qui locata */

        oldlock = CurrentDir(newlock);
        /* si sposta in quella directory */

        /* segue ricorsivamente il filo giù  sino in fondo */
        followthread(newlock,tab_level+1);

        /* dopo aver listato la nuova directory ritorna qui */

        ignoredlock = CurrentDir(oldlock); /* e procede */
        }
        success = ExNext(lock,m); /* esamina la nuova entrata */
        if (success)
        {
              printf("\n");
              for(i=0; i<tab_level; i++)
              printf("\t");
              /* stampa un tab per mostrare il livello della directory */
              printf("%ls,&m->fib_FileName[0]);
              if(m->fib_DirEntryType >0)
              {
                    printf(" [dir]");
```

```
                /* dice all'utente che questa è una directory */
                }
            }
        }
        if(lock) UnLock(lock);
        FreeMem(m,sizeof(struct FileInfoBlock));
}
```

Listato 2.8

Il programma mybranch usa una chiamata ricorsiva alla funzione di nome followpath. Il suo compito è di chiamare la funzione ParentDir finché non ottiene un valore di lock nullo e di stampare il nome della directory entro la quale entra ad ogni passo. Il programma stampa due punti quando è arrivato alla directory primaria di quel disco, e stampa uno slash per ogni subdirectory incontrata lungo la strada sino al path da dove il programma era partito. Quando followpath raggiunge la directory primaria, il lock che viene ritornato fornisce il nome del disco.

```
/* mybranch.c */
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "exec/memory.h"
extern struct FileLock *Lock(), *DupLock(), *ParentDir();
main()
{

        struct FileLock *oldlock;
        /* lock di lettura per la directory corrente */
        oldlock = Lock("",ACCESS_READ);

        if(oldlock != 0)
        {
              printf("\nIl Path per la directory corrente è: ");

              /* non scrive uno slash se è all'ultimo livello */
              followpath(oldlock,0)
        }
        else
        {
              printf(\n Non posso bloccare la directory corrente");
        }
        /* Si noti che in questo esempio followpath sblocca lock */
}

int followpath(lock,printslash)
struct FileLock *lock;
int printslash;
```

```
{
      struct FileInfoBlock *myinfo;
      struct FileLock *newlock;
      int success,error;

      /* se raggiunge la fine non stampa nulla */
      if(!lock) return(0);

      myinfo = (struct FileInfoBlock *)AllocMem(sizeof(struct
            FileInfoBlock),MEMF_CLEAR)
      if(myinfo == 0)
      {
                  printf("Out of memory\n");
                  return(0);
      }
      /* guarda se questa directory ne ha una superiore, se è così */
      /* passa ad essa */
      newlock = ParenrDir(lock);
      error = IoErr();
      /* newlock potrebbe non funzionare per qualche errore I/O */
      /* o perché qualcuno ha tolto il disco */

      if(newlock == 0 && error != 0)
                  printf("\n Errore I/O ! numero = %ld\n",error);
      /* chiama ricorsivamente la stessa funzione per seguire */
      /*il Path sino alla directory primaria*/

      followpath(newlock,1);

      /* archivia nel FileInfoBlock così si può stampare */
      /* il nome di questo nodo */

      success = Examine(lock,myinfo);
      if(success)
      {
            printf("%ls",myinfo -> fib_FileName[0]);
            if(newlock == 0)
                  printf(":");
      }
      else
      {
      /*  stampa uno slash solo se il parametro non è nullo */

      if(printslash) printf("/");
            UnLock(lock);
      }
      if(myinfo)
            FreeMem(myinfo, sizeof( struct FileInfoBlock));
      return(1);
}
```

Listato 2.9

Per provare questo programma, lo si compili e si chiami l'eseguibile mybranch, poi lo si copi in df0:c. Poi si esegua la seguente sequenza di comandi dal CLI:

```
CD df0:
MAKEDIR stuff
CD stuff
MAKEDIR morestuff
CD morestuff
CD>Il CLI risponderà con


df0:stuff/morestuff
```

Ora si scriva il comando:

```
mybranch
```

Il CLI risponderà con:

```
VOLUMENAME:stuff/morestuff
```

Dove VOLUMENAME sarà il nome del CLI di StartUp del disco (possibilmente il C-CLI se sono state usate le istruzioni che vengono dall'Amiga C).

Utility dell'AmigaDos

Le seguenti Utility sono disponibili come funzioni dell'AmigaDos richiamabili direttamente da un programma:

- Rinominare un file o una directory (Rename)
- Cancellare un file o una directory (Delete)
- Creare una directory (CreateDir)
- Proteggere un file (SetProtection)
- Fissare un commento per un file o un a directory (SetComment)
- Leggere la data corrente (DateStamp)
- Ottenere informazioni su un disco (Info)

Ci sono altre funzioni di utility che possono operare in Multitasking. Esse sono trattate nel terzo capitolo, insieme a tutto ciò che riguarda le chiamata di sistema del Multitasking e le sue strutture dati.

Utility normalmente usate dal CLI

Delle utility sopra citate, le seguenti sono di norma utilizzate dalle funzioni del CLI: Rename (dal comando RENAME), Delete (dal comando DELETE), CreateDir (dal comando MAKEDIR), SetProtection (dal comando PROTECT), e SetComment (dal comando FILENOTE). In questo paragrafo, sono spiegati tre differenti metodi per richiamare queste Utility, compreso l'uso della funzione Execute.

Come convenzione grafica, per distinguere tra le linee che si dovranno scrivere sul CLI e quelle da usare nel programma, i comandi del CLI saranno annotati totalmente in maiuscolo. In tutto questo paragrafo, se qualcosa appare totalmente in maiuscolo, è un comando del CLI; se una frase comincia con il valore ritornato della variabile success allora è una frase da inserire solamente nel proprio programma.

Cambiare nome a un file. La funzione Rename prende questa forma:

```
success = Rename(vecchionomepointer,nuovonomepointer);

int success;
char *vecchionomepointer, *nuovonomepointer;
```

I parametri sono dei puntatori a stringhe che rappresentano rispettivamente il vecchio e il nuovo nome del file. Se il valore di success e zero, allora si può porre un lock al valore dell'IoErr per vedere che cosa non ha funzionato. Si noti che si può usare questa funzione per spostare un file da una directory di un certo livello ad una di un altro purché il file rimanga nello stesso disco.

Per esempio per muovere un file di nome myfile da df0:stuff/morestuff a df0:stuff, si dovrà specificare vecchionomepointer come df0:stuff/morestuff/myfile e nuovonomepointer come df0:stuff/myfile.

Per cambiare nome a un file dal CLI, si può scrivere:

```
RENAME from vecchionome to nuovonome
```

o si può usare Execute, come segue:

```
success = Execute("rename from vecchionome to nuovonome",0,0);
```

Un altro modo è:

```
success = Rename("vecchionome","nuovonome");
```

Cancellare un file. La funzione Delete assume questa forma:

```
success = Delete(nomecorrente);

int success;
char *nomecorrente;
```

Il parametro nomecorrente è un puntatore a una stringa che descrive il nome del path del file che deve essere cancellato. Se è semplicemente un nome, esso si riferisce alla directory corrente. Se success fosse nulla allora si può porre un lock al valore dell'IoErr per vedere cosa non ha funzionato.Per cancellare un file dal CLI, si può scrivere:

```
DELETE nomecorrente
```

o si può utilizzare Execute, come segue:

```
success = Execute("delete nomecorrente",0,0);
```

Un'altra alternativa è:

```
success = Delete("nomecorrente");
```

Creare una directory. Per richiamare la funzione CreateDir si scriva:

```
lock = CreateDir(pointernomedir);

struct FileLock *lock;
char *pointernomedir;
```

Per creare una directory dalla CLI,
si può scrivere:

```
MAKEDIR nuovadirectory
```

o si può usare la funzione Execute:

```
success = Execute("makedir nuova directory",0,0);
```

Un'altra possibilità è:

```
lock = CreateDir("nuovadirectory");
```

Si noti che l'uso della funzione CreateDir ritorna un lock alla directory appena creata. Se il valore del lock è nullo, l'IoErr contiene le informazione del perché la funzione non ha operato correttamente.

Proteggere un file. La funzione SetProtection permette di specificare una maschera di protezione per un determinato file. La chiamata della funzione SetProtection assume questa forma:

```
success = SetProtection(nomepointer,mask);

int success;
char * nomepointer;
int mask;
```

Solo i quattro Bit più bassi di mask sono significativi. Il loro significato è la sequenza RWED, che sta per Read, Write, Execute, e Delete. Se viene settato ognuno di questi bit, si dirà all'AmigaDos se un file deve essere protetto dalla lettura, dalla scrittura o ne deve essere impossibilitata l'esecuzione diretta (nel caso di file testo o di normali file binari eseguibili); o devono essere protetti dalla cancellazione. Lo stato di mask deve essere esattamente l'opposto di ciò che normalmente si trova quando si usa un comando LIST dell'AmigaDos nel quale il comando stesso mostra gli RWED come flag di protezione, qui il significato è che il file può essere letto, scritto, e così via.

Quando si setta uno di questi flag, significa che il Bit corrispondente sarà settato e di conseguenza il file sarà protetto. L'AmigaDos pone attenzione solo al valore del flag D. Se un programmatore crea un programma Shell (un programma che sembra e opera come una CLI di alto livello) il programma Shell potrà

usare gli altri flag. Per esempio, se il flag E non è settato il programma Shell potrebbe non tentare di eseguire quel file.

Per proteggere un file dal CLI, si può scrivere:

```
PROTECT nomefile DW
```

oppure si può usare la funzione Execute:

```
success = Execute("protect nomefile dw",0,0);
```

o in alternativa, si può usare la funzione SetProtection:

```
/* .....RWED........*/
/* binario 0101 come flag W e D */
/* write-protect e delete-protect */

success = SetProtection("nomefile",5);
```

Creare una filenote. Si può usare la funzione SetComment per dotare un file o una directory di una filenote. Una chiamata alla funzione SetComment assume questa forma:

```
success = SetComment(nomepointer,commento);

int success;
char *nomepointer;
char *commento;
```

Si può settare la filenote dalla CLI scrivendo

```
FILENOTE nomefile "Questo è il commento che voglio aggiungere"
```

Oppure si può utilizzare, nel proprio programma, la funzione Execute:

```
success = Execute("filenote nomefile"Ecco il commento"",0,0);
```

In alternativa si può usare la funzione SetComment direttamente:

```
success = SetComment("nomefile","Ecco il commento");
```

Trovare la data corrente. Si può richiedere all'AmigaDos di fornire la data corrente, così come esso l'ha in memoria, che non coincide necessariamente con quella esatta, a meno che non vi siano le giuste informazioni da parte dell'utente. La funzione è chiamata DateStamp, ecco come la si richiama:

```
DateStamp(v);
```

dove v è l'indirizzo della prima delle tre variabili long (32 Bit l'una) che la funzione riempie con le informazioni sulla data e sull'orario. Si possono convertire le informazioni così ottenute nel formato mese, giorno, anno con la procedura del listato 2.10. Questo programma è stato scritto da Tom Rokicki ed qui accluso con il suo permesso. Ringrazio Tom per questa subroutine.

Ottenere informazioni riguardanti il disco. Dopo aver ottenuto un lock su un file o una directory, si può usare la funzione Info per avere informazioni sul disco nel quale il file risiede. Si passa un lock a una struttura InfoData vuota, e la funzione Info la riempie con le informazioni riguardanti il disco. Questo è diverso dal comando INFO, nel quale la funzione Info ottiene le informazioni riguardanti un disco alla volta, poiché il comando CLI riunisce e formatta tutti i dati sulle Device con struttura a blocchi nel sistema di archiviazione in una sola volta.

Una chiamata della funzione Info appare così:

```
success = Info(lock,address_of_InfoData);
```

```
char *months[]={"","January","February","March","April","May","june, "Ju-
ly","August","September","October","November","December"};
long n;
int m,d,y;
main()
{
      long v[];
      DateStamp(v);
      ShowDate(v);
}

ShowDate(v)
      long *v;
{
      long n;
      int m,d,y;
      n = v[0]-2251;
```

```
        y = (4*n+3)/1461;
        n -= 1461 * (long) y/4;
        y += 1984;
        m = (5*n +2)/153;
        d = n-(153*m+2)/5+1;
        m +=;
        {
              y++;
              m -= 12;
        }
        printf("%s %d, %d\n", months[m],d,y);
        return(0);
}
```

Listato 2.10

Come fileInfoBlock questa struttura dati InfoData deve essere trattata long.

Il programma Info.example del listato 2.11 usa la funzione Info per ottenere, e poi fornire informazioni riguardanti qualunque disco possegga la funzione DIR. Questo è di solito il proprio disco Workbench o CLI.

Funzioni miste

Le funzioni discusse in questo paragrafo non sono direttamente accessibili dalla CLI, però le informazioni che esse raccolgono sono spesso accessibili ad altri comandi, come ad esempio LIST. Le funzioni qui accluse permettono di fare le seguenti cose:

- Scoprire il nome del disco di Boot
- Attendere per un certo periodo di tempo
- Determinare quale processo sta utilizzando un dato I/O
- Cambiare il nome del disco

Scoprire il nome del disco di Boot. Si può scoprire quale fosse il nome del disco che fu usato come disco di Workbench originale (il disco cioè che si inseri-

sce subito dopo Kickstart, quando il sistema richiede un disco di Workbench) passando un valore nullo al lock del comando Examine.

Non importa quanti cambi di dischi siano avvenuti, attraverso l'uso dell'istruzione ASSIGN, il sistema ricorda comunque il nome del disco usato come Boot originale.

```
/* info.example.c */

#include "libraries/dos.h"
#include "exec/memory.h"

main()
{
      struct InfoData *id;
      int success,p;
      struct Lock *lock;
      struct
      {
            long pmask;
            char stringnull;
      }
      maskout;
      maskout.stringnull = '\0';

      /* otteniamo informazioni sul disco dove è locata la Dir */
            id = (struct InfoData *)AllocMem(sizeof(struct InfoData),
                  MEMF_CLEAR);
      lock = Lock("df0:c/dir", ACCESS_READ);
      if(lock)
      {
            success = Info(lock.id);
            if(success)
            {
                  if(id->id_Disktype == -1)
                  {
                        printf("\nNON CI SONO DISCHI");
                  }
                  else
                  {
                  printf("\nErrori Soft sin qui: %ld,id->
                  id_NumSoftErrors);
                  printf("\# dell'unità dove (è/era) montato: %ld",
                        id->id_unitNumber);
                  printf("\nStato del disco: ");
                  if(id->id_DiskState == ID_WRITE_PROTECTED)
                        printf("Write-Protected");
                  else if(id->id_DiskState == ID_VALIDATED)
                        printf("Read-Write");
```

```
                else if(id->id_DiskState == ID_VALIDATING)
                        printf("Validating Disk file Structure");
                printf("\nIl disco ha %ld blocchi",id->id_NumBlock);
                printf("\ndei quali %ld sono stati utilizzati"
                id->iid_NumBlocksUsed);
                printf("\nvi sono %ld byte per blocco",
                id->id_BytesPerBlock);
                printf("\nTipo di disco: ");
                maskout.pmask = id->id__DiskType;
                printf(&maskout);
                if(id->id_InUse ==0)
                        printf("\nIl disco è utilizzato");
                else
                        printf("\nIl disco non è utilizzato");
        }
        }
    }
}       /* fine di Info.example.c */
```

Listato 2.11

Il programma bootname mostrato nel listato 2.12 usa la funzione Execute per leggere e riportare il nome del disco dal quale è avvenuto i boot di Amiga.

Attendere per un periodo di tempo specificato. Si può usare la funzione Delay dell'AmigaDos se si vuole che il proprio programma resti inattivo per un certo periodo di tempo specificato in cinquantesimi di secondo. La chiamata della funzione prende questa forma:

```
Delay(time);

int time;
```

Ecco un esempio:

```
Delay(150)    /* rende inattivo il programma per 3 secondi */
```

Determinare quale processo sta utilizzando una certa Device di I/O. Questa tecnica è necessaria se si vogliono usare un paio o più delle più avanzate funzioni dell'AmigaDos. Invece di eseguire una funzione Dos di Utility come mostrato precedentemente, per certe funzioni è necessario inviare un messaggio a un determinato processo. Questo paragrafo, e il prossimo esauriscono la descrizione delle caratteristiche dell'AmigaDos, ma il passaggio di messaggi è trattato

esaurientemente nel capitolo seguente, così qui si troverà solo una breve spiegazione della tecnica di passaggio dei messaggi.

```
/* bootname.c */

#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include "exec/memory.h"

main()
{
      struct FileInfoBlock *myinfo;
      int success;

      myinfo = (struct FileInfoBlock *)
            AllocMem(sizeof(struct FileInfoBlock),MEMF_CLEAR);
      success = Examine(0,myinfo);
      if(success)
            printf("%ls\n",&(myinfo->fib_FileName[0]));
      else
            printf("ci sono problemi\n");
      FreeMem(myinfo,sizeof(struct FileInfoBlock));
}
```

Listato 2.12

Per un determinato device dell'AmigaDos si può determinare quale processo stia di fatto utilizzando il suo input-output, usando la funzione DeviceProc. Per richiamare la funzione DeviceProc si usi:

```
proc = DeviceProc(nome);
```

dove *nome* è una stringa null-terminated che contiene il nome del device in questione. Il parametro nome potrebbe essere "df0:","df1:", o ""(stringa nulla) se deve operare su qualunque directory corrente.

Cambiare nome a un disco. Se si deve cambiare nome a un disco da programma, si può usare la funzione Execute per richiamare il comando RELABEL, o si può inviare un insieme di messaggi al processo che sta utilizzando la Device contenente quel disco. Il formato del messaggio è ACTION_RENAME_DISK. Si trova quale sia il processo usando la funzione DeviceProc. Un esempio di programma, per un caso semplice, è mostrato di seguito:

```
/* relabeldisk */

/*nota: è solo mostrato come usare la funzione Execute */
/* il programmatore dovrà costruirsi la propria stringa */
/* per la funzione RELABEL        */

main()
{
   int success
   success = Execute("RELABEL vecchionome nuovonome",0,0);
   if(success)
   {
         printf("Execute' non ha funzionato\n");
   }
}
```

Questo capitolo ha trattato da vicino tutto ciò che può fare l'AmigaDos, dall'input-output basato sulla Console sino al trattamento dei file. Si è visto come aprire e chiudere i file, ottenere dati in input usando il Dos, e come muoversi e manipolare il sistema di archiviazione a file.

L'approccio è stato di tipo causa-effetto, piuttosto che esplicativo delle strutture dati usate internamente dall'AmigaDos. Se si necessita di maggiori informazioni su di esse, si veda l'AmigaDOS Technical References Manual e i file di tipo Include di nome libraries/dos.h e libraries/dosextens.h. Questo tipo di libreria è descritto esaurientemente nell'Amiga ROM Kernel Manual.

Se si sta tentando di usare le informazioni contenute in questo libro per adattare ad Amiga programmi scritti per altre macchine, c'è un'altra area molto importante che va conosciuta a fondo - l'input-output dei device. Si troveranno parecchie informazioni sui device per tutto il testo, con una trattazione specifica nel sesto capitolo.

Capitolo 3

L'Exec

Questo capitolo tratta ciò che è necessario sapere per dialogare correttamente con l'Exec. Ecco i principali argomenti trattati:

- La struttura dell'Exec
- Alcune delle sue routine di base
- Alcune delle sue funzioni di supporto

Non si troverà un'analisi approfondita di tutte le funzioni dell'Exec. Molte delle funzioni eseguite dall'Exec possono essere considerato ad alto livello. Piuttosto che impantanarci nella minuziosa descrizione delle caratteristiche avanzate del sistema, si preferisce dare una chiarificazione di quelle parti che sono alla base della comprensione di tutto il sistema stesso.

La struttura dell'Exec

L'Exec è una struttura esecutiva Multitasking basata su liste. Ogni cosa che gestita dall'Exec, si trova su una lista da qualche parte nel sistema. Multitasking significa che è in grado di caricare e far girare più programmi, saltando avanti e indietro da uno all'altro così velocemente da far apparire questi programmi, detti task, come operanti contemporaneamente.

Segue una lista delle funzioni dell'Exec:

- Alloca la memoria su richiesta dei vari task selezionandola in base a una lista dei blocchi di memoria liberi.
- Controlla i task gestendo le liste di task che stanno girando, che sono pronti a operare, o che stanno attendendo che avvenga qualcosa prima di essere nuovamente pronti a operare.
- Mantiene una lista delle librerie di funzioni che permettono ai vari task di spartirsi comuni codici di programma e liste di device (chiamate anche device driver) che i task possono usare per l'I/O di sistema.
- Mantiene una lista degli operatori di Interrupt per gestire i vari interrupt hardware e software generati dal 68000, dai chip custom, e dal software di sistema.

Perché le liste sono importanti

L'Exec utilizza delle liste linkate per permettere al sistema operativo di Amiga di avere una configurazione dinamica e di non avere limitazioni arbitrarie. Quando si accende la macchina e si opera un boot, l'intero sistema viene creato ex novo.

Alcune parti del software di sistema di Amiga richiedono che venga settata a parte la memoria per il loro uso interno. Se si ha nel proprio Amiga una quantità di memoria che va oltre le capacità di gestione dei chip custom (alcune espansioni creano questa situazione), allora il software di sistema può utilizzare di fatto questa memoria che esce da range lasciando un maggior spazio della memoria inferiore a disposizione dei chip Custom. Questo permette di avere uno spazio maggiore a disposizione per la grafica, per il suono, per molti Sprite, e così via.

Alcune funzioni dell'Exec e loro terminologia

Nei paragrafi seguenti si troverà un esame di alcune funzioni dell'Exec che manipolano o controllano ciò che segue:

- task e Process
- Allocazione della memoria
- Liste
- Segnalazioni
- Porte di messaggio
- Messaggi
- Librerie
- Device

Quegli utenti che siano principalmente interessati alla grafica di Amiga, possono direttamente passare al capitolo successivo, dove comincia appunto il discorso sulla grafica. Si faccia però attenzione che saranno usate, senza ulteriori spiegazioni, negli esempi di grafica delle routine qui introdotte. Si noti inoltre che

la piena comprensione dei messaggi, e delle porte di messaggio è essenziale per un corretto uso dei device di input-output. I device di input-output sono qui introdotti. I dati di I/O per i singoli device sono trattati nel sesto capitolo.

Task e process

Come menzionato sopra, l'Exec mantiene una lista dei task che possono essere fatti girare dal 68000. Se un task sta attendendo una certa forma di input, esso può usare varie funzioni di sistema per mettersi in stato di inoperatività per dare la possibilità agli altri task di girare. Inoltre, ogni qualvolta avviene un interrupt - come ad esempio un interrupt del timer, un carattere in arrivo dalla tastiera o dalla interfaccia seriale, o il completamento di una schermata - l'Exec controlla la sua lista di task e fornisce l'uso del microprocessore al task prioritario che sia pronto a girare. Dall'osservazione della lista può risultare una interruzione dell'esecuzione del task corrente a favore di uno che sia prioritario.

Per ogni task, l'Exec mantiene ciò che è chiamato il blocco di controllo del task. Vi è un solo microprocessore nel sistema che deve essere spartito tra i vari task. Così l'Exec usa il blocco di controllo del task per salvare il valore di tutti i registri macchina quando il task in questione non sta girando (è cioè "addormentato"). Il processo che porta a salvare lo stato dei registri di un task, a ripristinare i registri con i valori di un altro task, e a rendere attivo il nuovo task è detto task Switching.

Quando un task torna attivo ("sveglio"), l'Exec ripristina tutti i valori dei suoi registri macchina e il task continua le sue operazioni come se non avesse mai perso il controllo del microprocessore.

Un process è qualcosa di superiore a un task. Un blocco di controllo di processo come è def

inito per Amiga - il nome della struttura dati è "Process" - contiene un blocco di controllo del task insieme a molte altre strutture dati utilizzate dall'AmigaDos. La differenza fondamentale tra processo e task è nelle diverse funzioni di sistema che ognuno di essi può utilizzare. Se in particolare si vuole creare un programma che sia capace di girare indipendentemente del programma che lo ha generato, e si vuole utilizzare ogni funzione di I/O dell'AmigaDos, si dovrà creare

un process piuttosto che un task. L'AmigaDos utilizza le informazioni contenute nel blocco di controllo di processo per le proprie operazioni di I/O. Queste informazioni non sono disponibili nel blocco di controllo di base del task. Il nono capitolo tratta le caratteristiche dei processi e dei task in dettaglio.

Allocazione della memoria

L'Exec mantiene una lista delle aree di memoria disponibile nel sistema. Un task o un Process possono richiedere delle fette di memoria all'Exec. Quando si è terminato l'uso della memoria, si deve informare l'Exec affinché la riponga nuovamente nella lista della memoria disponibile.

Una volta che l'Exec ha allocato una parte di memoria per un certo task, esso non sa più nulla a proposito di questa. L'Exec gestisce solo aree di memoria libera, non di memoria allocata, così la si deve restituire al sistema quando si ha finito di utilizzarla. Altrimenti sarà persa fino al successivo reboot del sistema.

Allocazione semplice della memoria

Vi sono vari livelli di allocazione della memoria che possono essere utilizzati dall'Exec. In questo capitolo si tratterà solo il livello di base; questo è anche il metodo usato più spesso negli esempi di questo libro.

Per richiamare la routine di base per l'allocazione della memoria si può operare così:

```
indirizzo = AllocMem(size,requirements);
```

dove size è il numero di Byte che si desidera che il sistema allochi per i propri scopi, e requirements dice al sistema che tipo di memoria si deve utilizzare e cosa fare prima che sia compiuta l'operazione di allocazione, e indirizzo è l'indirizzo di partenza (il più basso) del blocco di memoria che il sistema deve allocare. Se il sistema non è in grado di allocare la quantità di memoria richiesta, allora il richiamo della funzione AllocMem restituisce un valore nullo.

Si deve sempre controllare tale valore per essere sicuri di avere la possibilità d'uso di un certo blocco di memoria.

Ecco il significato del valore dei requirements:

MEMF_CHIP	Indica che la memoria di utilizzo è quella accessibile dai chip
MEMF_FAST	Indica che la memoria di utilizzo è quella non accessibile dai chip.
MEMF_CLEAR	Setta a zero tutto il blocco di memoria fornita prima di informare l'utente della posizione della memoria stessa.
MEMF_PUBLIC	In preparazione di una possibile gestione della memoria (implementazione di memoria virtuale) assicura che tale memoria è allocata in uno spazio pubblico (non swappabile) continuamente accessibile da ogni task.

Quando la memoria è allocata in uno spazio MEMF_CHIP, significa che lo hardware con compiti specializzati può prelevare i dati che vengono li scritti. Si userà questo requirement per costruire spazi da visualizzare, buffer per il disco, forme d'onda sonore, e Sprite.

Quando la memoria è allocata in spazi MEMF_FAST, essa risiede al di fuori dell'area accessibile ai chip con compiti specializzati. Questa memoria si presta bene per scriverci programmi e dati di tipo non DMA. In certe circostanze, come in casi di un pesante uso di una attività DMA (alta risoluzione, grafica a 16 colori o spostamento di grosse masse di dati da parte di quel particolare chip detto Blitter), l'attività DMA dei chip specializzati può rallentare il 68000.
Se, mentre questa pesante attività è in corso sui Bus di memoria dei chip specializzati e il 68000 sta invece utilizzando una area di memoria localizzata nella RAM al di fuori del range, non vi saranno conflitti per l'uso dei singoli bus e pertanto non vi saranno rallentamenti. Così specificare MEMF_FAST, poiché fornisce una memoria che non viene divisa tra i vari cicli di funzionamento dei chip specializzati può velocizzare il sistema.

Usando MEMF_CLEAR si può ottenere automaticamente l'azzeramento di un'area di memoria. Questo è molto utile per inizializzare gli Array e le strutture dati, e permette di accorciare alcune parti dei propri programmi.

Il requirement MEMF_PUBLIC non è attivo per ora, ma permette di aggiungere ad un programma, in futuro, un possibile uso di una nuova versione del sistema operativo. Questo requirement deve essere applicato alla memoria utilizzata per i DMA di sistema, assegnata per l'uso di messaggi o di porte di messaggio tra task cooperanti, o assegnata ai codici di interrupt dei programmi e alle e loro strutture dati.

Restituire la memoria allocata precedentemente all'insieme della memoria libera

Una volta terminato l'uso della memoria, la si deve restituire al sistema usando la funzione FreeMem. Per richiamare la funzione si operi così:

```
FreeMem(address,size);
```

dove per address si intende l'indirizzo di partenza del blocco di memoria allocata con l'uso di AllocMem, e size è la dimensione della memoria che si era richiesta ad AllocMem.

Nel richiamare entrambe le funzioni AllocMem e FreeMem, il sistema arrotonda automaticamente il valore di size e libera il multiplo più vicino ad esso di MEM_BLOCKSIZE. (Il valore corrente di MEM_BLOCKSIZE è descritto nel File Include di nome exec/memory.h).

Programma per l'allocazione della memoria

Il programma seguente mostra la corretta sintassi per richiamare le funzioni AllocMem e FreeMem, e dimostra che si può liberare esattamente la stessa quantità di memoria che si era allocata.

```
#include "exec/memory.h"
main()
{
   char *address;

   address = AllocMem(300,MEMF_CHIP|MEMF_CLEAR);
   FreeMem(adress,300);
   address = AllocMem(120,Mef_FAST|MEMF_CHIP|MEMF_PUBLIC);
   FreeMem(address,120);

   address = AllocMem(12345,0);
   FreeMem(address,12345);
}
```

Si specifica la combinazione dei requirement di memoria come un OR logico dei singoli requirement. Specificare 0 come requirement significa "dammi questa quantità, in ogni tipo di memoria disponibile". Il sistema come Default tenta prima nella MEMF_FAST, poi nella MEMF_CHIP.

Le liste

Le liste sono composte di due tipi di strutture dati, la struttura List, che è di fatto una intestazione di lista (list header), e la struttura Node (anche chiamato nodo di lista) che è un componente della lista stessa.

Una intestazione di lista potrebbe essere pensata come un'ancora della lista. Infatti le routine di base per la manipolazione delle liste specificano l'intestazione di lista come uno dei parametri quando richiamano le funzioni. In altre parole, l'intestazione di lista specifica quale lista si vuole manipolare.

Inizializzare un'intestazione di lista

Il punto importante da ricordare circa le intestazioni di lista è che devono essere inizializzate in modo appropriato prima di poterle usare. Fortunatamente la libreria di Amiga contiene una funzione che permette di fare correttamente questa operazione, la funzione NewList.

Per richiamare tale funzione si opera così:

```
NewList(address_of_listheader);
```

dove address_of_listheader è un puntatore all'indirizzo iniziale di un blocco di memoria che serve da intestazione di lista. Non c'è bisogno di inizializzare in modo particolare questo blocco; la funzione NewList lo gestisce completamente da sola.

Significato dei nodi di lista

Il nodo di lista non è che un pezzetto di memoria che fa parte di molte altre strutture dati per le quali la lista è stata costruita. La cosa facile dei nodi di lista è che essi richiedono poca o nulla inizializzazione al fine di usare le routine di base del sistema.

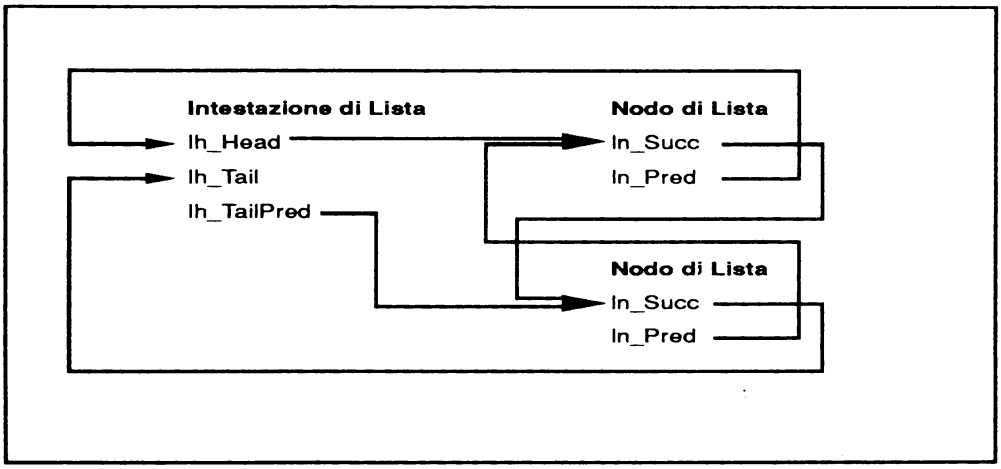


Figura 3.1

Si può richiamare la funzione AddHead (per aggiungere una voce che diventi la prima della lista) o AddTail (per aggiungere una voce alla fine della lista) e molte altre funzioni, senza il bisogno di sapere cosa avviene all'interno del nodo di lista.

La figura 3.1 è un diagramma che mostra una intestazione di lista (List Header) e un paio voci della lista, ognuna delle quali contiene un nodo di lista (List Node) come parte della sua struttura. Si noti che un insieme di puntatori all'interno sia dell'intestazione sia dei nodi è usato per unire tutte queste parti per formare una lista completa. L'Exec mantiene sia un puntatore di avanzamento (che punta alla voce seguente della lista) sia un puntatore di retrocessione (che punta alla voce precedente della lista).

Oltre alle voci all'interno dei nodi che sono usate per mantenere legata la lista, ci sono altri due campi utilizzati spesso dalle routine di sistema: il campo di nome e il campo di priorità.

Il campo di nome può essere usato per puntare a una stringa a terminazione nulla che contiene il nome del nodo di lista. Alcune delle routine di sistema permettono una ricerca per nome, come per i task o per le porte (FindTask, FindPort).

Il campo di priorità può essere usato per segnalare a una determinata routine di sistema in che ordine un certo numero di voci debbano essere aggiunte a una lista già esistente. I nodi che hanno una precedenza elevata, saranno inseriti prima di quelli che hanno una precedenza inferiore. Le routine che tipicamente usano il campo di priorità sono AddTask e AddPort. I valori di priorità possono andare da -128 a +127. Per molte applicazioni la priorità può essere lasciata a 0.

Routine che manipolano le liste

L'Exec contiene le routine mostrate nella tabella 3.1 per manipolare le liste direttamente. I parametri per le routine sono i seguenti:

- node è un puntatore a un nodo di lista
- list è un puntatore a un'intestazione di lista
- listnode è un puntatore a un nodo
- name è un puntatore a una stringa null-terminated
- start è un puntatore a una nodo o a una intestazione di lista

La ricerca di name comincia dal nodo seguente a quello a cui punta così non include il nodo corrente nella ricerca; pertanto, l'intestazione di lista è un punto perfetto da cui cominciare se lo si desidera.

Si possono creare e mantenere delle liste personali con l'uso di queste routine. Semplicemente si deve inizializzare una intestazione della lista richiamando la funzione NewList, creare la propria lista personalizzata facendo in modo di inserire un nodo di lista nella struttura dati che si vuole utilizzare e manipolare la lista con le routine di sistema.

Funzione	Scopo
AddHead(list,node);	Aggiunge una voce all'inizio della lista
AddTail(list,node);	Aggiunge una voce alla fine della lista
Enqueue(list,node);	aggiunge una voce alla lista in una sequenza di priorità, davanti alla prima voce che si abbia a una priorità inferiore
Insert(list,node, listnode);	inserisce un nodo in una lista davanti a un nodo preesistente nella lista
Remove(node);	~~aggancia~~ un nodo dalla lista alla quale esso è correntemente agganciato
node = RemHead(list);	aggancia il nodo che è correntemente all'inizio della lista e ne ritorna l'indirizzo; ritorna un valore nullo se la lista è vuota
node = RemTails(list);	aggancia il nodo che è correntemente alla fine della lista e ne ritorna l'indirizzo; ritorna un valore nullo se la lista è vuota
node = FindName (start,name);	trova il prossimo nodo della lista corrente che ha lo stesso nome di ciò che è puntato dal campo name della funzione chiamata

Tabella 3.1

Programma che utilizza le funzioni di lista

Il listato 3.1 è un semplice programma che utilizza una lista per creare una coda LIFO (Last-In-First-Out).Non ha importanza dove sia situato il nodo di lista nella struttura dati purché il suo indirizzo di inizio sia fornito alle routine di manipolazione delle liste. Per molte strutture dati usate dal sistema il nodo di lista è la prima voce della struttura stessa (come struct MsgPort, struct Message, e struct Task).

Comunque per i processi (struct Process), per le librerie (struct Library), e per i device (struct Device), il nodo di lista è in una posizione che è diversa dall'indirizzo iniziale. Questo accade perché le regole che governano l'uso di queste strutture dati permettono che un certo tipo di dati possa trovarsi in posizione di offset positivo (a indirizzo superiore) o in una posizione di offset negativo (a indirizzo inferiore) rispetto alla posizione del nodo di lista all'interno della struttura dati.

```
/* list.example.c */

#include "exec/types.h"
#include "exec/lists.h"

struct MyListItem
{
            struct Node n;
            char *data;
};

main()
{
            struct MyListItem mli[3];
            struct MyListItem *mynode;
            struct List MyListHead;
            int i;

            NewList (MyListHead); /* inizia l'intestazione di lista */

            mli[1].data = "primo";
            mli[1].data = "secondo";
            mli[2].data = "terzo";

            for(i=0; i<3; i++)
            {
                  AddTail(MyListHead, &mli[i]);
            }
```

```
            for(i=0; i<3; i++)
            {
                  mynode = (struct MyListHead *)RemTail(MyListHead);
                  printf("\n Ho appena rimosso la voce il cui dato è:
                        %ls",mynode->data);
            }
}                 /* fine del programma */
```

Listato 3.1

Si vedrà questa caratteristica nel discorso sulle librerie che faremo in questo capitolo.

I segnali

I segnali sono il mezzo attraverso il quale l'Exec controlla cosa succede a un task durante la sua esecuzione. Si può dire all'Exec come un task deve reagire quando gli arriva un particolare segnale. La ricezione di un segnale consiste semplicemente nel settare (a 1) un particolare bit all'interno di un valore long (32 bit) che si trova nel blocco di controllo del task. Di solito non sarà necessario analizzare i singoli bit nel blocco di controllo per utilizzare i segnali. Si userà, invece, il valore restituito dall'Exec per vedere cosa è successo.

Cosa può avvenire quando arriva un segnale

Ci sono cose differenti che possono accadere come risultato di segnali ad un task:

- Non accade nulla nel caso il task non sia programmato a reagire a quel particolare segnale.
- Un task che si trovava inoperativo in stato di attesa si risveglia grazie al segnale che lo informa, attraverso il settaggio di un bit, che ciò che attendeva è avvenuto.

- Un task viene forzato a un comportamento particolare, se un codice e dei dati di eccezione sono stati inviati con un particolare segnale.

Si noti che il processo di eccezione è un argomento molto avanzato e non sarà trattato in questo libro.

Il significato dei segnali nel Multitasking

I segnali sono spesso inviati come risultato dell'arrivo di un messaggio da una porta di messaggio che è governata dal task. Vi sono in ogni segnale (che consiste di una variabile long) 16 bit a disposizione del task per segnali da parte dell'utente e 16 bit a disposizione dell'Exec. Se ogni porta di messaggio governata da un task ha assegnato un unico bit di segnale, sarà facile determinare quale porta ha ricevuto il segnale semplicemente identificando la sorgente del segnale stesso. Se si necessita di più di 16 porte di messaggio per un certo task, si potrà spartire il bit di segnale tra molte porte, e testare poi ogni porta, a turno, per vedere se c'è qualche messaggio presente in essa.

Allocazione di un bit di segnale

Il proprio task alloca un bit di segnale per un suo uso attraverso la funzione AllocSignal. Per richiamare AllocSignal si operi così:

```
bitnumber = AllocSignal(number);
```

dove number è un valore da 16 a 31, se si vuole allocare un particolare bit di segnale, o -1 se non ci si preoccupa del numero specifico ma si desiderano semplicemente tutti i bit di segnale che sono disponibili; bitnumber è il valore che l'Exec ritorna come specifico del bit di segnale che ha allocato per uso dell'utente. Se AllocSignal ritorna -1, allora non è stato in grado di eseguire la richiesta fattagli. Si deve controllare il valore restituito per vedere se è un valore valido.

Questo valore di bitnumber è esattamente il valore corretto che è richiesto nella struttura dati per la porta di messaggio (mp_SignalBit) che specifica quale bit deve essere settato quando un messaggio arriva alla porta.

Ecco un esempio dell'uso di AllocSignal:

```
int signalbit;
signalbit = AllocSignal(-1);
/* mi fornisce ogni Bit disponibile */

if(signalbit = = -1)errror();
/* fa qualcosa se non ha funzionato */
```

Utilizzare i bit di segnale nel Multitasking

Per aumentare l'efficenza nel Multitasking, l'Exec scoraggia i loop di attesa. Per esempio, se si vuole far attendere una funzione, si deve settare il timer di sistema da qualche parte, poi porre il proprio task in stato di attesa finché il timer non da il segnale di sblocco. Se si attende un messaggio da un altro task o si attende che un'operazione di I/O termini, non bisogna porsi in un loop che attende finché non sarà settato un bit o arriverà un messaggio. Questi loop sono delle perdite di tempo e possono rallentare inutilmente il sistema, impedendo che altri task con operazioni importanti da compiere possano girare.

Il metodo preferibile è allocare un bit di segnale in relazione a ciò che il proprio task sta attendendo e di richiamare la funzione di sistema Wait specificando il bit, o i bit, di cui si è in attesa. Per esempio si potrebbe allocare un bit di segnale per segnalare l'arrivo di un messaggio; un altro bit potrebbe indicare che è trascorso un certo tempo e il timer ha segnalato ciò, e così via. Questo libera il processore a favore degli altri task. Il proprio task sarà reso libero di girare di nuovo quando uno o più bit di segnale saranno stati settati. Il sistema dice quali bit sono stati settati come valore di ritorno dalla funzione Wait, così si può saper cosa fare successivamente. Per richiamare la funzione wait si opera così:

```
wakeupmask = Wait(bitpattern);
```

dove bitpattern è l'OR logico dei bit di segnale che il task sta attendendo. Il settaggio di uno o più di questi bit con il significato di segnalazione per il task, permetterà a questo di girare nuovamente. Il valore restituito, wakeupmask, è l'OR logico di tutti i segnali che sarebbero serviti per svegliare il task.

Si noti che il task deve rispondere a tutti bit contenuti in wakeupmask, poiché questa è l'unica volta in cui viene segnalato che i bit sono stati settati. Quando il proprio task si sveglia, i bit che sono riportati in wakeupmask come settati, ormai non lo sono più. Così se si attende che avvengano due eventi, ognuno dei quali è associato a un unico bit di segnale, può accadere che entrambi avvengano nello stesso periodo di attesa. Si deve pertanto rispondere a tutti i bit, altrimenti rispondendo a uno solo e riponendo il task nuovamente in stato di attesa (con un altro Wait) lo si può mettere nella posizione di aspettare qualcosa che è già avvenuto.

Settare direttamente un bit di segnale

C'è una funzione che permette di settare uno o più bit di segnale, ma è usata raramente. Normalmente le segnalazioni sono settate automaticamente da cose come messaggi che arrivano a una porta di messaggio. D'altronde, è possibile usare un segnale per permettere a un task di informarne un altro che ha completato l'inizializzazione di una parte di memoria o che ha completato dei calcoli particolari e che vi è il risultato pronto in un'area di memoria accessibile a entrambi. La funzione è chiamata Signal. Per richiamarla si scriva

```
Signal(task,signalmask);
```

dove task è un puntatore al blocco di controllo del task che deve ricevere il segnale, e signalmask e una maschera che contiene i bit che devono essere settati.

Utilizzare bit multipli di segnale

Il listato 3.2 è un esempio di attesa di un segnale multiplo. Si noti che AllocSignal ritorna un valore intero. Per convertire questo valore intero nel reale numero di bit, si sposta un bit con valore 1 a sinistra del numero di posizioni del numero di segnale. Per attendere che avvenga un certo segnale, si opera un Or logico dei bit di segnale con una maschera singola a 32 bit. Per testare i singoli segnali, si opera un AND logico tra il numeri di bit e la maschera wakeup.

Il listato assume che si sia in attesa della pressione di un tasto o della ricezione di un carattere dalla porta seriale.

Si noti che il codice che testa la maschera per ognuno dei due segnali indipendentemente, deve essere posizionato in modo da assicurare che, se entrambi i segnali avvengono, entrambi gli eventi vengano notati. In più, bisogna assicurarsi che nel caso avvengano segnalazioni multiple prima che il task si svegli, si processano tutte le cose che possono aver causato il settaggio del bit di segnale.

```
#define KEYSTRIKESIGNAL ( 1 < sigkey )
#define SERIALSIGNAL (1 < sigser)

int sigkey, sigser, wakeupmask;

sigkey = AllocSignal(-1);
if(sigkey == -1)
      error();

sigser = AllocSignal(-1);
if(sigser == -1)
      error();

/* Questo potrebbe essere un codice per inizializzare /
/* una porta di messaggio e una struttura dati Message */
/* per comunicare con la tastiera e assegnare il signalkey */
/* a quella porta; oppure il codice per settare una porta e un */
/* messaggio per comunicazioni con device seriali */
/* assegnando signalser alla porta per la comunicazione seriale /

/* Attende che una o l'altra segnalazione */
/* si verifichi operando un OR logico sui Bit */

wakeupmask = Wait(KEYSTRIKESIGNAL | SERIALSIGNAL);
if(wakeupmask & KEYSTRIKESIGNAL)
{
/* codice per elaborare il tasto premuto */
}
if(wakeupmask & SERIALSIGNAL)
{
/* codice per elaborare il carattere seriale ricevuto */
}
```

Listato 3.2

Porte di messaggio

Una porta di messaggio (MsgPort) è una struttura dati settata nella memoria come il luogo nel quale il task può inviare un messaggio. Un messaggio è un blocco di memoria situato nello spazio di memoria del task. Esso contiene informazioni di cui necessitano altri task.

Una porta di solito appartiene a un task particolare. Quando un messaggio arriva a una porta, possono aver luogo parecchie cose. Le possibili azioni sono indicate dai valori dei flag di porta, chiamati mp_Flags, nella struttura dati della MsgPort:

PA_IGNORE	quando arriva il messaggio
PA_SIGNAL	segnala al task che governa quella porta che il messaggio è arrivato
PA_SOFTINT	causa un interrupt software del task che governa quella porta

Se il messaggio viene ignorato, esso è semplicemente aggiunto alla lista di messaggi che la porta mantiene. Poi, il task può utilizzare la funzione GetMsg(msgport) per rimuoverlo.

Se invece un messaggio deve essere segnalato al task, è allora possibile che quando esso arriva un task che era in stato di inattività si risvegli grazie ad esso.

Se un messaggio deve causare un interrupt software, significa che qualunque cosa il task stia facendo sarà sospesa per eseguire un set speciale di codici di Interrupt. Questo potrebbe accadere, per esempio, in un programma di terminale dove è importante recuperare ogni carattere che arriva attraverso la linea seriale nonostante il task stia finendo di eseguire un'elaborazione dei caratteri da tastiera.

Creare e distruggere una porta di messaggio

La funzione di libreria di Amiga chiamata CreatePort alloca la memoria per una porta di messaggio e inizializza per l'utente vari campi della struttura dati MsgPort. La porta di messaggio ha una intestazione di lista come parte della struttura dati all'interno della quale le liste di messaggi possono essere aggiunte come risultato della chiamata della funzione PutMsg.

CreatePort richiama la funzione NewList per inizializzare correttamente l'intestazione di lista. CreatePort alloca la memoria e i bit di segnale. Se CreatePort ha dei problemi o con la memoria (potrebbe non trovare abbastanza spazio) o con i bit di segnale (non ci sono più bit di segnale disponibili), essa restituisce un valore nullo. Così si deve controllare per sapere se una porta è effettivamente stata creata. Si può restituire la memoria al sistema e disallocare i bit di segnale attraverso l'uso della funzione DeletePort.

CreatePort inizializza i mp_Flags a PA_SIGNAL e rende il task che la richiama il possessore della porta inizializzando il valore di mp_Task perché punti al task richiamante. Così, il task che crea la porta è quello al quale verrà segnalato che i messaggi sono arrivati a destinazione. Per richiamare CreatePort si scriva:

```
mp = CreatePort(name,priority);
```

dove name è un puntatore a una stringa null-terminated che sarà il nome della porta di messaggio e priority è il valore da dare al campo priorità della porta di messaggio.

Se campo name non contiene un valore nullo, questa porta è aggiunta alla lista delle porte di messaggio di sistema, utilizzando la funzione AddPort. Questo permette agli altri task di localizzare la porta attraverso il suo nome, per mezzo della funzione FindPort. Se il campo name contiene un valore nullo, AddPort non viene chiamata. Si noti che non è necessario aggiungere una porta al sistema se non si deve ritrovare la porta attraverso il suo nome.

Se, per esempio, si hanno due porte con lo stesso nome, si può posizionare l'una davanti all'altra nella lista delle porte di messaggio del sistema settando un valore di priorità maggiore per una e minore per l'altra.

Quando si utilizza la funzione FindPort, nonostante abbiano lo stesso nome sarà trovata quella a priorità maggiore.

Per richiamare la funzione DeletePort si opera così:

```
DeletePort(mp);
```

dove mp è un puntatore alla porta di messaggio ottenuto chiamando la funzione CreatePort.

Se la porta contiene una lista linkata di messaggi, si devono prelevare tutti i messaggi che possono essere stati aggiunti a essa e poi rispondervi prima di cancellarla. Altrimenti, si potrebbe bloccare un task che ha inviato un messaggio e sta attendendo una risposta prima di riprendere il suo compito.

Aggiungere e rimuovere le porte di messaggio

Si aggiunge una porta di messaggio inizializzata alla lista delle porte di sistema usando la funzione AddPort. Si rimuove una porta di messaggio dal sistema usando la funzione RemPort.Per richiamare AddPort si scriva:

```
AddPort(mp);
```

Per richiamare RemPort si scriva:

```
RemPort(mp);
```

In entrambi i casi mp è il puntatore a una porta di messaggio completamente inizializzata. Il principale campo che deve essere inzializzato è il campo priorità, così che l'Exec sappia dove porre la porta all'interno della lista di sistema. Si deve anche inizializzare il campo name in modo che la porta sia anche accessibile attraverso il suo nome. Si noti anche che se la porta dovrà essere in grado di ricevere messaggi, l'intestazione della lista di messaggi dovrà essere inizializzata correttamente.

Si noti che la funzione CreatePort richiama automaticamente AddPort se si è provveduto a un nome attraverso il quale identificare la porta. CreatePort inizializza adeguatamente tutte le altre parti della struttura dati MsgPort. DeletePort usa RemPort automaticamente. Cosi, utilizzando queste funzioni di supporto dell'Exec, non bisogna preoccuparsi di usare di AddPort e RemPort o qualsiasi altra inizializzazione.

Ritrovare una porta di messaggio

Si può localizzare una porta di messaggio sulla lista di sistema delle porte di messaggio utilizzando la funzione FindPort. Per richiamarla si opera così:

```
foundport = FindPort(pointer_to_namestring);
```

dove pointer_to_namestring è un puntatore a una stringa di caratteri null-terminated rappresentante il nome della porta che deve essere trovata. Viene restituito un valore nullo se non vi è una porta con quel nome sulla lista di sistema delle porte di messaggio.

Si noti che quando una porta viene aggiunta alla lista, il campo priorità ne determina la posizione all'interno della lista. Se vi è più di una porta sulla lista di sistema avente lo stesso nome, allora la funzione FindPort trova solo quella che ha una priorità maggiore nel campo ln_Pri del campo mp_Node della struttura dati MsgPort.

Se si ha il sospetto che vi possa essere più di una porta con lo stesso nome, si può utilizzare la funzione FindName, per localizzare qualunque porta a priorità inferiore all'interno della lista. In questo caso, per richiamare FindName si scriva ciò che segue:

```
struct MsgPort *mp, *mpmore;

mp = FindPort("myport");

if(mp)
   /* se non è nullo (e si sospetta vi siano altre porte */
   /* con questo nome ) */
{
```

```
    mpmore = FindName(mp,"myport");

    if(mpmore)printf("Ne ho trovata un altra con lo stesso nome");
}
```

Frammenti di codice per l'uso delle porte di messaggio

Il listato 3.3 mostra come si alloca una porta di messaggio, come una maschera di segnale è formata e usata, e come la porta viene cancellata.

Cercare delle segnalazioni provenienti dalla porta di messaggio

Se un particolare evento deve settare un certo bit per avvertire un task che esso è avvenuto, e il task non risponde immediatamente, è possibile che avvenga un evento diverso che però setti lo stesso bit di segnalazione. Spesso questo capita, per esempio, quando un nuovo messaggio è arrivato a una porta ed è stato aggiunto alla lista di messaggi contenuta nella porta stessa

---

```
struct MsgPort *mp;                 /* puntatore a una porta di messaggio */
   struct MsgPort *foundport;       /* un'altro puntatore */

   int signalbit, signalmask;

   mp = CreatePort("myport",0)      /* nome della porta e priorità (=0) */

   signalbit = mp->mp_SigBit;       /* trova il Bit di segnalazione
                                       da essa allocato */

   signalmask = (1 < signalbit);    /* usa il numero del Bit
                                       per creare un waitmask */

   /* ora se vogliamo attendere un messaggio*/

   wakeupmask = Wait(signalmask);

   foundport = FindPort("myport");  /* mostra come un altro Task può */
                                    /* localizzare la mia porta
                                       di messaggio */
                                    /* in modo da potermi inviare
```

```
                                    messaggi */
                                    /* Per questi passi, il valore di
                                     foundport */
                                    /* deve essere lo stesso di mp */

if(foundport == 0)
{
        printf("non trovo 'myport'");
)
/* e una volta finito tutto, dopo aver anche risposto ai messaggi */

DeletPort(mp);                      /* disalloca la memoria e i Bit
                                     di segnalazione */
```

Listato 3.3

Un task non può porsi in stato di inattività in attesa di un messaggio e, dopo aver elaborato un messaggio solo, tornare in stato di inattività. Poiché vi è un solo bit di segnale che può essere assegnato a una porta di messaggio per segnalare l'arrivo del messaggio, non è possibile avere i segnali separati dell'arrivo di ogni singolo messaggio. Per questo un task deve assicurarsi di rispondere a tutti i messaggi immagazzinati nella porta di messaggio dopo aver ricevuto una segnalazione riguardante quella porta, poiché un messaggio che si trovi già li non può più risettare il bit di segnalazione per risvegliare il task. In altre parole, i messaggi si accodano in memoria, i bit no.

I messaggi

Un messaggio è semplicemente un blocco di memoria che è utilizzata per passare dati da un task all'altro. Il blocco di memoria appartiene al task che crea il messaggio.

Il processo per impostare un messaggio, all'interno della porta di messaggio, consiste nel linkarlo alla lista di messaggi che è mantenuta nella porta stessa. Le funzioni dell'Exec non copiano il messaggio, ma semplicemente settano dei puntatori che puntano al blocco di memoria in modo che il task che deve ricevere il messaggio sa dove trovare le informazioni in esso contenute.

Perché usare i messaggi?

L'uso dei messaggi è fortemente presente nel sistema di Amiga per inizializzare l'attività di input-output. Per esempio ci sono dei task che manipolano la porta parallela e quella seriale, la tastiera, la porta giochi, e il disco. Il processo per inizializzare l'I/O nel sistema consiste nel formulare un insieme di messaggi che specificano l'attività di I/O che deve avvenire e nel passare questi messaggi ai task che manipolano l'hardware specifico.

Il task ricevente è probabilmente in stato di inattività mentre attende un messaggio che gli dica cosa deve fare. Quando un task richiede qualche attività di I/O, esso potrebbe mettersi in stato di inattività attendendo i dati che gli devono essere riinviati. Quando il task di I/O avrà completato la richiesta, restituirà un messaggio al task richiedente, indicando che i dati sono ora disponibili o segnalando una condizione di errore. Così il task di I/O può ritornare inattivo, riattivando il task richiedente restituendogli il messaggio.

I contenuti di un messaggio

La struttura di un messaggio è molto semplice:

```
struct Message
{
struct Node mn_Node;
/* un nodo di lista */
struct MsgPort *mn_ReplyPort;
/*un puntatore a una porta di messaggio */

int mn_Length;
/* la lunghezza del messaggio*/
};
```

Il mn_Node è un semplice nodo di lista, usato per linkare un messaggio dentro la lista. Il mn_Node.mn_Type deve essere NT_MESSAGE, cioè come dire "questo nodo è di tipo Message". Il mn_ReplyPort è un puntatore alla porta di messaggio alla quale questo messaggio deve essere restituito quando il task ricevente ha finito di usarlo. Un messaggio ha origine da un certo task, e probabilmente deve essere passato ad un altro. Questo puntatore alla porta di risposta identifica l'inviante e, pertanto, il task al quale deve essere restituito.

Quando il messaggio viene restituito il mn_Node.ln_Type viene cambiato dal sistema in NT_REPLYMSG.

Spesso, un task che genera un messaggio, costruisce anche una porta di messaggio che sarà usata come luogo di ricezione del messaggio, poi designata come la porta di risposta dell'inviante. Il task inviante può decidere di inviare un messaggio (usando funzioni come PutMsg), e poi mettersi in stato di inattività aspettando che il messaggio venga nuovamente inviato alla sua porta di risposta.

Il task ricevente potrebbe essere stato inattivo in attesa dell'arrivo del messaggio che deve entrare nella sua porta di messaggio. Si può quindi svegliare, recuperare il messaggio (usando delle funzioni come GetMsg), e restituire il messaggio a chi lo ha originato (usando la funzione ReplyMsg) una volta che ha finito di utilizzare i contenuti del messaggio stesso.

La lunghezza del messaggio, rappresentata da mn_Length, non è utilizzata dalle routine di sistema che passano il messaggio. Si può assegnare a questo valore un significato che si desidera.

Il significato dei messaggi

Il passaggio di messaggi è effettuato attraverso dei riferimenti non attraverso una copiatura. Quando un task invia un messaggio ad un altro task, glielo passa attraverso l'indirizzo della struttura dati all'interno del suo spazio di memoria. Il task inviante da temporaneamente in custodia quella parte di memoria ad un altro task.

L'altro task può copiare il contenuto dello spazio all'interno della propria memoria (si veda il quinto capitolo), oppure può scrivere qualcosa dentro questo spazio, o può fare qualsiasi cosa. Quando l'altro task ha finito di usare il messaggio, esso deve essere restituito al task di origine attraverso l'uso della funzione ReplyMsg.

E' particolarmente significativo che tutti i messaggi siano restituiti il più velocemente possibile. In particolare, il task originario potrebbe essere in attesa del ritorno del messaggio sulla sua porta di risposta. Inoltre, di solito, il task originario possiede direttamente la memoria nella quale risiedono i dati del il messaggio.

L'Exec tiene una traccia solo della memoria che non è allocata a favore dei task, e ogni task può tenere una traccia della memoria che esso stesso ha allocato e restituirla all'Exec quando termina. Così, se un task non risponde ai messaggi che riceve, questo task non cooperante potrebbe impedire ad altri task di girare, o potrebbe impedire la restituzione di svariate parti di memoria al sistema.

E' quindi necessario che i messaggi che vengono inviati dai quali ci si aspetta una risposta ritornino effettivamente una risposta in modo da permettere al sistema di operare al meglio.

I messaggi personalizzati

Alcune delle strutture dati del sistema sono di fatto una versione personalizzata di messaggi che vengono usati per permettere a un task di passare informazioni a un altro speciale task di I/O. Le strutture dati di nome IORequest e IOStdReq sono un esempio di questa personalizzazione della struttura Message. Un messaggio personalizzato può essere settato nel seguente modo:

```
struct myCustomMessage
{
   struct Message mcm_Message;
   int item1;
   int item2;
   int item3;

};
```

Qui si vede la struttura di un messaggio standard con tre valori interi ad esso allegati. Questo messaggio personalizzato può essere manipolato utilizzando una qualsiasi delle routine di sistema atte a questo scopo.

Funzioni per la manipolazione dei Messaggi e delle porte di messaggio

La tabella 3.2 contiene una lista che si riferisce ai messaggi e alle porte di messaggio. I parametri sono i seguenti:

- msgport è un puntatore a una porta di messaggio
- msg è un puntatore a un messaggio
- priority è il valore di un byte che va da -128 a +127
- name è un puntatore al primo carattere di una stringa null-terminated

Programma che utilizza i messaggi e le porte di messaggio

Il listato 3.4 mostra il passaggio di messaggi attraverso le porte. Questo di solito è usato come forma di comunicazione tra task; comunque, questo semplice esempio ha il solo scopo di illustrare il processo. Nel nono capitolo si mostrerà come passare messaggi da un task agli altri.
In questo listato il campo name del nodo di messaggio è usato come luogo nel quale il messaggio è stato passato. Molti messaggi intertask possono non essere di tipo string-based, ma possono invece essere un pacchetto di informazioni allegato all'inizio o alla fine di una struttura dati Message. Ciò che qui ha importanza è illustrare la natura generale di queste routine.

Funzione	Scopo
AddPort(msgport);	Aggiunge una porta di messaggio preinizializzata alla lista lista di sistema delle porte di messaggio.
RemPort(msgport);	Rimuove una porta di messaggio dalla lista di sistema.
FindPort(name);	Trova la prima porta di messaggio con questo nome all'interno della lista di sistema.

msg = WaitPort(msgPort);	Pone un task in stato di inattività in attesa dell'arrivo di un messaggio su una porta di messaggio. Ha un effetto simile a quello della funzione Wait (1 < msgport -> mp_SigBit). Punta al primo messaggio che è arrivato alla porta, ma non rimuove il messaggio dalla porta. Bisogna utilizzare anche GetMsg per rimuoverlo.
PutMsg (msgport,msg);	Invia un messaggio a una porta di messaggio.
msg = GetMsg(msgport);	Se vi è un messaggio presente su quella porta restituisce il suo valore e lo disgiunge dalla lista dei messaggi di cui la porta è in possesso. Se non vi sono messaggi restituisce un valore nullo.
ReplyMsg(msg);	Risponde ad un messaggio trasmettendo il messaggio (mediante PutMsg) alla porta di risposta specificata nel campo ReplyPort del messaggio stesso.
msg port= CreatePort (name,priority);	Alloca la memoria per una porta di messaggio ritorna un puntatore da essa. Se il campo name non è zero, aggiunge questa porta alla lista di sistema delle porte di messaggio (usando AddPort). Pone la porta all'interno della lista secondo il valore di priority. Così è possibile posizionare questa porta all'inizio della lista davanti a una porta che è già nella lista stessa. FindPort troverà la porta di un dato nome avente priorità maggiore nella lista.
DeletePort (msgport);	Cancella una porta che è stata creata con CreatePort, se la porta ha un nome, esso sarà cancellato dalla lista di sistema delle porte di messaggio

Tabella 3.2

```
#include "exec/types.h"
#include "exec/ports.h"
main()
{
      struct Message *m;            /* il messaggio che passerà */
      struct Message *msg;
                                    /* un puntatore al messaggio che
                                     sarà restituito *
      struct Message *GetMsg();
```

```
struct MsgPort *mp;          /* un puntatore ad porta di messaggio */
struct MsgPort rp            /* un puntatore ad un'altra porta
                              di messaggio */
                             /* che opererà da porta di risposta */

struct MsgPort *CreatePort();
extern

mp = CreatePort(0,0);        /* notare che NON è necessario
                              dare un nome */
                             /* alla porta per poi poterle inviare
                              un messaggio */
                             /* è necessario darle un nome solo se
                              più tardi */
                             /* si vuole usare FindPort */
if(mp == 0) exit 20;         /* error in CreatEport() */

rp = CreatePort("reply",0);  /* la chiama reply dandole
                              una priorità 0 */

if(rp == 0)
{
            DeletePort(mp); exit 30;
}

m.mn_Node.ln_Name = "Hello world\n";
m.mn_ReplyPort = rp;         /* definisce la porta di risposta */
m.mn_Length = 0;             /* in questo caso la lunghezza è nulla */

PutMsg(mp,&m);               /* invia il messaggio a questa porta */
WaitPort(mp);                /* attende che questo arrivi */

/* finché il messaggio è la, il task non diventa inattivo */

/* se noi arriviamo qui, sappiamo che vi è realmente */
/* un messaggio presente */

while(msg = GetMsg(mp))
{
/* preleva tutti i messaggi da questa porta prima di */
/* tornare alla WaitPort o di distruggere questa porta */

            printf("il messaggio era: %ls\n",
                              msg->mn_Node.ln_Name);
            ReplyMsg(msg);
}

WaitPort(rp);                /* attende alla porta di risposta
                              il ritorno */
                             /* del messaggio originariamente
                              inviato... */
                             /* non richiamerò RepyMsg perché
                              io stesso */
                             /* ho inviato il messaggio */
```

```
    while msg = GetMsg(rp))

            printf("La porta di risposta ha ricevuto questo
            messaggio: %ls\n",msg->mn_Node.ln_Name);
    }
    DeletePort(mp);
    DeletePort(rp);
}
```

Listato 3.4

Le librerie

Una libreria è un insieme di routine correlate. Quando una libreria di routine viene inserita da qualche parte all'interno di Amiga tutti i task possono accedervi. Questo permette di scrivere programmi più corti perché non è necessario inserirvi le istruzioni riguardanti le funzioni più comunemente richiesta. Per esempio, la libreria dell'Exec contiene tutte le principali routine correlate all'Exec come quelle per il trattamento delle liste, il controllo dei task, i device di I/O, il passaggio dei messaggi, e così via; la libreria del Dos contiene le funzioni ad esso correlate; e la graphics.library contiene le funzioni inerenti la grafica.

La struttura di una libreria

Una libreria è di fatto una struttura dati e può essere linkata all'interno della lista delle Library di sistema. Questa struttura dati contiene un nodo di lista, un set di vettori delle funzioni della libreria, e un'area dati. La struttura di una libreria è illustrata in figura 3.2.

Il nodo di Library contiene informazioni sulla grandezza della libreria e le variabili che la controllano. Ecco una lista del contenuto del nodo di Library:

```
struct Library
{
   struct Node lib_Node;
   UBYTE lib_Flags;
   UBYTE lib_pad;
   UWORD lib_NegSize;
   UWORD lib_Version;
```

```
    UWORD lib_Revision;
    APTR lib_IdString;
    ULONG lib_Sun;
    UWORD lib_OpenCnt;
};
```

E' importante che si capisca come vengono usati i campi nel nodo di libreria.

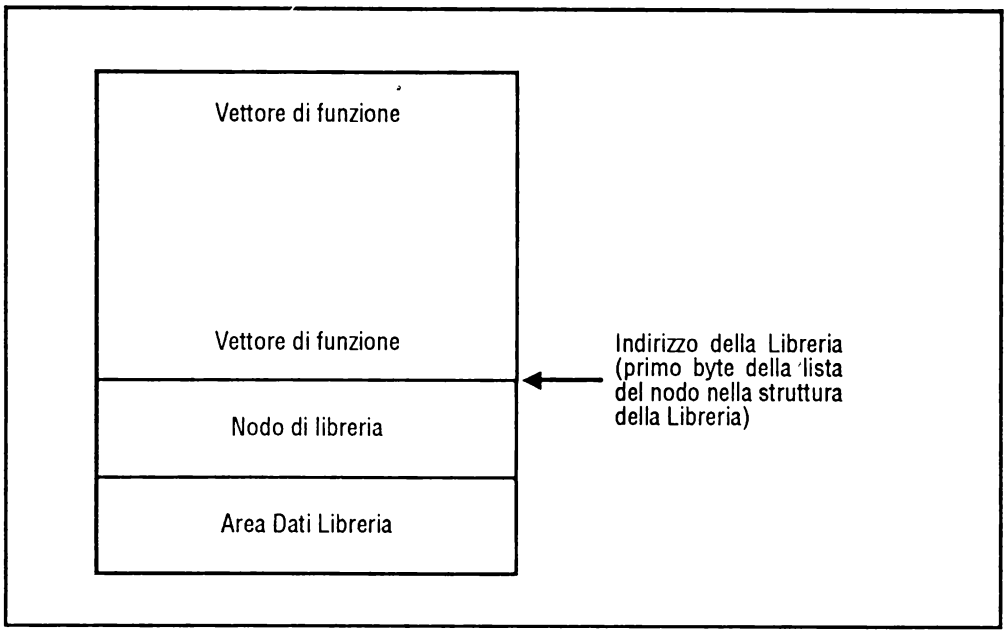


Fig. 3.2 La struttura di una libreria

Il campo lib_Flags. Questo campo è usato dall'Exec per tenere una traccia di ciò che accade alla libreria: se qualcuno ha cambiato un vettore della libreria; se la libreria sta subendo un Checksum; se vi è qualcosa che va cancellato. Un programmatore che scriva i suoi programmi in Assembler e voglia farli girare ricorrendo a un processo basato sugli interrupt di entrata del 68000 potrebbe aver bisogno di esaminare i bit all'interno di questi Flag, ma il programmatore che usi un linguaggio ad alto livello non avrà alcun bisogno di controllare questi Bit.

Ecco uno stralcio di programma che mostra cosa può succedere ai vari Flag della libreria. Si assume che il task che ha correntemente il controllo abbia bisogno di usare una routine contenuta nella libreria e voglia sapere se questa routine è ancora valida:

```
struct Library lib; /* inizializzata da qualche altra parte */

if(lib->lib_Flags & LIB_CHANGED)
{
   /* potrebbe non essere valida se è cambiata */
}
if(lib->lib_Flags LIB_SUMMING)
{
   /* il checksum non è valido; la Library non è */
   /* completamente stabile, forse non la si può ancora usare */
}
if(lib->lib_Flags & LIB_DELEXP)
{
   /* il sistema è a corto di memoria e vuole sbarazzarsi */
   /* di questa Library al più presto */
   /* quando il task che la sta utilizzando terminerà /
   /* essa sarà estromessa */
}

if(lib->lib_Flags LIB_SUMUSED)
{
   /* informa solo... alcuni task hanno settato questo */
   /* flag per impedire all'Exec di usare l'overhead */
   /* che servirebbe per generare e/o */
   /* controllare il checksum */
}
```

Il campo lib_Pad. Questo campo non è utilizzato. Definire questo campo assicura solo un allineamento dei campi che lo seguono sui limiti della Word (di lunghezza 16 Bit).

Il campo lib_NegSize. Questo campo dice quanti Bytes di dati associati alla libreria precedono il nodo stesso della libreria. Questi dati consistono di solito in vettori di funzioni, ognuno dei quali contiene sei Byte, ognuno con una forma di tipo Assembler in cui una istruzione di Jump lunga due Byte è seguita da un indirizzo lungo quattro Byte che serve per l'istruzione Jump stessa.

Il campo lib_PoSize. Questo campo dice quanti Byte di dati associati alla libreria vengono dopo il nodo stesso della libreria. Questa è quell'area della libreria che di solito è usata per contenere le variabili global utilizzate dalle routine interne, o magari potrebbe contenere le stesse funzioni di libreria alle quali fanno riferimento i vettori.

Il campi lib_Version e lib_Revision. Questi campi contengono dei numeri che identificano unicamente la versione e la Revision corrente della libreria. Quando si richiede di aprire una libreria, si può avere sul proprio dischetto, nella

directory LIB:, una versione particolare della libreria richiesta che si vuole utilizzare al posto di quella che è già stata linkata automaticamente nel sistema dalle routine del Kickstart. Specificando una versione differente da quella già presente si è sicuri di utilizzare la propria versione di certe routine contenute nella propria libreria.

Il campo lib_idString. Questo campo contiene il nome della libreria così come potrebbe essere mostrato da un debugger (come WACK). Le Library hanno di fatto due nomi ad esse associati: il nome col quale sono riconosciute dalla funzione OpenLibrary (graphics.library, layers.library, o diskfont.library) e magari un nome più lungo attraverso il quale possono essere identificate più specificatamente. Il campo lib_IdString può infatti essere lungo sino a 256 caratteri se lo si desidera.

Il campo lib_Sum. Questo è il checksum dei vettori delle funzioni della libreria. Una volta che la libreria è stata aggiunta al sistema, l'Exec ne controlla l'integrità, facendo un checksum dei vettori delle sue funzioni. Se i vettori sono cambiati, viene calcolato un nuovo checksum. Il valore dei checksum è a esclusivo uso dell'Exec.

Il campo lib_OpenCnt. Questo campo contiene il numero di volte che è stata usata la funzione OpenLibrary per quella libreria particolare. Quando il sistema è a corto di memoria è possibile disfarsi automaticamente di una libreria che non viene usata da tempo da nessun task operativo. Questo restituisce la memoria della libreria al sistema perché possa utilizzarla. Ogni qualvolta OpenLibrary viene richiamata questo valore viene incrementato. Ogni volta che è invece chiamata CloseLibrary esso viene decrementato. Quando il valore arriva a 0, se l'Exec ha bisogno di nuova memoria, le librerie con un lib_OpenCnt uguale a 0 si autoelimineranno (operazione di expunge) dal sistema e restituiranno tutta la memoria allocata al pool della memoria libera.

Aprire una libreria

Prima di poter utilizzare le routine contenute in una libreria bisogna aprire la libreria stessa. Ecco la forma usata per richiamare OpenLibrary:

```
lib_base = OpenLibrary(libraryname,version);
```

dove version è il numero che identifica la versione della libreria che si vuole utilizzare. Lib_base è una variabile puntatore che ha un nome corrispondente alla libreria da aprire, e libraryname è un puntatore a una stringa null-terminated che fa da nome della libreria alla quale si vuole accedere. Se si volesse utilizzare una particolare versione della libreria si specifichi esattamente il numero desiderato. Per esempio, version 31 è il valore assegnato alla "V 1.1" del software di sistema. Se invece si vuole semplicemente utilizzare qualsiasi versione sia disponibile si specifichi un valore nullo.

OpenLibrary ritorna lib_base che è un puntatore all'indirizzo base della struttura dati della libreria. Si noti che questo è un puntatore al primo Byte del nodo della libreria e che ci saranno delle parti della libreria stessa sia prima sia dopo questo indirizzo. Se la funzione OpenLibrary non è in grado di esaudire la richiesta fattagli restituirà un valore nullo.

Per ogni libreria, vi è un nome specifico dato alla variabile che eguaglia quello della libreria-base. La tabella 3.3 fornisce una lista dei nomi delle Library di Amiga, l'indirizzo di base, e il tipo di routine in esse contenute.

La libreria che si vuole aprire e utilizzare può essere nella RAM o risiedere su ROM o sul disco. Se essa non è ancora in memoria l'AmigaDos cercherà la corrente directory LIB: per vedere se esiste una libreria con quel nome.

Indirizzi Base delle Library

Nome della Library	Nome della variabile	Contenuto
clist.library	ClistBase	Routine per la manipolazione delle stringhe e dei caratteri
diskfont.library	DiskfontBase	Routine per i Font su disco
exec.library	ExecBase	Tutte le funzioni dell'Exec
dos.library	DosBase	Funzioni DOS
graphics.library	GfxBase	Funzioni grafiche

icon.library	IconBase	Funzioni per gli oggetti della Workbench
intuition.library	IntuitionBase	Funzioni per l'interfacciamento ad Intuition
layers.library	LayersBase	Funzioni per la sovrapposizione e la creazione delle Window
mathffp.library	MathBase	Funzioni matematiche di base
mathtrans.library	MathTransBase	Funzioni matematiche trascendenti
mathieeedoubbas.library	MathIeeeDoubBasBase	Funzioni matematiche IEEE in precisione Double
timer.library	TimerBase	Aritmetica del Timer
translator.library	TranslateBase	La funzione Translate

Tabella 3.3

Indirizzi base delle librerie e loro nomi

Il motivo per il quale si assegna un nome particolare a ogni puntatore che si ottiene con la chiamata della funzione OpenLibrary sta nel fatto che le librerie di Amiga contengono un codice specifico di interfacciamento per ogni routine in ogni libreria. Questo codice è linkato automaticamente all'interno del programma richiamante al momento del linkaggio; esso preleva il valore corrente delle variabili specifiche e calcola, da questo valore, dove potrà trovare il vettore di Jump per la singola routine. Poi preleva il valore che la routine C dell'utente ha posto nello stack, salva alcuni registri, carica questo valore in un registro speciale, e infine chiama la routine specificata.

Questo processo di salvataggio e riimpostazione dei registri e di richiamo delle routine residenti è completamente spiegato nel Manuale della ROM Kernel di Amiga. Qui è sufficiente sapere che è necessario usare il nome appropriato, che deve essere non nullo, prima di richiamare una funzione della libreria.

Il sistema operativo di Amiga è stato progettato per essere dinamico. Il codice della libreria, infatti, può essere caricato nella memoria o mentre il sistema viene costruito (attraverso Kickstart) o con una call della funzione OpenLibrary per un tratto della libreria che sia residente su disco. La funzione OpenLibrary può caricare e inizializzare una libreria residente su disco e linkarla al resto del sistema perché la utilizzino il task richiamante e anche tutti gli altri task. Quando una libreria viene richiamata da disco, il codice che la compone viene reso rilocabile dal linker e viene caricato in memoria pezzo per pezzo, con tutti vettori di indirizzo correttamente settati per indicare esattamente dove si trovano le routine.

Così quando si carica una libreria, la posizione della sua Library-Base non cambia; si può memorizzare la variabile Library-Base e essere sicuri che essa rimanga valida finché la libreria è aperta. Nonostante l'indirizzo base della libreria non cambi finché essa è residente in memoria, è possibile che i vettori delle funzioni all'interno della libreria cambino. Alcuni programmatori possono essere tentati, una volta trovata la libreria, di memorizzare i suoi vettori per render più veloce l'accesso alle sue funzioni. Poiché le Library di Amiga possiedono una funzione che permette al task richiamante di variare in valori dei vettori delle sue funzioni, è sempre meglio passare attraverso il codice standard di interfacciamento con la libreria per essere sicuri di usare la versione corrente della funzione di libreria.

Utilizzare le funzioni delle librerie

Una volta che la libreria è aperta, si può richiamare qualsiasi funzione che sia all'interno della libreria. Questo è molto facile da fare con il linguaggio C perché è necessario solo precisare il nome della funzione e i suoi parametri:

```
result = Function(parametro1,parametro2);
```

Il resto della sequenza di chiamata è svolto automaticamente dalla libreria di Amiga.Per i programmatori in Assembler, c'è un po' più di lavoro da fare, nonostante sia semplificato dalle macro del linguaggio stesso che si possono trovare in un file chiamato exec.libraries.i. Per prima cosa bisogna essere sicuri di aver salvato sullo stack tutti i registri appropriati e di aver caricato tutti i valori richiesti nei registri dalla routine. Poi se il valore base della libreria è già nel registro A6, si usi la macro CALL_LIB:

```
CALL_LIB_LVOFunction
```

Se il valore di A6 non è corretto, si usi la macro LINK_LIB

```
LINK_LIB_libraryBase_LVOFunction
```

Qui, _LibraryBase contiene l'indirizzo base della libreria nella quale risiedono i vettori delle funzioni, e _LVOFunction è il Library Vector Offset (LVO). Questo è il valore negativo che va aggiunto all'indirizzo della Library-Base per creare l'indirizzo del quale viene dato l'accesso ai vettori di jump.

Chiudere una libreria

Una volta finito di usare le routine della libreria, si deve chiudere la libreria per terminare l'accesso ad essa. Se si è l'ultimo utilizzatore della libreria, è possibile che la libreria si autoelimini dal sistema e restituisca la memoria che aveva utilizzato. Si termina l'acceso a una libreria richiamando la funzione CloseLibrary. Per richiamare tale funzione si scriva:

```
CloseLibrary(libBase);
```

dove libBase è l'indirizzo base della libreria.

Programma per aprire, usare, e chiudere una libreria

Il listato 3.5 è un programma che apre una libreria, ne utilizza una funzione, e infine la chiude.
Si noti che se si usano i file standard di Startup dal C - che sono Astartup.asm (che produce Astartup.obi) o Lstartup.asm (che produce Lstartup.obj) - non è necessario OpenLibrary per ottenere il valore di DosBase. I file di Startup fanno questo per l'utente. Così, nel proprio programma si è liberi di usare tutte le routine di dos.library senza aprire questa libreria.

```
#include "exec/types.h"
#include "intuition/intuition.h"
#include "intuition/intuitionbase.h"
extern struct Library *OpenLibrary();
long IntuitionBase *IntuitionBase;
main()
{

/* APRE la libreria */

      IntuitionBase = (struct IntuitionBase*) OpenLibrary("intuition.libra-
ry",0);

      if(IntuitionBase == 0)
      {
            printf("non posso aprire l'Intuition\n");
      }
      else
      {
            printf("Ok aperta l'intuition\n");

/* utilizza una delle routine */

            DisplayBeep(NULL); /* opera un Flash sul display */



/* CHIUDE la Library */

            CloseLibrary(IntuitionBase);
      }
}
```

Listato 3.5

I device

Device, su Amiga, è il nome dato a una particolare struttura dati per mezzo della quale si accede a una certa entità di input-output (come la porta seriale, la porta parallela, una console, o la traccia di un disco). I device sono qualcosa al di sopra delle librerie; infatti, esse possono essere costruite dal programmatore per mezzo di varie istruzioni di librerie. Vi sono routine per aprire, chiudere, ed estromettere i device, così come vi sono routine simili per le librerie.

Come per le librerie, si può fare riferimento al device per mezzo di un nome come trackdisk.device. E come una libreria un device può risiedere in memoria o sul disco. Quando viene usata la funzione OpenDevice, se il device non viene trovato in memoria, l'Exec cercherà nella directory corrente dell'AmigaDos assegnata come DEVS: per vedere se c'è un device con quel nome sul disco. L'Exec poi lo carica e lo inizializza e infine lo aggiunge alla lista dei device di sistema.

I device contengono delle unità, che sono le parti operative dei device stessi. Per esempio, il trackdisk.device può avere parecchie unità, ognuna delle quali controlla uno dei disk driver reali attaccati al sistema; il device timer ha due unità, uno per ogni timer reale che essa controlla, uno dei quali è più preciso dell'altro. Si troveranno altre informazioni sulle unità più avanti nel paragrafo quando si parlerà di come aprire un device.

In aggiunta ai quattro vettori di funzione standard comuni a tutte le librerie (OPEN,CLOSE,EXPUNGE, e RESERVED), i device hanno due vettori standard per le funzioni che sono progettati per essere i punti di entrata nel device. Essi sono i vettori di funzione per BEGINIO e ABORTIO. Nonostante questi due punti di entrata siano disponibili su Amiga nelle funzioni di libreria BeginIO(iorequest) e AbortIO(iorequest), sono considerati come argomenti di alto livello e non sono trattati in questo libro. Si consiglia di consultare il Manuale della ROM Kernel di Amiga per una spiegazione di queste funzioni.

Come avviene una richiesta di I/O

La comunicazione con i device consiste nel formulare un pacchetto di messaggio, detto blocco di IORequest (o I/O Request Block) e nel passarlo a una porta di messaggio controllata da un device. Normalmente vi è un processo indipendente che parte da ogni device. Questo processo è inattivo, in attesa di un messaggio che arrivi alla sua porta di messaggio. Il messaggio gli dice che tipo di I/O trattare e come trattarlo.

L'IORequest Block è una struttura dati Message standard che informa il processo su dove inviare il Request Block (a una porta di risposta) una volta che ha finito di utilizzarlo. Esso identifica anche il tipo di comandi che devono essere eseguiti e i flag che dicono come tali comandi devono essere eseguiti e magari un indirizzo dal quale o al quale devono essere trasferiti i dati.

Comandi per i device

I comandi per i device sono usati per direzionare l'input-output. La tabella 3.4 mostra i comandi di device e i valori (dati nell'io.h) che indicano il numero del comando che il device deve eseguire. Questo è il valore da porre nel campo io_Command nell'IORequest per richiedere al device di eseguire quel particolare comando.

Aprire un device

Si comunica con un device passandogli una forma IORequest. L'effettivo passaggio di questa forma di messaggio è gestito da un set di funzioni comune a tutti i device di input-output; i nomi di queste funzioni sono: DoIO, SendIO, WaitIO, e CheckIO. Queste funzioni accettano il messaggio inviato loro e lo rinviano al device corretto esaminando i campi di dati io_Device e io_Unit all'interno del messaggio. Questi campi sono inizializzati dalla chiamata di OpenDevice. Si prepara l'accesso a un device aprendo il device stesso con la funzione OpenDevice. Per richiamare tale funzione si scriva:

```
error = OpenDevice(DEVICENAME,unit,ioRequest,flags);
```

dove error è uguale a 0 se la chiamata ha avuto successo, è invece diversa da 0 se il device non si è aperto.

DEVICENAME è il nome del device con il quale si vuole comunicare. Una lista dei device che sono comunemente disponibili per essere aperti da OpenDevice si trova in una tabella del paragrafo seguente.

Il parametro unit è un identificatore dell'unità dei device che si vuole utilizzare. Per il timer potrebbe essere, per esempio, UNIT_VBLANK o UNIT_MICROHZ. Ogni device ha il suo metodo di usare questo campo. Per molte device (Keyboard, Input, Console,e Parallel per esempio) questo valore può essere nullo.

Il parametro ioRequest è un puntatore a una struttura dati IORequest di una dimensione appropriata al singolo device che si sta cercando di aprire. L'Exec utilizza le informazioni contenute nella richiesta di OpenDevice per riempire una parte dell'IORequest che si è provveduta per usi futuri. Essenzialmente, OpenDevice è un inizializzatore di un messaggio che, tra le altre cose, specifica l'indirizzo del device stesso e identifica in modo unico l'unità del device. Per un device di timer, per esempio, una timerequest. Per un device serial, si usa una IOExtSer. E' importante passare al device una forma di IORequest della giusta dimensione poiché il device inizializza alcune parti della richiesta. Se gli si passasse una IORequest troppo piccola, il device potrebbe scrivere in un'area di memoria che va oltre quella appropriata, creando effetti disastrosi.

Comandi device	Valore	Scopo
Reset	CMD_RESET	Setta il device ai suoi valori di default, riportando ogni cosa nello stato in cui era quando il device è stato creato
Read	CMD_READ	Legge un dato numero di byte all'interno di una data area di memoria

Write	CMD_WRITE	Scrive un dato numero di byte all'interno di una data area di memoria
Update	CMD_UPDATE	Scrive all'esterno il contenuto dei buffer interni; se dei dati devono essere trattenuti prima di essere scritti su disco, questo comando assicura che il disco eguaglierà il contenuto corrente della memoria
Clear	CMD_CLEAR	Cancella tutti i buffer interni senza eseguire un update; butta via il contenuto corrente della memoria
Stop	CMD_STOP	Blocca l'unità di device; permette al device di accettare l'accodamento di nuovi comandi ma evita che qualsiasi comando accodato venga eseguito
Start	CMD_START	Fa ripartire un device bloccato da Stop
Flush	CMD_FLUSH	Fa abortire tutte le richieste di I/O sia in attesa sia operative; tutte le richieste vengono restituite con un valore che indica un errore di I/O abort

Tabella 3.4

Alcuni device possono avere delle opzioni specifiche abilitate dall'utente in fase di apertura. Le variabili di flag sono riservate proprio a questo uso. Le seguenti istruzioni mostrano come aprire il timer.device:

```
struct timerequest tr:
long error;

error = OpenDevice("timer.device",UNIT_VBLANK,tr,0);

if(error)
{
   printf("Timer open error; value %ld\n",error);
}
```

Nomi dei device normalmente disponibili

La tabella 3.5 mostra i nomi dei device che sono disponibili nel sistema quando avviene il boot e di quelle che possono essere caricate dal disco e aggiunte al sistema come risultato di una OpenDevice. La colonna nomi dei device contiene la stringa null-terminated (DEVICENAME) che si deve porre nella chiamata della funzione OpenDevice per richiamare quella Device particolare.

La struttura di un tipico blocco di IORequest

Ecco la forma comune della struttura dati IORequest; essa è chiamata richiesta standard di IO, o IOStdReq:

```
struct IOStdReq
{
   struct Message io_Message;
   struct Device * io_Device;
   struct Unit *io_Unit;
   UWORD io_Command;
   UBYTE io_Flags;
   BYTE io_Error;
   ULONG io_Actual:
   ULONG io_Length;
   APTR io_Data;
   ULONG io_Offset;
};
```

Una richiesta di I/O necessita di una modalità attraverso la quale essa può essere linkata a una lista di operazioni di I/O in attesa (io_Message) così come di una modalità attraverso la quale il device e le unità corrispondenti che devono operare su questa richiesta possano essere identificate (io_Device, io_Unit). Entrambe io_Device e io_Unit sono supportate dall'Exec durante l'operazione di apertura di OpenDevice. Questi valori non devono essere modificati dal programmatore.

Nomi delle Device	Scopo
audio.device	Controlla i canali hardware dell'audio
clipboard.device	Permette alle applicazioni di trasferire dati da e ad aree common per utilizzazioni tipo Cut e Paste tra le applicazioni stesse
console.device	Permette a una window di eseguire I/O standard come se fosse la console di un terminale connesso a un unità centrale di un computer
gameport.device	Controlla l'hardware delle porte giochi; sono gestiti sia gli Input digitali che quelli analogici
keyboard.device	Monitorizza e interpreta gli input da tastiera
input.device	Convoglia gli input dalla porta giochi e dalla tastiera in unico canale sincronizzato che deve essere interpretato dall'Intuition e magari da un device di console
timer.device	Gestisce i timer hardware per il Vertical Blanking e un timer ad alta precisione per gli intervalli dell'ordine dei microsecondi
trackdisk.device	Gestisce i dati in transito dal e al disco
narrator.device	Gestisce i dati testo per l'output del sintetizzatore vocale
serial.device	Gestisce l'hardware dell'interfaccia seriale
parallel.device	Gestisce l'hardware dell'interfaccia parallela
printer.device	Provvede a un comune formato input e a un formato output selezionato dalle preferences per le stampanti sia seriali sia parallele che possono essere connesse ad Amiga

Tabella 3.5

Una richiesta di input-output ha bisogno che gli venga detto quale comando deve eseguire il device (io_Command). I comandi specifici dei device sono inseriti nei file Include dei device stessi. I comandi più comuni sono elencati nel exec/io.h.

Spesso per formulare una richiesta si devono trasferire alcuni tipi di dati, infatti la IOStdReq include delle variabili che specificano:

- Quali opzioni particolari usare in caso di (io_Flags)
- Dove trovare i dati (io_Data)
- Quanti byte dati inviare o ricevere(io_Length)
- In quale posizione deve cominciare il trasferimento dei dati (io_Offset) se si tratta di un device con struttura a blocchi.
- Dove deve essere restituita la richiesta dal device una volta completato l'input-output (io_Message.mn_ReplyPort)

Le informazioni sullo status che vengono restituite specificano:

- Quanti byte di dati sono stati trasferiti come valore di ritorno dalla Request (io_Actual)
- Se si è verificato un errore, e che tipo di errore (io_Error)

Normalmente, io Length e io_Actual funzionano. Se non funzionano correttamente si potrà vederne il perché esaminando il valore di io_Error.

Nonostante la IOStdReq sia una forma molto comune di struttura dati per il trasferimento di informazioni dai ed ai device, molti dei device di Amiga hanno delle versioni speciali di strutture IORequest. Ogni particolare struttura è trattata nel capitolo che riguarda quel particolare device.

Inizializzazione minima necessaria per una richiesta di I/O

Nonostante essa vari per i diversi device che si vogliono utilizzare, ecco la minima inizializzazione che deve essere approntata prima che si possa trasferire una IORequest:

1. Aprire il device. Questo inizializza i campi io_Device e io_Unit

2. Inizializzare nella Request i seguenti campi:

```
block.io_Message.mn_Node.ln_Type = NT_MESSAGE;
                          /* il blocco Request è un messaggio */
block.io_Message.mn_Node.lnPri = 0;
                          /* setta la priorità del messaggio */
block.io_Message.mn_Node.ln_Name = NULL;
                          /* non è necessario un nome */
block.io_Messagge.mn_ReplyPort = NULL;
                          /* sia un valore nullo sia un effettivo */
                          /* indirizzo di una porta di risposta */
                          /* vanno bene */
```

(Questi sono i campi dati da inizializzare per una IOStdReq. Si devono inizializzare i campi corrispondenti del proprio blocco di risposta in accordo con questi.)

3. Specificare il comando che deve essere eseguito:

```
block.io_Command = WHATEVER;
                          /* si scriva il comando desiderato */
```

Inviare un comando a un device

Dopo aver aperto un device, si riempiono i campi appropriati della IORequest per comunicare al device cosa deve fare. Poi si invia al device questa Request in uno dei tanti modi possibili. Vi sono due modi normalmente usati per trasmettere una Request:

```
SendIO(request)           /* fa la richiesta di I/O */
                          /* ma non ne attende il compimento */

DoIO(request)             /* fa la richiesta e resta in attesa che */
                                /* venga esaudita */
```

Sui blocchi Request. Dopo aver inviato un blocco di Request (Request Block), (o un qualunque messaggio) al device, nonostante esso appartenga alla memoria controllata dal task non si deve assolutamente scrivere nello spazio ad esso riservato prima che il messaggio sia restituito alla porta di risposta. Questo spazio è di fatto affidato al device o ad un altro task per farne ciò che vogliono. Solo dopo che è stato restituito si può pensare di riutilizzarlo.

Il ritorno del blocco di Request alla porta di risposta significa che esso è stato aggiunto alla lista dei messaggi attaccati a quella porta di messaggio. Per riutilizzarne la memoria, bisogna richiamare la funzione GetMsg(ReplyPort) al fine di svincolare il messaggio dalla porta.

Sul SendIO. SendIO fornisce un puntatore alla Request e tenta di allegare il proprio messaggio alla lista dei messaggi del device. Il device è gestito da un task separato di Amiga all'interno del multitasking di Exec. La Request è eseguita in una sequenza FIFO (First-In-First-Out). Se il device è occupato da una request antecedente, il task del programmatore può continuare a fare qualcos'altro e controllare più tardi per vedere se l'I/O è stato completato. Poiché il task non attende che l'I/O sia completato, si dice che questa è una richiesta asincrona.

Quando la richiesta di I/O è stata esaudita, sia con sia senza successo, il blocco dati di richiesta (un messaggio) viene rimandato alla porta di risposta. Lo si deve rimuovere dalla porta di risposta prima di poterlo riutilizzare. Se si specifica un valore di risposta nullo allora non viene usata alcuna porta di risposta. Si può usare la funzione CheckIO per vedere se l'Input-Output è stato completato.

Su DoIO. DoIO è passato come puntatore alla richiesta. Quando si utilizza DoIO per trasmettere la richiesta, il task richiedente si pone in stato di inattività finché l'I/O richiesto non è completato o viene ritornato un errore. DoIO rimuove automaticamente il blocco di Request dalla porta di risposta prima che il proprio task si riattivi. Se il valore della porta di risposta è nullo, allora non saranno usate porte di risposta durante l'input-output.

Per trasmettere una richiesta a un device vi è un terzo modo, la funzione BeginIO(request). Però essa presuppone una buona conoscenza delle caratteristiche interne delle Device, aspetto non trattato approfonditamente in questo libro. La maggior parte degli esempi di questo volume utilizzano o SendIO o DoIO.

Altre funzioni di I/O

Le altre funzioni alle quali si può essere interessati sono WaitIO, AbortIO, e CheckIO. Per richiamare WaitIO si scriva :

```
WaitIO(request);
```

Se è stata inviata una richiesta a un device usando SendIO, e il programma è arrivato ad un punto dal quale non può più avanzare a meno che non venga completata la richiesta, si utilizza la funzione WaitIO per porre in stato di attesa il task finché la request non viene esaudita. WaitIO rimuove automaticamente il blocco di request dalla porta di risposta.

Per richiamare AbortIO si scriva:

```
AbortIO(request);
```

Si utilizza AbortIO per terminare una certa richiesta, che può essere già stata esaudita completamente o no. Per richiamare CheckIO si scriva:

```
result = CheckIO(request);
```

Il valore di result che viene restituito è true se la richiesta di input-output è stata completata. Se non fosse stata completata si può decidere di utilizzare per essa AbortIO.

Semplici chiamate delle funzioni di I/O

I programmi del listato 3.6 hanno il compito di illustrare un tipico uso delle funzioni di I/O. (Ulteriori esempi possono essere trovati nel sesto capitolo.) Questi programmi usano il timer e le seguenti strutture dati:

```
struct timerequest *message;     /* un puntatore a un messaggio */
                                 /* trovato in una porta
                                    di messaggio */
```

```
struct MsgPort myReplyPort;      /* la porta di risposta nella */
                                       /* si riceverà il messaggio
                                          quando */
                                       /* la richiesta sarà rinviata */
                                       /* dalla Device */

struct timerequest myTimeReq; /* un messaggio di Time Request */
```

Tutti i programmi danno per scontato che si sia aperto il device timer nel modo seguente:

```
long error;

error = OpenDevice("timer.device".UNIT_VBLANK,myTimeReq,0);
```

Tutti i programmi danno anche per scontato che si sia inizializzata la Time request con le seguenti istruzioni:

```
myTimeReq.tr_node.ln_Type = NT_MESSAGE;
                            /* il blocco di Request è un messaggio */

myTimeReq.tr_node.ln_Pri = 0;
                            /* setta la priorità del messaggio */
myTimeReq.tr_node.ln_name = NULL
                            /* non c'è bisogno di un nome */

                            /* usiamo una porta di risposta! */

myTimeReq.tr_node.mn_ReplyPort = &myReplyPort;
```

E infine, ogni programma da per scontato che il Timer sia stato inizializzato dalle seguenti istruzioni:

```
myTimeReq.tr_time.tv_secs = 3;          /* 3 secondi */
myTimeReq.tr_time.tv_micro = 0;         /* 0 microsecondi */
```

Si noti che nei programmi 1 e 2, non c'è bisogno di rimuovere il blocco di richiesta dalla porta di risposta, perché lo fanno automaticamente DoIO e WaitIO.

Perché usare una porta di risposta?

Come si è visto non è necessario specificare una porta di risposta quando si ha a che fare con un device di I/O nella quale l'I/O può essere completato in modo accettabile specificando un valore nullo per la porta di risposta. Perché, allora usare una porta di risposta?

```
/* Programma 1:. rende inattivo il task in attesa el completamento del-
l'I/O */

      DoIO(myTimeReq);  /* attende il completamento dell'I/O */

      /* si può ora procedere con qualcos'altro */
/* Programma 2: Fa partire l'I/O, si risincronizza con esso più tardi */

      SendIO(&myTimeReq);           /* fa partire il Timer */

      /*...fa altre cose...*/

      WaitIO(&myTimeReq);
      /* diventa inattivo fino al completamento dell'I/O */

      /* poi va avanti a fare qualcos'altro */

/* Programma 3: Fa partire l'I/O, fa altre cose, controlla poi ogni */
/* tanto per vedere se è stato completato */

      SendIO(myTimeReq);

otherthings:

      /*...fa altre cose..*/

      result = CheckIo(&myTimeReq);

      if(result == FALSE)
      {
            goto otherthings;
      }

      /* rimuove il blocco di Request dalla porta di risposta */
      message = GetMsg(&myReplyPort);

      /* e continua... */

/* Programma 4: Fa partire l'I/O, fa altre cose, controlla ogni tanto. Se
non è stato completato abortisce il tentativo e prosegue */

      SendIO(&myTimeReq);
```

```
/* ... fa altre cose ... */

result = CheckIO(myTimeReq);

if(result == FALSE)
{
      printf("Sto abortendo la Request di I/O"_;
      AbortIO(&myTimeReq);
}
else
{
      printf("L'I/O è stato completato");
}
/* Rimuove la Request dalla porta di risposta */

message = GetMsg(myReplyPort);

/* (il valore di message deve essere &myTimeReq) */

/* e continua ... */
```

Listato 3.6

Bene, all'inizio di questo capitolo si è parlato di segnali, e si è visto che un task può attendere in stato di inattività che uno o più eventi si verifichino prima di riattivarsi. Il ricevere un messaggio a una porta di risposta (che è poi una porta di messaggio) costituisce proprio uno di questi eventi. Per questo, settare le porte di risposta è uno strumento utile per il multitasking, e per una sequenza multieventi.

Accodamento di richieste multiple.

Se si desidera che un device compia parecchie operazioni in modo asincrono rispetto alle operazioni del task richiedente, si può utilizzare SendIO per inviare delle richieste multiple al device. Si noti che si deve inizializzare un blocco di request separato per ogni singola richiesta. Come notato precedentemente la strada normale per inizializzare un blocco di request è l'uso della funzione OpenDevice. Però, se si utilizzano per uno stesso device delle richieste multiple l'una dietro l'altra, i valori di io_Device e di io_Unit che sono stati ricevuti dalla prima chiamata di OpenDevice possono essere copiati dal blocco iniziale di request in ognuno di quelli successivi, e saranno validi finché il task mantiene aperto il device.

Ecco alcune istruzioni dove una seconda richiesta viene copiata da quella originale:

```
struct timerequest tr;
struct timerequest tr2;

error = OpenDevice("timer.device",UNIT_VBLANK,&tr,0);
if(error == 0)
{
tr2.io_Device = tr.io_Device;
tr2.io_Unit = tr.io_Unit;
}
/* ora entrambe possono essere usate per l'I/O della Device */
```

Accedere alle funzioni di libreria dei device

La struttura di una device e la struttura di una libreria sono molto simili, e i vettori delle funzioni di device sono settati nello stesso modo di quelli delle librerie. Alcuni device, come il timer e il console, hanno delle funzioni che possono essere eseguite in C come se fossero delle routine di libreria. Questi device hanno inoltre una variabile riservata per l'indirizzo base. Per il timer, il nome di questa variabile è TimerBase. Per la Console, il nome della variabile è ConsoleBase.

Per usare le routine nella biblioteca del device, si deve fornire un valore appropiato alla variabile dell'indirizzo base. Si ottiene questo valore aprendo il device e usando il valore di io_Device che l'Exec fornisce nel blocco della IORequest. Ecco un programmino che setta correttamente la variabile del device timer:

```
          /* Programma... per settare il valore di TimerBase */
          /* permette di utilizzare le Routine aritmetiche */
          /* AddTime, CmpTime, SubTime */

extern struct Library *TimerBase;
long error;
struct timerequest tr;

error = OpenDevice("timer.device".UNIT_VBAN,tr,0);

if(error == 0)
{
   TimerBase = tr.tr_node.io_Device;
```

```
}
else
{
   printf("errore %ld nell'acceso al timer"IOErr());
         /(* non si tenti di usare le funzioni di Timer) */
}
```

Una volta che la variabile è stabilita, si possono utilizzare le funzioni di time documentate nel Manuale della ROM Kernel di Amiga - AddTime, CmpTime, e SubTime.

Il console.device ha una routine nella sua libreria, RawKeyConvert. Il settaggio della variabile di console verrà spiegato nel quinto capitolo poiché presuppone la conoscenza delle window e questo concetto non è ancora stato introdotto.

Chiudere un device

Una volta terminato l'uso del device bisogna chiuderlo per terminare l'accesso del task ad essa. Questo è molto importante per i device residenti su disco poiché ognuno di questi device una volta caricato e inizializzato, occupa della memoria e magari sfrutta del tempo della CPU finché l'ultimo task la mantiene aperta. Per questo, una volta finito l'uso del device, lo si chiude per permettergli di liberare la memoria e le altre risorse che sta occupando e di restituirle all'Exec perché le riutilizzi.

Si chiude un device richiamando la funzione CloseDevice. Per utilizzare tale funzione si scriva:

```
CloseDevice(request);
```

dove request è un puntatore al blocco di IORequesr che si è inizializzato con la funzione OpenDevice. Si noti che se il blocco della request era stato copiato per creare delle richiesta multiple accodate, si deve chiudere il device usando uno solo dei Request Block. In altre parole, si richiami CloseDevice per chiudere il device come se il task l'avesse appena aperta.

Inoltre, poiché lo si chiude ci si accerti che il device abbia risposto a tutte le request di I/O inviatele. Se non ha risposto a tutte, si può utilizzare la funzione AbortIO per far abortire queste richieste. Facendo questo si svuota di fatto la li-

sta di messaggi del device, messaggi ai quali essa doveva rispondere prima che il task dicesse al device che non ha più bisogno delle sue funzioni.

I device possono essere ripartiti tra i vari task. Ogni volta che un task apre un device, esso incrementa il suo contatore degli utenti. Ogni volta che un task lo chiude, esso decrementa tale contatore. Se, quando un task chiude il device, il contatore va a 0, il device si può autoeliminare dal sistema e liberare le risorse occupate.

Capitolo 4

La grafica

Nel presentare le capacità grafiche uniche di Amiga, questo capitolo fornisce degli utili tool che potranno essere inseriti nei propri programmi. Tutti i programmi che si troveranno qui sono compatibili con l'intuition, l'interfaccia utente di Amiga. Vi sono dei punti di entrata a basso livello per le routine di sistema, soprattutto per la grafica e i layers, che non saranno trattati qui per far si che si possano creare programmi compatibili gli uni con altri e con l'intuition.

Questo capitolo fornisce le funzioni di base su come creare e inizializzare le aree di disegno; come specificare e selezionare i colori; come riempire le aree; e come disegnare linee e rettangoli.

Si troveranno prima delle applicazioni che operano sullo schermo del Workbench e poi su degli schermi personalizzati. Si potrà creare un diagramma a barre su Workbench e poi una mappa geografica su uno schermo personalizzato. Si vedrà come programmare il testo per i vari font e i vari formati. Infine, si disegnerà una figura.

Aprire una window sullo schermo del Workbench

Per creare una window sullo schermo del Workbench, si deve specificare ciò che segue:

- Dove posizionare la window (cioè dove posizionare l'angolo superiore sinistro)
- Quanto debba essere grande (quanto larga, quanto alta)
- In che colore disegnare i contorni e i gadget di sistema
- I flag per controllare il tipo di window che si sta creando e i gadget che devono essere applicati ad essa automaticamente
- Se la window può essere ridimensionata, i valori minimi e massimi di tale ridimensionamento
- Il tipo di schermo (in questo caso, il Workbench) nel quale si vuole aprire la window

Definire una nuova window

Per creare una window, si definisce una struttura dati chiamata NewWindow. Il listato 4.1 è una definizione tipica di una NewWindow. Si userà questa definizione per il programma del diagramma a barre.

```
Struct NewWindow myWindow
{
/* window1.h */
      /* alla risoluzione orizzontale corrente, */
                              /* dall'angolo sup. sin. dello schermo */
      30,                     /* l'estremità sup. della window è */
                              /* misurata in linee a partire dall'alto */
                        /* dello schermo */
      500,150,                /* Larghezza e altezza di questa window */
      -1,                     /* DetailPen - che penna deve essere */
                              /* usata per disegnare il bordo */
      -1,                     /* BlockPen - che penna deve essere usata */
                              /* per disegnare i Gadgets */
                              /* (per DetailPen e BlockPen il valore -1 */
                              /* è quello di Default) */
      CLOSEWINDOW | NEWSIZE | REFRESHWINDOW,
                              /* Flag IDCMP */
      SIMPLE_REFRESH | NORMALFLAGS | GIMMEROZERO,
                              /* Flag della window */
      NULL,                   /* FirstGadget... spiegato nel quinto */
                              /* capitolo */
      NULL,                   /* CheckMark... spiegato nel quinto */
                              /* capitolo */
      "Sample Chart",         /* Titolo della window */
      NULL,                   /* puntatore allo schermo se non è */
                              /* Workbench */
      NULL,                   /* puntatore alla Bitmap se si tratta */
                              /* di una window SuperBitmap */
      10,10,                  /* larghezza minima, altezza minima */
      640,200,                /* larghezza massima, altezza massima */
      WBENCHSCREEN            /* tipo di schermo nel quale aprirla */
};
```

Listato 4.1

I flag IDCMP. L'intuition è in grado di inviare messaggi a un task creato dal programmatore per informarlo che un utente ha selezionato un certo tipo di gadget o che ha fatto qualcosa che il task del programmatore dovrebbe sapere. I messaggi che l'intuition invia sono ricevuti dalla Intuition Direct Communication Message Port (IDCMP). In questo paragrafo, si troverà un programma che atten-

de l'arrivo di un messaggio alla IDCMP. Esso poi agirà in un modo determinato dal messaggio arrivatogli.

Ecco i flag che sono stati selezionati per queste prime applicazioni:

CLOSEWINDOW	così l'applicazione sa quando l'utente vuole terminare il programma .
NEWSIZE	in modo che, se l'utente varia a grandezza della window, l'applicazione possa ridisegnare la grafica della window per riempire le parti vuote che vengono a formarsi.
REFRESHWINDOW	in modo che se l'utente pone la window davanti o la mette dietro alle altre, l'applicazione possa ridisegnare il contenuto della window stessa.

Flag di window. Vi sono tre flag specificati nella struttura della di Window1:SIMPLE_REFRESH, NORMALFLAGS, e GIMMEZEROZERO.

Settare il flag SIMPLE_REFRESH significa che se una window è posta dietro, e poi riportata di nuovo in avanti, il sistema non deve salvare automaticamente le sezioni che erano state coperte. Questo preserva la memoria, ma il programma deve essere in grado di ripristinare le fattezze della window dopo essersi accorto che è avvenuto un Refresh.

Per questa applicazione ho settato NORMALFLAGS come la maggior parte delle volte che ho progettato una schermata, ho voluto che la mia window avesse tutti in normali gadget di sistema, come segue:

WINDOWDRAG	permette di spostare la window con il mouse
WINDOWSIZING	permette di ridimensionare la window
WINDOWCLOSE	permette di segnalare all'applicazione che deve cessare le operazioni e chiudere tutto
WINDOWDEPTH	permette di posizionare la window di fronte o dietro alle altre
NORMALFLAGS	viene definito dalle istruzioni seguenti:

```
/* mydefines.h */

#define WC WINDOWCLOSE
#define WS WINDOWSIZING
#define WDP WINDOWDEPTH
#define WDR WINDOWDRAG

#define NORMALFLAGS (WC|WS|WDP|WDR)
```

Quando si apre una window viene creata e inizializzata una struttura dati window. Una struttura dati RastPort è una parte della struttura window. Essa è usata per controllare dove e come i disegni vengono eseguiti dalle routine grafiche.

Il settare il flag GIMMEZEROZERO implica che l'area di disegno avrà delle coordinate cartesiane per le quali l'origine si trova nell'angolo in alto a sinistra all'interno della window (0,0). Significa anche che tutto ciò che viene disegnato è contenuto nei bordi della window. Se non si specifica GIMMEZEROZERO, l'area effettiva di disegno per una window è posizionata a partire dall'angolo in alto a sinistra ma il disegno può uscire dai bordi della window stessa.

Aprire una window

Ecco un insieme di istruzioni che permette di aprire una window del tipo descritto nella struttura Window1 (myWindow):

```
struct Window *w;
w = OpenWindow(myWindow);
if(w == 0)
{
   printf("la window non si è aperta");
}
```

Si fornisce un puntatore a una struttura dati NewWindow (myWindow), e il sistema restituisce un puntatore alla struttura dati window che ha inizializzato seguendo i parametri contenuti nella struttura NewWindow.

Se il valore restituito è nullo, può darsi che non vi sia abbastanza memoria libera perché il sistema possa creare la struttura dati window e tutte le aree di memoria ad essa associate. Se il valore ritornato non è nullo, si sa di avere un puntatore valido ad una struttura dati window. Significa anche che la window sopra descritta è ora aperta sul Workbench.

Manipolazione degli eventi dall'Intuition

I flag IDCMP che si settano nella struttura dati NewWindow informano Intuition che deve inviare tre diversi tipi di eventi al task. C'è bisogno di una routine che faccia la cosa appropriata se avvengono uno o più di questi eventi. Chiameremo qui tale routine HandleEvent.

HandleEvent sarà richiamata dal programma principale, più avanti nel capitolo, per gestire degli eventi come la pressione del tasto del mouse sul gadget CLOSEWINDOW della window, o un ridimensionamento o un movimento fronte-retro. Se viene percepito un evento CLOSEWINDOW, tutto ciò che deve fare HandleEvent è restituire un valore nullo. Questo informa il programma principale che deve terminare. Per una richiesta di ridimensionamento o di spostamento fronte-retro, la grafica sarà ridisegnata, sempre che si tratti di una simple refresh window.

Il listato 4.2 è il programma per manipolare questi eventi. Ulteriori manipolazioni degli eventi sono mostrate nel prossimo capitolo.

Posizionare la RastPort

Il passo successivo consiste nel posizionare la RastPort all'interno della neocreata struttura dati window. Una RastPort è una struttura dati che controlla il disegno all'interno di un'area di disegno.

```
/* eventol.c */

HandleEvent(class)
      LONG class;             /* fornita dal main */
{
      switch(class)
      {
            case CLOSEWINDOW:
                        return (0);
                        break;

            case NEWSIZE;
            case REFRESHWINDOW;
                        redraw();
                        break;
            default:
                        break;
      }
                  return(1);
}
```

Listato 4.2

Per molti programmi, non è necessario conoscere nulla sui contenuti interni della RastPort. Si deve solo avere un puntatore per un struttura dati RastPort valida da passare alle funzioni di sistema, che utilizzano il contenuto corrente della RastPort per esaudire la richiesta loro fatta.

Tra le informazioni contenute nella RastPort vi è la posizione attuale della penna di disegno, il colore attuale della penna di disegno, e la larghezza e la altezza del font attualmente selezionato. Si troverà un descrizione approfondita della struttura delle variabili delle RastPort nel secondo capitolo del primo volume dell'Amiga Programmer's Handbook di E. P. Mortimer (SYBEX, 1987). La descrizione delle struttura va al di la dello scopo di questo libro; le variabili della struttura saranno trattate solo per l'uso che se ne fa. Si può locare la RastPort per la neocreata window usando le seguenti istruzioni:

```
Struct RastPort *rp;
/* un puntatore alla RastPort */
rp = w->RPort;
/* la struttura window contiene */
/* una RastPort, così abbiamo bisogno */
```

```
/* l'indirizzo di questa parte della */
/* struttura Window */
```

E questo è tutto ciò che vi è da dire sulla RastPort.

Disegnare sulla window

Fin qui, le varie parti di programma hanno definito le caratteristiche della window e hanno fornito un puntatore alla struttura RastPort che serve a controllare il tracciamento dei disegni nella window. Ora sono terminati questi passi preliminari che hanno portato il programma ad essere compatibile con l'intuition, si possono utilizzare quelle istruzioni che utilizzano realmente il puntatore alla RastPort - le istruzioni cioè che creano i disegni. Questi programmi utilizzano le funzioni interne della graphics.library .

In questo paragrafo, si disegnerà un istogramma. Per rendere questo esercizio più utile, questa applicazione fornisce dei tool di uso generale nella grafica. Vi si possono, infatti, trovare i tools per:

- Selezionare i colori
- Selezionare un modo di tracciamento
- Disegnare gli assi
- Dare un nome agli assi
- Disegnare dei quadrati
- Disegnare delle linee tratteggiate

Disegnare un grafico spesso implica piazzare da qualche parte all'interno della window gli assi cartesiani, e poi disegnare i componenti del grafico stesso. E' molto conveniente poter riferire i componenti del grafico al sistema di coordinate del grafico stesso e non a quello della window. Così si deve creare un set di routine che facciano questo.

Tutte le funzioni grafiche qui usate utilizzano una struttura dati comune che permette di convertire le coordinate del grafico in coordinate della window. Le funzioni si aspettano una X e una Y relative al sistema di riferimento della win-

dow, così queste routine correggono le richieste di sistema. Il listato 4.3 è la struttura dati usata per maneggiare i valori base.

```
/* xybase.h */

struct XYBase
{
      WORD xaxis;         /* posizione dell'asse X nella window */
      WORD yaxis;         /* posizione dell'asse Y nella window */
      WORD xlength;       /* lunghezza dell'asse X */
      WORD ylength;       /* lunghezza dell'asse y */
};
```

Listato 4.3

Selezionare i colori

Quando si lavora sullo schermo del Workbench, ci sono quattro colori di penna tra i quali poter scegliere. Il sistema li identifica con i numeri di penna 0, 1, 2 e 3.

Il colore che si sta scegliendo può essere visto nello schermo primario delle Preferences nell'angolo in basso a sinistra. Il colore 0 è quello più a sinistra, il colore uno è quello dopo, e così via. Qualunque numero di penna si selezioni, quel particolare colore corrispondente sarà utilizzato per costruire l'immagine. Si noti che il numero 0 corrisponde al colore dello sfondo.

Ci sono tre diversi tipi di penne per disegnare, conosciute come APen, BPen, e OPen.

La APen. La APen è il colore della penna primaria . Quando si traccia una linea ben marcata o un area colorata, questo è la penna che si utilizza. Si stabilisce il colore della penna primaria usando l'istruzione :

```
SetAPen(rp,penNumber);
```

dove rp è il puntatore alla RastPort da utilizzare, e penNumber è un valore minore o uguale al massimo numero di colori disponibili per quella RastPort. Per questo esempio, il massimo numero è 3.

Se poi si vuole trovare il valore correntemente assegnato alla APen, si può cercare nella struttura dati RastPort, sotto il nome FgPen (la penna di Foreground). Si accede a tale valore, usando il puntatore alla RastPort, come segue:

```
CurrentPen = rp->FgPen;
```

La BPen. La BPen è il secondo colore di penna. Una linea disegnata con un pattern o una area disegnata con un pattern possono esser create con una combinazione di bit-1 e bit-0 all'interno di un pattern. Quando una linea o un area di questo tipo vengono tracciate, le zone occupate dai bit-1 nel pattern sono riempite con il colore primario di penna (APen), e le zone occupate da bit-0 con il colore di penna secondario (BPen). Questo capita solo se si è settato il modo di disegno JAM2 per la RastPort corrente (si troverà più avanti una spiegazione dei vari modi di disegno).

Si stabilisce il colore della BPen usando l'istruzione:

```
SetBPen(rp,penNumber);
```

Il valore BPen è immagazzinato nella struttura dati RastPort come voce di nome BgPen (penna di Background). Si può accedere a questo valore, utilizzando il puntatore alla RastPort, nel modo seguente:

```
CurrentBPen = rp->BgPen;
```

La OPen. La OPen è anche chiamata la penna di Outline. Se si sta creando un poligono pieno e se è stato settato un valore per la OPen, subito dopo che il poligono è stato riempito (filled) il sistema lo contornerà automaticamente con una linea del colore di OPen.

Si stabilisce il valore del colore della penna di Outline con la seguente istruzione:

```
SetOPen(rp,penMumber);
```

Questo valore viene immagazzinato nella struttura dati RastPort sotto la voce AOIPen (penna di Area-Outline). Vi si può accedere usando il puntatore alla RastPort come segue:

```
CurrentOPen = rp->AOIPen;
```

Si potrebbe voler recuperare questo valore e assegnarlo alla APen quando si stanno compiendo operazioni di colorazione totale (Flood). (Tali operazioni saranno discusse nell'esempio dello schermo personalizzato più avanti in questo capitolo).

Selezionare il modo di disegno

Se si vuole tracciare una linea semplice si seleziona il modo JAM1. Questo significa "usa un colore solo nell'area di disegno". Lo si può usare per disegnare gi assi x e y come delle semplici rette solide. Si setta il modo di disegno utilizzando la funzione StDrMd, come segue:

```
SetDrMd(rp,JAM1);
```

Il modo JAM2 significa "disegna utilizzando due colori, non solo uno". La APen è utilizzata ovunque vi sia un bit-1 presente nel pattern di linea o nel pattern dell'area. LA BPen è utilizzata dovunque vi sia un bit-0 presente nel pattern.

Il modo COMPLEMENT significa che, qualunque colore sia presente nell'area bersaglio (target), ogni bit-1 che si trovi nella sorgente o nel pattern fa in modo che il complementare del colore nell'area bersaglio (target) sia disegnato. Un colore complementare è un colore nel quale i bit-1 diventano bit-0 e viceversa. Per esempio, per un codice binario di colore a 4-bit 0101, il colore complementare è 1010. Il modo COMPLEMENT è utile quando si vuole disegnare, e poi cancellare, una linea (come nel caso dei rettangoli per gli spostamenti delle window). La si disegna prima in modo COMPLEMENT; la linea appare. La si ridisegna in modo COMPLEMENT; tutti i bit della linea ritornano al loro valore originale e la linea svanisce.

I modi JAM1 e JAM2 possono anche essere usati in combinazione con il modo INVERSEVID. Esso e spiegato nella discussione sul testo più avanti nel capitolo.

Disegnare gli assi

Prima di cominciare a disegnare bisogna posizionare la penna nella locazione desiderata, e poi tracciare la linea fino all'ultimo punto. Questa operazione è svolta dalle funzioni Move(rp,x,y) e Draw(rp,x,y.)

La funzione Move preleva la penna e la posiziona in una nuova locazione. La funzione Draw traccia un segmento di retta, nel modo di disegno correntemente assegnato, dalla posizione corrente alla posizione in essa specificata.

Molte funzioni di tracciamento possono spostare la posizione corrente della penna. Se, invece di settare specificatamente la pozione della penna con la funzione Move, si volesse semplicemente sapere dove si trova la penna in un certo momento, si possono esaminare le variabili cp_x e cp_y, usando il puntatore alla RastPort nel modo seguente:

```
CurrentXPenPosition = rp->cp_x;
CurrentYPenPosition = rp->cp_y;
```

Il listato 4.4 disegna gli assi secondo i valori della struttura XYBase ricevuta dalla funzione DrawAxes. I suoi input sono un puntatore alla struttura XYBase che mostra già dove debbano essere posizionati gli assi, un puntatore alla RastPort all'interno della quale gli assi devono essere disegnati, il colore della penna con la quale tracciare gli assi, la giacitura degli assi e la loro lunghezza.

Si noti che, per semplicità non viene incluso nessun controllo degli errori. Per esempio, se il valore di yaxis meno ylength risulta negativo, potrebbe verificarsi un errore. Si noti inoltre che la funzione DrawAxes è una funzione definita dal programmatore e non una funzione della ROM Kernel.

Dare un nome agli assi. Per mettere un'intestazione a ogni asse, si deve sapere come usare i testi su Amiga. Per usare un testo, bisogna specificare dove

deve essere disegnato nella RastPort, il puntatore al testo che deve essere scritto, e la lunghezza della stringa.
Per specificare dove posizionare il testo, si usa la funzione Move; essa muove la posizione della penna. Quando dei caratteri di testo vengono generati, si inizia la loro scrittura come se la penna di disegno si trovasse alla loro linea di base. Per i font standard di sistema, i cui caratteri sono creati in una griglia di 8x8 pixel, la linea di base è la settima linea a partire dall'alto della griglia di contenimento. Si può trovare il valore corrente della linea di base leggendo il valore di TxBaseline all'interno della RastPort. Questo può essere fatto nel seguente modo:

```
CurrentTextBaseline = rp-> TxBaseline;
```

```
/* drawaxes.c */

DrawAxes (rp,xyb,xaxis,yaxis,xlength,ylength,color)

      struct RastPort *rp;
      struct XYBase *xyb;
      LONG xaxis, yaxis;
      LONG xlength, ylength;
      LONG color;

{
      SetAPen (rp,color);

      /* disegna l'asse x */

      Move(rp,xaxis,yaxis);
      Draw(rp,xaxis + xlength,yaxis);

      /* disegna l'asse y */

      Move(rp,xaxis,yaxis);
      Draw(rp,xaxis,yaxis - ylength);

      /* adesso setta i valori di XYBase in modo che possano */
      /* utilizzarlo altre routine */

      xyb->xaxis = xaxis;
      xyb->yaxis = yaxis;
      xyb->xlength = xlength
      xyb->ylength = ylength
}
```

Listato 4.4

Quando il carattere è stato disegnato, la penna si è mossa automaticamente di un carattere a destra, in modo da poter cominciare a scrivere un altro carattere, a partire dalla linea di base, se lo si desidera.

Una volta che la penna è posizionata e il modo di disegno e il colore della penna sono stati stabiliti, la funzione che si richiama per scrivere il carattere è chiamata Text. Per richiamare Text si scriva:

```
Text(rp,tptr,length);
```

dove rp è un puntatore a una RastPort all'interno della quale tracciare il testo, tptr è un puntatore al testo da scrivere, e length è il numero di caratteri del testo stesso.

Per usare Text per dare un nome agli assi, si deve posizionare il testo in accordo con la sua lunghezza e con la lunghezza degli assi. Se il testo deve essere centrato sulla sua posizione, bisogna conoscerne la lunghezza. Per questo, si può utilizzare la funzione TextLength. Per richiamarla si scriva

```
length = TextLength(rp,tptr,length);
```

dove i valori passati alla funzione sono esattamente gli stessi passati a Text.

Il listato 4.5 scrive un nome per entrambi gli assi. L'array per questi nomi è assunto come array di una stringa null-terminated. Per convenienza e velocità si utilizza l'aritmetica degli interi .

E' più efficiente tracciare i caratteri raggruppati a stringhe che non singolarmente per due ragioni. Innanzi tutto vi è un limite associato alle chiamate ripetute delle stesse subroutine. Inoltre, se si è stabilito uno stile speciale per il proprio testo, come italics o bold, il testo non verrà reso correttamente se si disegnano dei caratteri singoli. Il sistema residente di output per i testi può accettare un insieme di caratteri formattarlo correttamente come stringhe. Per esempio, se un set di lettere italics si ricoprono o si appoggiano l'una all'altra, una singola chiamata della funzione Text formatterà tali lettere correttamente in una lunga stringa.

Disegnare dei rettangoli

Gli istogrammi sono composti da rettangoli di diverse dimensioni e di diverso aspetto. Alcuni possono essere solo dei contorni disegnati, altri possono essere dei rettangoli riempiti con un colore. Il colore può essere poi uniforme o a pattern per meglio distinguere una barra dall'altra all'interno del diagramma. Questo paragrafo mostra come utilizzare le funzioni di sistema per generare dei rettangoli vuoti, colorati uniformemente e riempiti con un pattern.

Come nelle altre subroutine per i diagrammi qui presenti, i rettangoli verranno generati relativamente al sistema di riferimento del grafico e non a quello della window. Così l'utente può creare un grafico come se disegnasse su una carta millimetrata.

Rettangoli vuoti. Un rettangolo vuoto può essere creato con la funzione Move seguita da quattro diverse funzioni Draw. Poiché tali funzioni sono spiegate nel paragrafo precedente, questo esercizio è lasciato al lettore.

Rettangoli colorati o riempiti con pattern. I rettangoli pieni, di colore uniforme o di un pattern, possono essere prodotti dalla funzione di sistema chiamata RectFill. Per richiamare RectFill si scriva:

```
RectFill(rp,xmin,xmax,ymin,ymax);
```

dove rp è il puntatore alla RastPort all'interno della quale deve essere tracciato il rettangolo; xmin, ymin è l'angolo superiore sinistro del rettangolo e xmax, ymax è l'angolo inferiore destro.

Ciò che risulta dalla chiamata di RectFill dipende dal valore correntemente settato in alcuni parametri della RastPort, incluso il modo di disegno, i pattern di riempimento delle aree e i colori delle tre penne di disegno.

```
/* labelaxws.c */
LabelHorizontal(rp,xyb,labels,howmany)

      struct RastPort *rp;
      struct XYBase *xyb;
```

```
        char *labels[];              /* un array di nomi di label */
        LONG howmany;                /* quanti label sono richiesti */
{
        WORD i, labelwidth, segmentwidth, currentx;
        WORD actualy, actulx;

        segmentwidth = (xyb->xlength)/(howmany - 1);
        currentx = xyb->xaxis;

        actualy = xyb->yaxis;
        /* lo pone sotto la linea dell'asse */

        for(i=0; i<howmany; i++)
        {
                labelwidth = TextLength(rp,labels[i],strlen(labels[i]));

                labelwidth = labelwidth/2;  /* centra il testo */

                actualx = currentx + (segmentwidth * i) - labelwidth;

                Move(rp,actualx,actualy);
                Text(rp,labels[i],strlen(labels[i]));
        }
}               /* fine del label orizzontale */

LabelVertical(rp,xyb,labels,howmany)
        struct RastPort *rp;
        struct XYBase;
        char *labels[];              /* un array di nomi */
        LONG howmany                 /* quanti labels sono richiesti */

        WORD i, labelwidth, segmentheight, currentx;
        WORD actualy, actualx,currenty;

        segmentwidth = (xyb->ylength)/(howmany - 1);
        currentx = xyb->xaxis-2;     /* due pixel dall'asse */
        /* centra verticalmente il testo, utilizzando il valore */
        /* dell'altezza di testo contenuto nella RastPort */
        currenty = xyb->yaxis + ((rp->TxHeight)/2);;

        for(i=0; i<howmany; i++)
        {
                labelwidth = TextLength(rp,labels[i],strlen(labels[i]));
                /* allinea il testo lungo il margine destro dell'asse */
                actualx = currentx - labelwidth;

                actualy = currenty - (segmentheight * i);
```

```
            Move(rp,actualx,actualy);
            Text(rp,labels[i],strlen(labels[i]));
        }
}               /* fine del label verticale */
```

Listato 4.5

Per un rettangolo a colore uniforme, si può usare la parte di programma che segue:

```
SetPen(rp,mycolor);
/* setta il colore della penna */
SetDrMd(rp,JAM1);
/* utilizza un solo colore */
/* nell'area di disegno */
/* poi richiama RectFill */
```

Per un rettangolo a colore uniforme con bordo delineato si usi ciò che segue, e si specifichi la penna di outline prima di richiamare RectFill:

```
SetOPen(rp,myoutline);
/* setta il colore della penna di Outline */
```

Rettangoli senza contorni. Una volta che il colore di Outline è stato stabilito, il sistema userà quel colore sia per RectFill sia per il riempimento di aree grafiche. Il riempimento di aree è trattato nell'esempio dello schermo personalizzato più avanti in questo capitolo). Per terminare l'uso della penna di Outline, si deve usare la macro di sistema BNDRYOFF:

```
BNDRYOFF(rp);
```

Rettangoli riempiti con pattern a due colori. Per creare un rettangolo riempito con un pattern a due colori, è necessario specificare il colore della penna primaria e di quella secondaria, il modo di disegno JAM2, e bisogna utilizzare un pattern per il riempimento. Il pattern è un array di parole a 16 bit usate dalle funzioni di riempimento come se fossero immagazzinate una sopra l'altra per formare un vasto pattern a 16 bit di qualunque grandezza si voglia. La restrizione principale è che la dimensione del rettangolo deve essere una potenza di due (2, 4, 8, eccetera).

Il listato 4.6 è un pattern che forma una scacchiera una volta che viene visualizzato. Quando le funzioni grafiche usano questo pattern, se il modo di disegno

è JAM2, dovunque vi sia un bit-1 nel pattern, il sistema disegna con un colore di penna APen. Dovunque vi sia un bit-0 nel pattern il sistema usa un colore di penna BPen. Se il modo è JAM1, e utilizzata solo la APen.

Per dire al sistema quale pattern debba essere usato, si utilizzi la funzione SetAfPt (set area-fill pattern).Questa funzione richiede un puntatore alla RastPort che sarà coinvolta, un puntatore al pattern, e la dimensione del pattern che si utilizza (espressa in potenza di due). Il pattern di esempio è lungo 8 parole, così la sua dimensione è $8(2^3)$.

```
SetApen(rp,myPrimaryColor);
SetBPen(rp,mySecondaryColor);


   SetAfPt(rp,mypaettern,size);        /* setta un pattern */
SetDrMd(rp,JAM2);                      /* disegnando,
                                        usa entrambe le penne */
                                       /* poi richiama RectFill */
```

Per disegnare un rettangolo riempito con un pattern, si usino le seguenti istruzioni prima di richiamare RectFill:

```
      /* mypattern.h */

UWORD mypattern[] =
{
      0xf0f0,      /* 4 Bit-1 e 4 Bit-0 */
      0xf0f0,
      0xf0f0,
      0xf0f0,
      0x0f0f,
      0x0f0f,
      0x0f0f,
      0x0f0f,
};
```

Listato 4.6

Rettangoli riempiti con pattern multicolori. Invece di disegnare un pattern a due colori, si può disegnare un pattern contenente tutti i colori disponibili in quell'area grafica. Per questo esempio, dove è utilizzato lo schermo del Workbench, vi è una disponibilità di quattro colori. In uno schermo personalizzato, si può avere a disposizione una scelta fra 4096 colori diversi per ogni bit del pattern in modo HAM (Hold-And-Modify). Si veda l'appendice B del primo volume dell'Amiga Programmer's Handbook per la discussione dei modi di visualizzazione di Amiga.

I colori che sono visualizzati sullo schermo di Amiga sono determinati da un set di data-bit prelevati da varie parti della memoria conosciute come piani di bit (BitPlanes).E'una combinazione di questi colori che seleziona il colore che si vede sullo schermo.

Per visualizzare un pattern multicolore, si deve specificare il pattern nella forma di una sovrapposizione di vari BitPlane. La combinazione dei valori binari dei bit, prelevati dai singoli BitPlane del pattern, determina il colore visualizzato in una certa posizione del pattern.

Ecco come vengono combinati i bit:

Bit nel Plane 1		Bit nel Plane 2		Colore della penna
0	+	0	=	0
0	+	1	=	1
1	+	0	=	2
1	+	1	=	3

Ecco un pattern che forma delle strisce multicolori:

Posizione dei bit nei dati del pattern

```
7 6 5 4 3 2 1 0
0 1 0 2 0 3 0 1
0 1 0 1 0 3 0 1
1 0 2 0 3 0 1 0
1 0 2 0 3 0 1 0
```

```
02030102
02030102
20301020
20301020
```

Il listato 4.7 forma delle strisce multicolori sul colore di sfondo.

```
/* multypat.h */

UWORD mymulti[] =
{
      /* parte del piano 0 del Pattern multicolore */

      0x3033,
      0x3033,
      0xc0cc,
      0xc0cc,
      0x0330,
      0x0330,
      0x0cc0,
      0x0cc0,

      /* parte del piano 1 del Pattern multicolore */

      0x0330,
      0x0330,
      0x0cc0,
      0x0cc0,
      0x3003
      0x3003,
      0xc00c,
      0xc00c
};
```

Listato 4.7

Si può specificare che il pattern è multicolore con le seguenti istruzioni:

```
SetAPen(rp,255);                /* disegna in tutti
                                   i piani disponibili */
SetBPen(rp,0);                  /* la BPen deve essere 0 */
SetDrMd(rp,JAM2);               /* usa il modo di disegno JAM2 */

SetAfPt(rp,mymulti,-3);         /* -3 rappresenta il numero delle */
                                /* word che creano
                                ogni immagine del Pattern */
```

Ci devono essere tante immagini quanti sono i BitPlane nei quali disegnare. Workbench utilizza uno schermo a quattro colori; così esso richiede due immagini. Una di esse appare in un BitPlane, l'altra appare nel secondo BitPlane. Il valore mostrato è -3 perché vi sono 8 (2) parole (word) in ogni piano dell'immagine.

Sui pattern in generale. Nonostante le capacità dei pattern su Amiga siano impressionanti, vi è una limitazione: i pattern sono tutti disegnati a partire da un punto relativo all'angolo superiore sinistro dell'area di disegno della RastPort. Così, ogni figura disegnata con un pattern può interagire negativamente con uno sfondo creato con un pattern fisso. Questo può essere particolarmente fastidioso se si vogliono animare degli oggetti creati con dei pattern disegnandoli e ridisegnandoli velocemente contro uno sfondo fisso che non contenga pattern.

Come esempio, si supponga di voler disegnare due carte da gioco, entrambe aventi sul retro lo stesso pattern, e si muova l'una lentamente davanti all'altra. Basandosi sulla posizione della carta in movimento, si può cambiare il pattern che il sistema usa per il riempimento dell'area così da mantenere uguale l'aspetto del pattern stesso a prescindere da dove si sposti la carta sullo schermo. Come facile alternativa, si può creare una RastPort fuori schermo, creare qui la propria immagine, e poi copiarla sulla RastPort dello schermo, mantenendo uguale il pattern senza curarsi di dove la carta si muova sullo schermo.

Adattamento delle coordinate di riferimento. Il listato 4.8 trasforma le coordinate del rettangolo portandolo dal sistema relativo alla window a quello relativo agli assi. Esso inoltre include una specificazione della posizione dell'angolo in basso a sinistra della barra, dell'altezza e della larghezza, e una conversione in un formato comprensibile per la funzione RectFill. Si noti che la APen, la BPen, il modo di disegno, e il pattern di riempimento devono essere correttamente specificati prima di richiamare la funzione DrawBar.

Sarebbe stato bello e appropriato che questo fosse una percentuale delle X e Y massime, come è stato fatto per i label degli assi, ma può essere fatto dal lettore se lo desidera.

```
/* drawbar.c */

DrawBar(rp,xyb,x,width,height)

      struct RastPort *rp;
      struct YXBase *xyb;
      LONG x, width, height;  /* dove va posta, quanto deve essere grande */
{
      WORD xmin, ymin, xmax, ymax;

      /* la barra deve essere adagiata all'asse x */
      /* si assume che i colori, il modo di disegno,e il Pattern
         siano già propriamente settati */

      xmin = x + xyb->xaxis;  /* posizione rispetto asse x */
      xmax = xmin + width - 1;

      ymax = xyb->yaxis - 1;  /* la fa giacere sull'asse x */
      ymin = ymax - height + 1;

      RectFill(rp,xmin,ymin,xmax,ymax);
}
```

Listato 4.8

Linee tratteggiate

Quando vengono tracciati gli assi, vengono disegnate delle rette solide. Se si desidera, si può applicare un pattern al tracciamento di linee, come lo si può applicare al riempimento di aree. Un pattern di linea è stabilito dalla funzione SetDrPt (si veda disegnare i pattern).

Il pattern in se stesso è una word di 16 bit priva di segno, contenente l'insieme di bit settati a 1 o a 0 che definisce la linea pattern. Per esempio, una linea tratteggiata appare così:

```
1100110011001100
```

Quando la linea è tracciata, se il modo di disegno è JAM1, dovunque vi siano degli uno nel pattern, sarà usato il colore della APen. Dove vi sono gli zero nel pattern, il colore dello sfondo o il colore del pattern, se c'è, resterà inalterato. Se il modo di disegno è JAM2, dovunque vi sia un uno nel pattern, sarà utilizzato il colore APen. Dove vi sono gli zero sarà utilizzata la BPen.

Per utilizzare il pattern per le linee tratteggiate sopra menzionato, si deve specificare un valore di pattern di linea di 0xCCCC. Se non si seleziona esplicitamente un pattern di linea, verrà usato il pattern di default. Il valore di default è 0xffff, che crea una linea 'solida'.

Disegnare più linee richiamando una sola funzione

Nel listato del main program che segue, delle linee tratteggiate sono usate per unire le estremità superiori di due barre dell'istogramma e una solida è usata per connettere la terza.

Il tracciamento delle linee è ottenuto richiamando una funzione che disegna più linee multiple connesse in una sola volta. La funzione è PolyDraw e per richiamarla di scriva

```
PolyDraw(rp,xytable,count);
```

dove rp è il puntatore alla RastPort; xytable è una tabella di words, delle quali ogni coppia definisce le coordinate di connessione in un insieme di linee connesse; e count definisce il numero di coppie presenti nella tabella xytable.

Il programma principale

Ora che tutte le parti sono state spiegate, possiamo passare al corpo principale del listato 4.9 che disegna un istogramma. Per evitare una ripetizione, il programma non commenta i singoli pezzi precedentemente commentati negli esempi precedenti.

Evitare di ridisegnare il contenuto di una window

Il programma per l'istogramma utilizza una window di tipo simple-refresh per il suo disegno. Questo significa che per ogni operazione di ridimensionamento, o per ogni operazione che porti un'altra window sopra quella del programma, è necessario ridisegnare la porzione che viene cancellata per riportare la window al suo aspetto originale.

Si può evitare tale operazione di redraw: invece di una window simple-refresh, si può selezionare una window smart-refresh o una window superbitmap.

Le window smart-refresh. Quando si seleziona una window smart-refresh, il sistema salva e ripristina automaticamente ogni parte della window che venga oscurata e poi rimessa a nudo dopo l'esecuzione di qualsiasi operazione fronte-retro. Vi sono dei prezzi per questo ripristino:

- Per salvare e ripristinare l'area oscurata è necessaria la stessa memoria che servirebbe per salvarla.
- Quando si disegna all'interno di una window smart-refresh, se viene oscurata una carta parte della window, le routine grafiche disegnano ciò che devono sia nella parte esposta sia in quella oscurata. Questo è insieme un vantaggio e uno svantaggio perché maggiore è la quantità di parti nascoste, maggiore è il tempo necessario al sistema per costruire un immagine. Ma vista l'estrema velocità con cui Amiga traccia i disegni un tale incremento di tempo potrebbe anche non essere notato per una sola window simple-refresh.

```
/* main barchart program */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "graphics/gfxmacros.h"

#define NORMALFLAGS (WINDOWSIZING|WINDOWDRAG|WINDOWCLOSE|WINDOWDEPTH)

#include "mypattern.h"
#include "multipat.h"
#include "xybase.h"
#include "window1.h"
#include "event01.c"
#include "drawaxes.c"
#include "drawbar.c"
#include "labelaxes.c"

struct Window *w;
struct RastPort *rport;

extern struct Window *OpenWindow();
int GfxBase;
int IntuitionBase;

main()
{
      struct IntuiMessage *msg;
      LONG result;
      w = 0;
      GfxBase = 0;
      IntuitionBase = 0;
      GfxBase = OpenLibrary("graphics.library",0);
      if(GfxBase == 0)
      {
            printf("graphics.library non si apre!\n");
            exit(10);
      }
      IntuitioanBse = OpenLibrary("intuition.library",0);
      if(IntuitionBase == 0)
      {
            printf("intuition.library non si apre!\n");
            exit(10);
      }

      w= OpenWindow(&mywindow);
      if(w == 0)
      {
            printf("la Window non si apre!\n");
            exit(10);
      }
      rport = w->RPort;
```

```
        redraw();               /* ridisegna per la prima volta */

        /* ora attende una messaggio dall'Intuition */
        /* (il task diviene inattivo aspettando il messaggio) */

        /* riprende il messaggio dalla porta */

        while(1)    /* 'per sempre ' */
        {
              WaitPort(w->UserPort);
              msg = (struct IntuitionMessage *)GetMsg(w->UserPort);
        handleit:
              result = Handleevent(msg->Class);
              if(result ==0)          /* ottiene un CLOSEWINDOW */
                    break;
              /* è possibile che l'Intuition invii più di un */
              /* messaggio al task mentre esso è in stato di attesa */
              /* così il loop può svuotare la porta ogni qualvolta */
              /* va a controllare il suo contenuto */

              msg = (struct IntuiMessage *)GetMsg(w->UserPort);

              if (msg != 0)           /* sarà 0 quando non vi saranno */
                                      /* più messaggi */
                    goto handleit;
        }

        ClosesWindow(w);
        CloseLibrary(GfxBase);
        CloseLibrary(IntuitionBase);

}
char *hlabels[] = {
                  " ", "84", "85", "86", "87" }
char *vlabels[] = {
                  "0", "10", "20", "30", "40" }

struct XYBase myxyb;
redraw()
{
        WORD i;

        DrawAxes(rport,myxyb,35,120,350,100,1);

        LabelHorizontal(rport,&myxyb,hlabels,5);
        LabelVertical(rport,myxyb,vlabel,5);

        SetAPen(rport,1);
        SetDrMd(rport,JAM1);

        for(i=1; i<5; i++)
        {
              DrawBar(rport,&myxyb,-31+i * 348/4,20,i * 12);
              /* dove, offset, posizioneX, larghezza, altezza */
        }
```

```
        SetApen(rport,1);
        SetBPen(rport,2);
        SetAfPt(rporyt,mypattern,3);  /* Pattern a due colori */
        SetOPen(rport,3);             /* il terzo colore è di Outline */
        SetDrMd(rport,JAM2);
        for(i=1; i<5; i++)
        {
              DrawBar(rport,&myxyb,-10+i * 348/4,20,i * 18);
              /* dove, offset, posizioneX, larghezza, altezza */
        }

        SetApen(rport,255);
        SetBPen(rport,0);
        SetAfPt(rport,mymulti,-3);    /* Pattern a due colori */
        SetOpen(rport,1);             /* il primo colore è di Outline */
        SetDrMd(rport,JAM2);

        for(i=1; i<5; i++)
        {
              Drawbar(rport,&myxyb,11+i * 348/4,20,i * 22);
              /* dove, offset, posizioneX, larghezza, altezza */
        }
}             /* fine del Redraw */
```

Listato 4.9

- Se si ridimensiona la window per renderla più piccola, il sistema salverà e ripristinerà solo quelle porzioni del disegno che si trovano all'interno dei bordi di limitazione della window. Quando si ridimensionerà la window rendendola più grande, il sistema farà apparire delle aree vuote solo intorno alla dimensione precedente. Si capirà allora che, quando si disegna all'interno di una window smart-refresh di una data dimensione, ogni disegno che vada al di fuori dei bordi di contenimento viene tagliato, cioè non viene per nulla disegnato. Così, una window smart-refresh, nonostante non debba rispondere agli eventi REFRESHWINDOW dell'intuition, può dover rispondere agli eventi NEWSIZE dell'Intuition stessa.

Per usare il tipo smart-refresh, si rimpiazzi il flag SIMPLE_REFRESH con il flag SMART_REFRESH nella struttura dati Newwindow. Poi si può eliminare il flag REFRESHWINDOW dalla routine di controllo degli eventi.

Se si vuole anche evitare che venga ridisegnato il contenuto della window a causa di un NEWSIZE, l'alternativa è semplice: si cancelli WINDOWSIZING dai flag della struttura dati NewWindow. Se non è presente il gadget per il ridimen-

sionamento, il sistema ignorerà i valori delle variabili di minima e massima dimensione nella struttura dati NewWindow e non provocherà nessun ridimensionamento.

Window SuperBitmap. Se si vuole poi essere liberi di ridimensionare la propria window senza dover rispondere ne agli eventi di REFRESHWINDOW ne a quelli di NEWSIZE, si può utilizzare una window SuperBitmap.

Quando si seleziona SUPER_BITMAP, si specifica che la propria area di disegno deve essere linkata al sistema. Qualunque cosa venga disegnata all'interno di una tale area verrà mantenuta qui per sempre. Qualunque cosa accada poi alla window il sistema preserverà sempre ciò che vi è all'interno. Questo significa che non sarà necessario rispondere ne agli eventi di REFRESHWINDOW ne a quelli di NEWSIZE.Anche questo ha il suo costo:

- Come per le window smart-refresh, questo metodo occupa una memoria maggiore di quella usata per le window simple-refresh. Inoltre si richiede che il programmatore conosca e fornisca una bitmap appositamente inizializzata e la memoria ad essa associata.
- Come per le window smart-refresh, questo metodo va a incrementare il tempo di tracciamento perché deve essere tacciata anche quella parte di disegno che non appare sullo schermo.

Un vantaggio di usare una window SuperBitmap è che la Bitmap può avere una dimensione di 1024-per-1024 pixel, e si può, se lo si desidera, posizionare la propria window in una qualsiasi posizione di questa grande mappa senza curarsi di quanto sia grande la window stessa. La posizione di default quando si apre all'inizio la window è tale che l'angolo superiore sinistro della mappa sullo screen e di quella fuori screen sono allineati.

Una window SuperBitmap deve anche essere una window GIMMEZEROZERO. Se non fosse GIMMEZEROZERO, l'Intuition posizionerebbe i gadget e i bordi della window sulla SuperBitmap, e se poi più tardi si andasse ad allargare la window o a scrollarla, i gadget e i bordi si troverebbero ancora lì.

Il listato 4.10 mostra le dichiarazioni di dati e i gruppi di istruzioni che possono essere inseriti nel programma dell'istogramma proprio prima della chiamata di

OpenWindow per creare la window SuperBitmap. Si noti che tali dichiarazioni vanno fatte nella parte di dichiarazione del main program.

Per prima cosa bisogna cambiare il flag SIMPLE_REFRESH nella struttura dati NewWindow in modo che si legga SUPER_BITMAP. Inoltre si devono rimuovere NEWSIZE e REFRESHWINDOW dalle variabili Flag IDCMP. Poi si inserisca l'aggiunta del listato 4.10 e si compili il programma.

```
/* superbitmap declaration (parte di programma) */

struct BitMap myBitMap;
extern PLANEPTR AllocRaster();
ULONG *m;

/* codice aggiuntivo per la SuperBitmap */

myWindow Width = 120;
myWindow.Height = 40;   /* la rimpicciolisce per la SuperBitmap */

/* permette alla SuperBitmap di essere full-screen */

InitBitMap(myBitMap,2,640,400);
                /* #piani, larghezza, altezza */

/* ora alloca la memoria per i piani di Bit associati all Bitmap */

m = AllocRaster(640,400);
if(m == 0)
{
            printf("Non c'è memoria per la SuperBitmap\n");
            exit(30);
}
myBitMap.Planes[0] = m;       /* si cura della prima delle due */

m AllocRaster(640,400);

if(m == 0)
{
            printf("Non c'è memoria per la SuperBitmap\n");
            FreeRaster(myBitMap.Planes[0],640,400);
            exit (30);
}

myBitMap.Planes[1] = m;       /* finisce di allocare la memoria */

/* Ora che la SuperBitmap è pronta per l'uso, la si può */
/* applicare alla struttura della NewWindow */
BitClear(myBitMap.Planes[0],(400*640)/8,0);
/* cancella la SuperBitmap altrimenti si avrebbe un brutto effetto */
```

```
myWindow.bitMap = &myBitMap;
FreeRaster(myBitMap.Planes[0],640,400);
/* all'uscita del programma */
```

Listato 4.10

Ora quando si farà girare il programma la window sarà più piccola. Però quando sarà ridimensionata il disegno apparirà nella sua totalità e non sarà richiesto nessun redraw.

Come menzionato, la posizione di default della window deve essere allineata con l'angolo superiore sinistro della sua SuperBitmap. Si può cambiare tale posizione aggiungendo e utilizzando la routine del listato 4.11.Poiché ScrollWindow usa una delle funzioni della Library dei Layer, tale libreria deve essere aperta prima che la funzione possa essere accessibile. Per questo si deve aggiungere, all'inizio del programma, l'insieme di istruzioni che aprono la libreria dei Layer:

```
/* dichiarazione dell'indirizzo base della Libreria di Layer */

LONG LayersBase;
/* apertura della libreria di Layer, si installi questo */
/* segmento dopo l'apertura della libreria di Intuition */

LayersBase = OpenLibrary(layers.library",0);

if(LayersBase == 0)
{
   printf("La libreria di Layer non si apre!\n");
   CloseLibrary(IntuitionBase);
   CloseLibrary(gfxBase);
   exit(40);
}
```

```
/* scrollwindow.c */

ScrollWindow(wi,dx,dy);
            struct Window *wi;
            SHORT dx,dy;
{
            struct RastPort *ra;
            struct LayerInfo *li;
            struct Layer *l;
```

```
            if(ra = wi->Rport)      /* setta il puntatore alla RastPort */
            {
                  if(l = ra->Layer) /* setta il puntatore al Layer */
                  {
                        if(li = l->LayerInfo)
                        /* setta il puntatore al LayerInfo */
                        {
                              ScrollLayer(li,l,dx,dy);
                        }
                  }
            }
}           /* fine di ScrollWindow */
```

Listato 4.11

Poi, alla fine del programma, quando vengono chiuse le Library, si inserisca:

```
CloseLibrary(LayersBase);
```

ScrollLayer è usata per riposizionare un Layer già esistente sopra la sua SuperBitmap. Se si specifica un valore di scroll che va oltre la massima distanza di scroll, il sistema automaticamente limita tale valore alla distanza massima disponibile. Il seguente insieme di istruzioni, aggiunte al proprio programma, fa muovere la window sopra la grande area di disegno utilizzata dalla SuperBitmap:

```
for(i = 0; i&LT40; i++)
{
   ScrollWindow(w,i,i);          /* la muove diagonalmente */
   Delay(5);                     /* un attesa di 1/10 di secondo */
}
for(i=39; i>0; i--)
{
   ScrollWindow(w,-i,-i);
   delay(5);                           /* la riporta indietro */
}
```

La libreria di Layer e utilizzata molto dall'intuition per creare, muovere, ridimensionare, e settare i piani di bit delle window che il programmatore intende costruire. Questo libro non si addentra tra le funzioni proprie della libreria di Layer. Per maggiori informazioni circa questa libreria, si veda il quinto capitolo del primo volume dell'Amiga Programmer's Handbook.

Nel quinto capitolo di questo libro, si troveranno descritti i gadget corrispondenti. Si potranno utilizzare alcuni di essi per controllare quella parte della Su-

perBitmap che è visibile nella window. Ma per adesso, proseguiamo con la grafica.

Progettare e aprire uno screen personalizzato.

In questo paragrafo, si creerà uno screen personalizzato, cioè uno screen nel quale la scelta dei colori è fatta dal programmatore. Invece di essere vincolati ai quattro colori che fornisce il Workbench, si può specificare il proprio set composto di un numero di colori che può arrivare a 32. Su questo screen, si creerà una mappa utilizzando un certo numero delle funzioni di visualizzazione di Amiga.

Si devono specificare i seguenti parametri per permettere al sistema di aprire un nuovo screen:

- Dove posizionare la screen in relazione all'area di visualizzazione (dove porre il suo angolo superiore sinistro).
- Quanto deve essere grande (quanto largo e quanto alto)
- Quale titolo di default deve esserci sulla sua barra
- Che colori utilizzare per disegnare i contorni e i gadget di sistema
- Le caratteristiche dei font che devono essere utilizzati dai Menu, dalle window, dalle intestazioni e così via
- Il tipo di screen (in questo caso CUSTOMSCREEN) che si vuole creare
- Quanti colori possono essere visualizzati su questo screen
- Quale modo particolare di visualizzazione deve essere usato

Definire uno screen personalizzato

I parametri sopra mostrati devono essere definiti in una struttura dati chiamata NewScreen. Il listato 4.12 è la definizione di questa struttura dati e sarà usata nel programma Map.

```
/* myscreen1.h */

/* myfont1 specifica le caratteristiche del Font di Default; */
/* un font da 80-colonne che viene visualizzato a 40 */
/* essendo lo schermo a bassa risoluzione */

struct TexAttr myfont1 =
{
            "topaz.font", 8, 0, 0
};

struct NewScreen myscreen1 =
{

            0, 0,       /* posizione dello screen */
            20,200,     /* dimensioni dello screen */
            5,          /* # piani di Bit, significa che lo screen */
                        /* possiederà 2 alla quinta (32) colori */
                        /* tra i quali scegliere */
            1,0         /* DetailPen, BlockPen */
            0,          /* modo di visualizzazione */
                        /* 0 = bassa risoluzione */

            CUSTOMSCREEN      /* tipo di screen */

            &myfont1,         /* font di default dello screen */

            "32 Color Test" /* Titolo di default dello screen */

            NULL,       /* i Gadget dello screen sono ignorati */

            NULL,       /* indirizzo della Bitmap dello screen */
                        /* non utilizzato in questo esempio */
};
```

Listato 4.12

Aprire lo screen personalizzato

Si può aprire lo screen personalizzato mediante la funzione OpenScreen. Ecco come si richiama la funzione OpenScreen:

```
struct Screen *s                    /* dichiara un puntatore
                                     alla struttura Screen */

s = OpenScreen(&myscreen1);         /* cerca di aprirlo */

if(s == 0)
{
   printf("Non riesco ad aprire myscreen1\n");
   exit(10);
}
```

Se questa funzione restituisce un valore nullo, lo screen non si è aperto. Dando per scontato che siano stati specificati tutti i parametri correttamente nella struttura dati NewScreen, la ragione per cui più comunemente uno screen non si apre è la mancanza di memoria nel sistema. Può darsi che si tenti di far girare troppi programmi nello stesso tempo e si deve magari chiuderne qualcuno per far girare questo programma.

Aprire una window sullo screen personalizzato

Il listato 4.13 è una versione modificata della struttura NewWindow definita all'inizio del capitolo. Le istruzioni sottostanti definiscono alcune variabili, specificando cosa viene cambiato e perché. Tra queste istruzioni, una volta che viene aperto uno screen, il puntatore allo screen della struttura NewWindow deve essere cambiato in modo opportuno, per riferire al sistema in quale screen la window deve essere aperta.

```
myWindow.Screen = s;        /* punta allo schermo personalizzato */
               /* e avverte il sistema che non si tratta dello */
                            /* screen della Workbench */
myWindow.Type = CUSTOMSCREEN;

/* poi, usa le normali chiamate di funzioni e i normali */
/* controlli d'errore di OpenWindow come mostrato */
/* all'inizio del capitolo */
```

Se lo screen e la window si aprono entrambi correttamente, si possiede uno screen e una window personalizzati sul proprio schermo.

```
/*window2.h */

struct NewWindow myWindow =
{
      0,          /* margine sinistro della Window misurato in pixel */
                  /* alla risoluzione orizzontale corrente */
                  /* dall'estremità sinistra dello schermo */
      15,         /* margine superiore della window misurato in */
                  /* linee a partire dall'alto dello schermo */
      280,150,    /* larghezza e altezza di questa window */
      0,          /* DetailPen - è il numero di penna da usare */
                  /* per disegnare i contorni della window. */
      1,          /* BlockPen - è il numero di penna da usare */
                  /* per i Gadget di sistema della window */

      CLOSEWINDOW | NEWSIZE | REFRESHWINDOW /* i Flag IDCMP */

      SIMPLE_REFRESH | NORMALFLAGS | GIMMEROZERO /* Flag di window */

      NULL,
      NULL,

      "Sample Chart",   /* nome della window */

      NULL,             /* puntatore allo screen se non */
                              /* è lo screen del Workbench */

      NULL,             /* puntatore alla Bitmap se si tratta */
                              /* di una window SuperBitmap */

      10,10             /* minima larghezza e minima altezza */
      320,200                 /* massima larghezza e massima altezza */

      WBENCHSCREEN            /* tipo di screen nel quale aprirla */

};
```

Listato 4.13

Settare i colori

Ora che si possiede uno screen proprio, si possono definire i colori tra i quali si potrà scegliere per disegnare. Invece di essere limitati ai colori del Workbench,

ora si possiede una palette personalizzata da modificare a proprio piacimento per le proprie operazioni .

La struttura dati NewScreen specifica che si può scegliere tra 32 diversi colori. Si stabilisce la palette personalizzata per mezzo della funzione SetRGB4 per i singoli colori e della funzione LoadRGB4 per i colori multipli che devono essere settati con una sola chiamata di funzione.

Per utilizzare l'una o l'altra di queste funzioni, si deve recuperare, dalla struttura dati Screen, il puntatore alla porta di visualizzazione (ViewPort) che è stata inizializzata per lo screen nella fase di apertura dello screen stesso. Ecco le istruzioni per stabilire un puntatore alla porta di visualizzazione:

```
struct ViewPort *vp;

xp = &(s->ViewPort);        /* lo schermo contiene una ViewPort */
```

Ora questo puntatore alla porta di visualizzazione può essere utilizzato sia dalla funzione SetRGB4 sia dalla LoadRGB4. Per richiamare SetRGB4 si scriva:

```
SetRGB4(vp,colorNumber,rValue,gValue,bValue);
```

dove vp è un puntatore alla ViewPort; colorNumber è il registro di colore che deve essere caricato con i valori RGB del colore descritti dai rimanenti parametri; e rValue, gValue, e bValue sono i valori del rosso, del verde e del blu (possono assumere un valore che va da 0 a 15).

Ecco tre esempi che illustrano la chiamata della funzione SetRGB4:

```
/* setta il colore dello sfondo, cioè il colore 0 */
/* in modo che sia nero (r=0, g=0, b=0) */
SetRGB4(vp,0,0,0,0);

/* setta il colore numero 1 in modo che sia un rosso intenso */
SetRGB4(vp,1,15,0,0);

/* setta il colore numero 2 in modo che sia bianco */
/* tutti i valori RGB sono al massimo */
SetRGB4(vp,2,15,15,15);
```

Si può usare la funzione SetRGB4 per far variare i colori ciclicamente all'interno di uno o più registri di colore, in modo da dare una sorta di effetto di animazione. LoadRGB4 è usato più spesso per settare colori di uno screen personalizzato subito dopo che lo schermo è stato creato.

Per richiamare LoadRGB4 si scriva:

```
LoadRGB4(vp,coloTable,howmany);
```

dove vp è un puntatore alla ViewPort; colorTable è un puntatore alla tabella di colori che deve essere caricata; e howmany specifica quanti colori ci sono in questa tabella.

LoadRGB4 parte sempre a caricare dal colore numero zero e procede a caricare i registri che gli sono stati specificati. I colori sono sempre formulati utilizzando numeri privi di segno a 16-Bit con quattro Bit riservati a ogni colore. I Bit sono ordinati in questo modo:

```
0000 RRRR GGGG BBBB
```

dove i primi quattro Bit sono ignorati, i quattro Bit seguenti rappresentano il valore del rosso, quelli seguenti il valore del verde, e gli ultimi quattro il valore del blu.

Si possono caricare sino 32 valori di colore nella ViewPort del nuovo screen. Ecco un esempio di tavolozza, contenente un totale di 16 valori. I nomi dei colori che essi rappresentano si trovano nei commenti.

```
UWORD mycolortable[] = {

0x0000,0x0e30,0x0fff,0x0b40,0x0fb0,0x0bf0,
0x05d0,0x0ed0,0x07df,0x069f,0x0c0e,
0x0f2e,0x0feb,0x0c98,0x0bbb,0x0df
};

/* nero, rosso, bianco, rosso fuoco, arancione, giallo, */
/* verde chiaro, verde, verde acqua, blu scuro, porpora */
/* violetta, bronzo, marrone, grigio, blu cielo */
```

Per richiamare con la funzione LoadRGB4 questa tabella sul proprio screen si scriva

```
LoadRGB4(vp,&mycolortable[0],16);
```

Determinare il colore correntemente in uso

Questo sistema conserva una tavolozza per ogni screen che viene creato. Se non si carica una propria tabella di colori, o se non si caricano tutti i 32 valori, il sistema li caricherà automaticamente da una tabella di default quando lo screen viene aperto. Se lo screen è stato aperto dall'intuition (cioè con una chiamata di OpenScreen), si può entrare nella ViewPort dello screen per determinare il colore assegnato a un particolare registro della tabella. La routine che fa ciò e GetRGB4. Per richiamarla si scriva:

```
value = GetRGB4(vp,entry);
```

dove vp è un puntatore all'indirizzo della ViewPort dello screen, e entry è il numero del registro di colore al quale si è interessati. Il valore restituito è -1, se quel numero di entry non contiene un valore valido, o l'attuale valore RGB con i quattro bit per l'intensità del rosso, del verde, e del blu, dove i quattro Bit più significativi del valore restituito contengono 0, i seguenti quattro il valore del rosso, quelli dopo il valore del verde, e gli ultimi quattro il valore del blu.

Riempire con un colore delle figure

Amiga contiene la routine Flood per riempire una figura con un colore,con un effetto a inondazione, a partire da una specifica posizione dell'area di disegno. Vi sono due modi per utilizzare questo tipo di riempimento. Il modo 0 riempie la figura a partire dalla posizione corrente della penna e spargendo il colore in tutti i pixel adiacenti, non fermandosi finché non trova dei punti che abbiano un colore pari a quello di Outline (quello settato con la funzione SetOPen). Il modo 1 riempie la figura a partire dalla posizione corrente della penna e continua a riempire i pixel ad essa adiacenti che abbiano però lo stesso colore del primo pixel che è stato riempito.

La cosa interessante a proposito della funzione Flood, e che essa utilizza il modo di disegno corrente (JAM1 o JAM2) e il pattern, se esso c'è. Così si può creare un'area racchiusa di ogni figura, e riempirla con pattern multicolore se lo si desidera. Se si utilizza questa funzione, bisogna essere sicuri che la figura che si vuole riempire non abbia interruzioni nei suoi contorni. Ogni qualvolta, infatti, vi sia un pixel mancante nella figura, la funzione uscirà dai bordi della figura stessa e inonderà tutta la window o addirittura l'intero screen con il pattern di riempimento.

Per richiamare la funzione flood si scriva:

```
Flood(rp,mode,x,y);
```

dove rp è un puntatore alla RastPort; mode è il numero di modo di riempimento; e x,y sono le coordinate del punto dal quale deve partire il riempimento.

Il listato 4.14 utilizza la funzione Flood in una routine per creare una figura riempita con un pattern multicolore.

Riempire aree di figure bizzarre

In aggiunta a Flood, Amiga mette a disposizione nel suo Software una serie di altre funzioni per il riempimento delle aree di disegno. Con queste funzioni, prima si definisce l'intera figura, poi si dice al sistema di riempirla, a meno che non si disegni un poligono formato da rette e si richiami la funzione Flood.

```
/* drawdiamond.c */

DrawDaimond(rport,xcenter,ycenter,xsize,ysize)
            struct rastPort *rport;
            WORD xcenter, ycenter, xsize, ysize;
{
            BYTE oldAPen;
            WORD xoff, yoff;

            oldAPen = rport->FgPen /* salva il vecchio valore di APen */
```

```
        SetAPen(rport,rport->AOlPen);
                        /* stesso colore della penna di Outline */

        xoff = xsize/2;   /* centratura */
        yoff = ysize/2;

        /* disegna la figura del diamante */

        Move(rport,xcenter - xoff,ycenter);
        Draw(rport.xcenter,ycenter + yoff);
        Draw(rport,xcenter + xoff,ycenter);
        Draw(rport,xcenter,ycenter - yoff);
        Draw(rport,xcenter - xoff,ycenter);

        /* riempimento a partire dal centro della figura */

        Flood(rport,0,xcenter,ycenter);

        SetAPen(rport,oldAPen);
              /* ripristina il valore precedente di APen */
}
```

Listato 4.14

Il vantaggio dell'uso delle funzioni di riempimento delle aree è che esse automaticamente sanno operare su qualsiasi strana figura l'utente abbia disegnato, senza il pericolo che una smagliatura nel bordo della figura stessa causi il riempimento totale dell'area di disegno.

Ecco le funzioni che vengono usate per il riempimento di aree specifiche:

```
error = AreaMove(rp,x,y);
error = AreaDraw(rp,x,y);
AreaEnd(rp);
```

Il parametro rp è un puntatore alla RastPort, e x e y sono le coordinate di Move o Draw.

Un valore restituito di -1 implica che non vi era più spazio nel buffer per maneggiare queste particolari AreaMove o AreaDraw; se non avvengono errori, viene restituito un valore nullo.

AreaMove opera in una maniera simile a Move, nel senso che essa significa "prendi la penna di disegno e spostala da un'altra parte". AreaDraw opera in mo-

do simile a Draw, cioè essa significa "disegna una linea di contorno (con o senza una penna di Outline per la figura finale) dalla posizione corrente della penna a una nuova posizione specificata". Ne AreaMove ne AreaDraw hanno alcun effetto sulla posizione corrente della penna usata da Draw e da Move.

Si possono utilizzare più funzioni AreaMove e AreaDraw, ognuna delle quali definisce una propria figura. Quando alla fine si richiama AreaEnd, tutte le figure che sono state definite vengono disegnate in una sola volta. Nessuna linea e nessuna figura viene tracciata finché non viene richiamata la funzione AreaEnd.

Quando si richiama AreaMove, ogni precedente figura disegnata da una serie di chiamate di AreaDraw viene automaticamente chiusa come se si fosse chiamata AreaDraw una volta ancora, specificando le coordinate del primo punto di quella figura. Per esempio, per disegnare un quadrato pieno, è necessario solo chiamare AreaMove per spostarsi dal primo angolo e poi richiamare AreaDraw per disegnare gli altri tre angoli. La successiva chiamata di AreaMove (o AreaEnd), automaticamente disegna il lato che chiude il quadrato.

Requisiti necessari per compiere le operazioni sulle aree. Quando si richiama la funzione AreaDraw o la funzione AreaMove, il sistema costruisce una lista delle operazioni di tracciamento e riempimento. Quando si richiama la funzione AreaEnd, il sistema elabora questa lista, disegnando e riempiendo le figure. Ci sono un paio di cose che devono essere fatte prima di richiamare queste funzioni per preparare la propria RastPort perché sia utilizzata dal sistema. Si devono fornire le cose seguenti:

- Una struttura dati AreaInfo per la RastPort in questione e una corrispondente area dati per la struttura AreaInfo
- Una struttura dati TmpRas per la RastPort in questione e uno spazio di memoria corrispondente

La struttura dati AreaInfo contiene le variabili che il sistema utilizza per mantenere una traccia delle richieste di AreaMove e AreaDraw. Essa contiene anche un puntatore alla zona di memoria che viene utilizzata per salvare le richieste. Si ricordi che nessun riempimento di aree ha luogo finché non si richiama la funzione AreaEnd. Tutte le richieste intermedie di AreaDraw e AreaMove vengono immagazzinate in un array della memoria che deve essere fornito. Esse sono poi eseguite tutte insieme quando viene richiamata AreaEnd.

Si deve fornire un Array di word a 16-bit dove il sistema possa immagazzinare la richieste che gli vengono fatte. L'array deve contenere un numero di word cinque volte superiore al numero totale di chiamate di AreaMove e AreaDraw che si vogliono effettuare prima di richiamare AreaEnd. Se si stanno creando le figure una alla volta, si devono fornire solo tanti punti quanti sono i vertici del poligono costituente la figura che si sta disegnando (più un paio di punti per sicurezza). Per questo esempio, si fornisce un massimo di 20 punti. Si deve inizializzare una struttura dati AreaInfo mediante l'uso della funzione InitArea, passandole l'indirizzo della propria struttura AreaInfo, l'indirizzo dell'Array nel quale il sistema memorizzerà le richieste, e il massimo numero di punti permessi:

```
WORD areaarray[100];        /* 20 punti, 5 word per punto */

struct AreaInfo myAreaInfo;

Init(&myAreaInfo,&areaArray[0],20);

rp->AreaInfo = &myAreaInfo;      /* linkalo alla RastPort */
```

Quando il sistema comincia a eseguire le richieste dell'utente per il riempimento delle aree, essa ha bisogno di un area di lavoro nella quale costruire e riempire la figura prima che la figura stessa venga posta nell'area di disegno corrente specificata dall'utente. La figura è costruita in ciò che è chiamata la TmpRas.

Il numero di word dati richiesto per una struttura dati TmpRas è definito dalla più grande figura rettangolare che si vuole creare.Per esempio, se la figura più grande è alta 200 linee e larga 640, allora è necessario uno spazio nel quale far stare 640x200 bit.Solo il numero vero del prodotto linee verticali per linee orizzontali è preso in considerazione. Cioè, la memoria è riutilizzata, per ogni operazione di riempimento, in maniera appropriata alla forma dell'oggetto e non forzata dal valore delle coordinate rettangolari sulle quali si era basata l'allocazione della memoria. Per esempio, se si fornisce un spazio sufficiente a contenere una figura che stá in un rettangolo di 100x100, essa può essere più tardi riutilizzata per una figura che sta in rettangolo di 1000x10.

Per prima cosa si deve allocare lo spazio di utilizzo della memoria, poi si deve utilizzare la funzione InitTmpRas per inizializzare la struttura dati TmpRas, e poi si deve linkarla alla RastPort. Ecco una sequenza tipica:

```
PLANEPTR workspace;

struct TmpRas myTmpRas;

workspace = AllocRaster(640,200);
if(workspace == 0)
{
   printf("Non c'è spazio per il Raster temporaneo!\n");
}
else
{
   InitTmpRas(&myTmpRas,workspace,RASSIZE(640,200));
   rp->TmpRas = &myTmpRas; /* da linkare alla RastPort */
}
```

La macro RASSIZE è utilizzata per informare il sistema su quante word sono state riservate dalla chiamata di AllocRaster. Il sistema, poi, saprà quanto può essere grande la figura da creare in base alla grandezza di quest'area.

Prima di usare la funzione Text, si deve fornire una struttura TmpRas per la propria RastPort, perché se Text non ha a disposizione questa struttura, essa deve allocare e disallocare una certa quantità di memoria ogni volta che viene richiamata. Fornire una TmpRas risulta più efficiente e quindi permette al sistema di operare più velocemente. L'uso della funzione Text è descritto più avanti in questo stesso capitolo.

Disegnare e leggere i singoli pixel

A volte, si desidera disegnare o leggere il colore di un singolo pixel. Le routine grafiche di Amiga includono le funzioni WritePixel e ReadPixel proprio per queste operazioni. Per richiamare tali funzioni si scriva:

```
WritePixel(rp,x,y);
penNumber = ReadPixel(rp,x,y);
```

dove rp è un puntatore alla RastPort; e x e y sono le coordinate del pixel da scrivere o da leggere.

Per WritePixel, il colore che viene disegnato è quello della APen. Per ReadPixel, il valore restituito, penNumber, va da 0 a 255. Se non è possibile leggere alle coordinate specificate (magari perché vanno al di fuori delle coordinate della RastPort corrente), viene restituito un valore -1.

Nel programma Map che segue, WritePixel e ReadPixel vengono usate in combinazione con la funzione di Amiga per la generazione di numeri Random per movimentare la mappa visibile.

Disegnare una mappa

Il programma Map del listato 4.15 crea una mappa geografica dello Utah, del Colorado, dell'Arizona, e del NewMexico. Da un colore diverso ad ogni stato, e poi scrive il nome dei singoli stati con una abbreviazione di due lettere. Una volta che gli stati sono disegnati, il programma 'fa nevicare' sull'Arizona. Per far partire l'effetto neve, si schiacci il tasto di destra del mouse.

I testi

Fino ad ora abbiamo utilizzato il font di Default a 80 colonne per tutti i testi che sono stati scritti. Ora si vedrà come creare effetti particolari per i testi e come utilizzare tutti gli altri font disponibili.

Per muovere correttamente il cursore a prescindere dal font che si sta utilizzando, si deve sapere come trovare l'altezza, la larghezza, e la linea di base del testo. E per utilizzare font diversi, si deve sapere come scegliere i singoli font e come richiedere gli stili bold, italics, plain o reverse.

```
#include "exec/types.h"
#include "intuition.intuition.h"

#define AZCOLOR 1       /* I numeri dei colori da usare per l'Arizona */
#define WHITECOLOR /* e per la neve */
```

```
#include "myscreen.h"
#include "window2.h"
#include "graphics/gfxmacros.h"
#include "evento1.c"          /* carica il controllore degli eventi */

/* definisce la posizione iniziale dei 4 angoli degli stati */

#define CORNERX 150
#define CORNER 5

struct Window *w;       /* puntatore a una window */
struct RastPort *rp;    /* puntatore a una RastPort */
struct Screen *s;       /* puntatore a uno screen */
struct ViewPort *vp;    /* puntatore a una ViewPort */

struct AreaInfo myAreaInfo;
PLANEPTR workspace;
struct TmpRas myTmpRas;

struct Window *OpenWindow();
struct Screen *OpenScreen();

LONG GfxBase;
Long IntuitionBase;

WORD areaArray[100];    /* 20 punti, 5 word per punto */

Word utahxy[] =
            {
            0, 0, -40,0, -38,-70, -15,-70, -1,55, 0,-55, 0,0
            };

Word coloradoxy[] =
            {
            0,0 75,0, 75,-55, 0,-55
            };

Word arizonaxy[] =
            {
            0,0, -40,0, -40,10, -50,10, -50,30, -60,55, -30,70, 0,70
            };

Word newmexicoxy[] =
            {
            0,0, 0,70, 8,70, 68,70, 68,70
            };

UWORD mycolortable[] =
            {
            0x0000, 0x0e30, 0x0fff, 0x0b40, 0x0fb0, 0x0bf0,
            0x05d0, 0x0ed0, 0x07df, oxo69f, 0x0c0e,
            0x0f2e, 0x0feb, 0x0c98, 0x0bbb, 0x07df
            };
```

```
/* nero, rosso, bianco, rosso fuoco, arancione, giallo, */
/* verde vivo, verde, verde acqua, blu scuro, porpora */
/* violetto, bronzo, marrone, grigio, blu cielo */

#include "drawdiamond.c"

main()
{
      struct IntuiMessage *msg;
      LONG result;
      PLANEPTR workspace;
      WORD rx, ry;

      GfxBase = OpenLibrary("graphics.library",0);
      IntuitionBase = OpenLibrary("intuition.library",0);
      /* (per abbreviare sono tralasciati i controlli di errore) */
      /* (si deve controllare che nessun OpenLibrary restituisca uno 0) */

      s = OpenScreen(&myscreen1); /* si cerca di aprirlo */
      if(s == 0)
      {
            printf("Non posso aprire myscreen1\n");
            exit(10);
      }
      myWindow.Screen = s;
            /* informa circa la posizione dello schermo */
      myWindow.Type = CUSTOMSCREEN;
            /* dice all'Intuition di guardare al W.Screen */

      w = OpenWindow(&myWindow);
      if(w == 0)
      {
            printf("La window non si è aperta!\n");
            CloseScreen(s);
            exit(20);
      }
      vp = &(s->ViewPort);

      /* setta i colori per questa ViewPort */

      LoadRGB4(vp,mycolortable[0],16);

      rp = w->RPort;

      workspace = (PLANEPTR)AllocRaster(640,200);
      if(workspace == 0)
      {
            printf("Non c'è spazio per il Raster provvisorio\n");
            CloseWindow(w);
            CloseScreen(s);
            exit(30);
      }
      InitTmpRas(myTmpRas,workspace,RASSIZE(640,200));
      rp->TmpRas = &myTmpRas; /* lo linka alla RastPort */
```

```
InitArea(&myAreaInfo, &areaArray[0],20);
rp->AreaInfo = &myAreaInfo;   /* lo linka alla RastPort */
redraw();                     /* per la 'prima' volta */

/* Ora attende un messaggio dall'Intuition */
/* (il task si pone in stato di inattività durante l'attesa) */

WaitPort(w->UserPort);

/* Preleva il messaggio dalla porta */

while(1)    /* 'per sempre' */
{
      msg = (struct IntuiMessage *)GetMsg(w->UserPort);
handleit:
      result = HandleEvent(msg->Class);

      if(result == 0)   /* vi è una CLOSEWINDOW */
            break;
      msg = (struct IntuiMessage *)GetMsg(w->UserPort);


      if(msg != 0)            /* sarà 0 quando */
                              /* non vi saranno più messaggi */
            goto handleit;
      /* Normalmente per interfacciarsi correttamente con il */
      /* Multitasking, ci sarebbe qui una istruzione */
      /* con l'effetto di     */
      /*                      */
      /*          WaitPort(w->UserPort)*/
      /*                      */
      /* per porre in stato di inattività il task in */
      /* attesa di un altro messaggio finché non sono */
      /* stati elaborati tutti i rimanenti. Ma in questo caso, il task */
      /* deve rimanere operativo per far nevicare */
      /* sull'Arizona, così la WaitPort non è stata */
      /* inserita             */

            /* L'Arizona esiste come una strana figura contenuta */
      /* in un rettangolo che va da (0,0) a (-60,70). Vogliamo */
      /* movimentare questo stato con della neve, senza per */
      /* altro far nevicare all'esterno dei suoi confini */
      /* così preleviamo dei valori random per la x e la y, */
      /* e poi utilizziamo la funzione ReadPixel per vedere */
      /* quel punto ha il colore usato per colorare l'Arizona, */
      /* se la risposta è positiva vi poniamo il bianco */
      /* con l'uso di WritePixel */

      rx = CORNERX - RangeRand(60);
      ry = CORNERY + RangeRand(70);

      if(ReadPixel(rp,rx,ry) == AZCOLOR)
      {
            SetAPen(rp,WHITECOLOR);
```

```
                WritePixel(rp,rx,ry);
            }
        }
        /* fatto! adesso cancelliamo */

        CloseWindow(w);
        CloseScreen(s);
        FreeRaster(workspace,640,200);
}

afill(w,pairs)
        WORD *w;          /* puntatore a una word */
        WORD pairs; /* quante coppie di word */
{
        WORD i;
        AreaMove(rp,CORNERX+w[0],CORNERY[1]);
        w++; w++;
        for (i=1; i<pairs; i++)
        {
            AreaDraw(rp,CORNERX+w[0],CORNERY+w[1]);
            w++; w++;
        }
        AreaEnd(rp);
}
redraw()
{
        SetDrMd(rp,JAM1);
        SetAPen(rp,1);
        afill(coloradoxy[0],4);

        SetAPen(rp,5);
        afill(newmexicoxy[0],5);

        SetAPen(rp,AZCOLOR);
        afill(arizonaxy[0],8);

        SetOPen(rp,12);
            /* quando opera il Redraw, aggiunge i bordi agli stati */
        SetAPen(rp,3);
        DrawDiamond(rp,20,20,20,10);
        SetAPen(rp,8);
        DrawDiamond(rp,20,120,20,10);

        SetAPen(rp,6);
        SetBPen(rp,0);
        SetDrMd(rp,JAM1);

        /* scrive nomi stati */
        Move(rp,CORNERX-30,CORNERY-20);
        Text(rp,"UT",2);

        Move(rp,CORNERX-30,CORNERY+30);
        Text(rp,"AZ",2);

        Move(rp,CORNERX+35,CORNERY-20);
```

```
        Text(rp,"CO",2);

        Move(rp,CORNERX+35,CORNERY+30);
        Text(rp,"MN",2);
}
```

Listato 4.15

Questi sono gli argomenti trattati in questo paragrafo.

L'altezza del testo può essere trovata all'interno della variabile TxHeight della struttura RastPort. Se si possiede un puntatore alla RastPort di nome rp, si può accedere all'altezza del testo scrivendo:

```
myTextHeight = rp->TxHeight;
```

La larghezza nominale del testo, che sarebbe la larghezza del rettangolo entro il quale è racchiuso il carattere regolare (non bold o italic) è accessibile mediante un'istruzione di questo tipo :

```
myTextWidth = rp->TxWidth;
```

Si può voler utilizzare i valori dell'altezza e della larghezza del testo per produrre un cursore della stessa dimensione del testo sul quale lo si sta posizionando.

Il valore della linea base del testo è memorizzato nella variabile TxBaseline della struttura RastPort. Si accede a questo valore così:

```
myTextBaseline = rp->TxBaseline;
```

La linea base è quella retta immaginaria sulla quale si appoggiano i caratteri del testo. Alcuni font hanno delle parti che vanno sotto tale linea (discendenti), come ad esempio le lettere y e g. La linea base è calcolata a partire dalla cima della cella standard (il rettangolo che da l'altezza e la larghezza del testo).

Per posizionare un carattere in modo tale che l'angolo superiore sinistro della sua cella si trovi in una certa posizione, si deve eseguire la funzione:

```
Move(rp,X,Y+Baseline);
```

Poi, quando si userà la funzione Text il carattere si troverà dove ci si aspetta.

Aprire un font

Vi sono due tipi differenti di font: quelli residenti in ROM e quelli residenti su disco. Il software di Amiga fornisce due routine per accedere a questi font. Una è OpenFont, per i font residenti in ROM. L'altra è OpenDiskFont, per quelli che si trovano su disco. In pratica, però, la routine OpenDiskFont è l'unica che si deve utilizzare perché è in grado di aprire entrambi i tipi di font, cioè anche quelli ROM. Essa utilizza le informazioni contenute nella lista dei font di sistema per determinare dove trovare un font e se i suoi dati si trovano già in memoria o se devono essere caricati.

Se si vuole utilizzare un font diverso da quello a 80 o 60 colonne che è stato specificato nelle preferences, oppure si vuole utilizzare un font diverso da quello dichiarato nella struttura del NewScreen, si deve aprire tale font, e bisogna selezionarlo per la RastPort in uso.

Aprendo un font residente in ROM lo si rende immediatamente disponibile per l'uso, poiché viene restituito alla routine richiamante un puntatore alla struttura dati che descrive e controlla il font. Aprendo un font che si trova su disco si fa la stessa cosa, ma in più, lo si carica fisicamente dal disco, se esso è presente nella directory di nome FONTS:. Così, una volta aperto, un font diventa disponibile per l'uso.

Per richiamare OpenFont e OpenDiskFont si scriva:

```
font = OpenFont(&ta);
font = OpenDiskFont(&ta);
```

dove font è un puntatore a una struttura dati TextFont restituito dalla routine se è risultato possibile aprire il font; e &ta è l'indirizzo della struttura dati TextAttr.

Definire gli attributi di un testo. Il nome, la dimensione, e lo stile di ogni font sono contenuti nelle variabili della struttura dati TextAttr:

- ta_name è l'indirizzo della stinga null-terminated che contiene il nome del font così come esso appare nella directory FONTS (per esempio, Garnet).
- ta_YSize è l'altezza nominale del font espressa in linee. Nelle subdirectory con i nomi dei font della directory FONTS ci sono i nomi dei file che forniscono le varie altezze dei font (per esempio, Garnet/9, è una versione allungata verticalmente del font garnet). I dati qui presenti esprimono in annotazione binaria i font stessi.
- ta_Style è lo stile primario del font. Alcuni font saranno in italics, altri in bold e saranno memorizzati come tali. Altri font possono essere manipolati per apparire in stili diversi.
- ta_Flags contiene le Preferences del flag. Le Preferences di font sono quei flag che possono essere settati per chiedere al sistema di esaudire determinate richieste. Per esempio, se si vuole utilizzare un font che è stato progettato per l'uso in alta risoluzione, non interlacciata, si può settare il flag FPP_TALLDOT. Se ci sono due font nel sistema che hanno lo stesso nome, e la stessa YSize, ma uno dei quali è stato progettato per l'uso ad alta risoluzione, sarà caricato proprio quest'ultimo poiché si è settato il flag corrispondente.

Ecco un esempio di struttura per gli attributi di testo:

```
struct TextAttr myAttr = {"garnet.font",9,0,0};
```

Ecco un esempio che richiama OpenDiskFont:

```
struct TextFont *tf;

tf = OpenDiskFont(myAttr);
```

Stabilire il font di RastPort. Si specifica quale font sarà usato da una certa RastPort per le chiamate di Text, utilizzando la routine SetFont. Per richiamare SetFont si scriva:

```
SetFont(rp,tf);
```

dove rp è un puntatore a una RastPort, e tf è un puntatore a una struttura dati TextFont restituito da OpenFont o OpenDiskFont. Dopo aver settato il font, la funzione Text utilizzerà sempre tale font.

Aggiungere un font alla lista di sistema dei font. Ci sono due font già presenti sulla lista di sistema dei font (selezionabile tramite le Preferences), di nome topaz.font di altezza 8-linee (il font da 40/80 colonne) e il topaz.font di altezza 9-linee (il font da 32/64 colonne). Per quanto riguarda gli altri font, se si vuole che siano accessibili da tutti i task correntemente attivi, si deve aggiungerli alla lista di sistema usando la funzione AddFont. Per richiamare AddFont si scriva

```
AddFont(tf);
```

dove tf è un puntatore a una struttura dati TextFont ritornato dalla routine OpenFont o dalla OpenDiskFont.

Se due task devono utilizzare lo stesso font, ognuno di essi dovrà usare OpenDiskFont per ottenere il puntatore alla TextFont. La prima cosa che il sistema fa in questi casi è guardare se un altro task ha già caricato questo font. Se è così, incrementa di uno il contatore degli utilizzatori del font e restituisce il puntatore alla struttura dati TextFont al richiamante. Se il font non è stato aggiunto alla lista di sistema dei font, allora i dati riguardanti quel font saranno caricati di nuovo, occupando inutilmente dell'altra memoria.

Quali font sono disponibili. Normalmente, un programmatore saprà quali font sono disponibili per l'uso e può richiedere quello specifico mediante il nome e le caratteristiche reali. Le routine OpenFont e OpenDiskFont cercano di fungere da maggiorazione delle preferences e possono caricare qualcosa di molto simile al font richiesto se questo non fosse disponibile nella directory FONTS. Per esempio, se si specifica garnet.font si altezza 9, e esso non è presente, si può ottenere un altro font di altezza 9. Si dovrebbero controllare le caratteristiche del font (usando AskFont) dopo aver richiamato OpenFont o OpenDiskFont per vedere esattamente cosa ha trovato il sistema.

Si può creare una lista dei font che hanno alcune caratteristiche in comune usando una routine di nome AvailFonts. Utilizzando questa routine, si possono, per esempio, mettere in lista tutti i font che sono residenti in ROM, o i font che hanno determinate caratteristiche di spaziatura, o molte altre caratteristiche.

Per richiamare AvailFonts si scriva:

```
AvailFonts(&afh,AFSIZE,types);
```

dove afh è un puntatore ad un'area di memoria dove mettere la struttura AvailFonts-Header (intestazione delle AvailFonts) e una serie di strutture AvailFonts; AFSIZE informa il sistema sulle dimensioni dell'array dove collocare sia l'intestazione sia i vari elementi; e types è un singolo byte contenente i bit che indicano quali tipi di font si vorrebbero includere in questa lista. Se ci sono più font di quanto sia lo spazio disponibile nella lista, il sistema ne blocca il caricamento prima di andare in overflow.

Un valore di types che sia 0xff mette in lista tutti i font. Si può trovare una lista completa dei flag di font in graphics/text.h. Ecco alcuni flag che devono essere settati:

FPF_ROMFONT	lista i font localizzati in ROM.
FPF_DISKFONT	lista i font localizzati su disco.
FPF_REVPATH	lista i font progettati per essere scritti da destra a sinistra invece che da sinistra a destra (per esempio, Hebrew)
FPF_WIDEDOT	lista i font progettati per essere usati a bassa risoluzione, con visualizzazione interlacciata
FPF_TALLDOT	lista i font progettati per essere usati ad alta risoluzione, con una visualizzazione non interlacciata.

Si selezionano combinazioni delle varie caratteristiche operando un OR sui singoli flag. Una volta generata questa lista, la si può utilizzare per trovare in essa quei font che hanno le caratteristiche che si desiderano.

AvailFont richiama automaticamente AddFont per ognuno dei font che è stato messo in lista e che non si trova ancora nella lista di sistema dei font. Questo implica un effetto interessante, infatti, se si richiama una seconda volta AvailFonts, richiedendo tutti i tipi di font, si vedrà che il sistema listerà due volte i font residenti su disco nella struttura dati AvailFonts dell'utente. Poiché AvailFonts richiama AddFont la prima volta che la si richiama, il font apparirà sia nella directory FONTS: sia nella lista di sistema. Così sarà listato due volte.

Sapendo che vi sono solo due font attualmente residenti in ROM, si può non settare il flag FPF_ROMFONT, e si avrà solo una lista dei font che sono disponibili su disco quando ritorna la routine.

Dopo che AvailFont ritorna dall'esecuzione, il sistema avrà riempito l'array di intestazione dell'AvailFonts dell'utente, così si saprà quali font si possono usare e quali sono le loro caratteristiche. Ecco come la struttura AvailFontsHeader e le strutture dati ad essa associate sono poste in memoria:

```
AvailFontsHeader          {UWORD afh_NumEnties}
AvailFont                 {UWORD af_Type;
                          /* 1= mem.res, 2=disco */
                          stuct TextAttr af_Attr;
                          }     /* attributi */
AvailFonts
<prosegue>
```

L'AvailFontsHesder contiene semplicemente il numero delle voci AvailFonts che lo seguono. L'AvailFonts contiene il tipo del font listato (in memoria o su disco) seguito da una struttura dati TextFont.

Una volta che è stata richiamata AvailFonts, si può stabilire un puntatore a ognuna delle voci listate nella tabella dei font disponibili e si può passare questo puntatore alla OpenDiskFont, come mostrato fra poco.

Per allocare memoria sufficiente per i propri array per le chiamate di AvailFonts, si deve usare la seguente sequenza:

```
#define AF_NUMERENTRIES 30

struct AvailFontsHeader *aftable;
struct aft =
{
               struct AvailFonts aft_Head;
               struct AvailFonts aft_Entry[AFNUMERENTRIES];
};

int aftablesize;
```

Poi si richiami AvailFonts scrivendo:

```
AvailFonts(aft,aftablesize,0xFE);
```

dove la variabile types è settata a 0xFE per escludere dalla lista i font residenti su ROM. Gli attributi di testo per i font residenti su ROM sono espressi come segue e possono essere usati separatamente se lo si desidera

```
struct TextAttr size32_64 = {"topaz.font",9,0,0};
struct TextAttr size40,80 = {"topaz.font",8,0,0};
```

Il listato 4.16 fornisce un insieme di istruzioni che permette di prelevare e utilizzare le informazioni sui font disponibili contenute nella tabella corrispondente.

```
/* estrae informazioni dalla tabella dei Font */
UWORD howmany_entries;

struct TextAttr *myTextAttr;

howmany_entries = aft.aft_Head.afh_NumEntries;

/* punta alla prima voce nell'Array dell'AvailFont */

myTextAttrPointer = (struct TextAttr)(aft.aft_Entry);

/* usa queste voci */

int i;
struct TextFont *OpenDiskFont(), *tf;

for(i = 0; i< howmany_entries; i++ )
{
      tf = OpenDiskFont(mytextAttrPointer[i]);

      if(tf != NULL)          /* se il Font è aperto... */
      {
            SetFont(rp,tf);

            /* rp è un puntatore a una RastPort che */
            /* deve essere usata per stampare qualsiasi */
            /* testo con questo Font */
            Text(rp,"sample text",11);
            Delay(30);
            CloseFont(tf)           /* chiudiamo ciò che abbiamo aperto */
      }
}
```

Listato 4.16

Caratteristiche del testo. Si può controllare come vengono scritti i font all'interno della RastPort. Le caratteristiche che possono essere controllate sono il colore, (incluso il reverse), se è boldface, se è italics, e se è underline.

Si seleziona il colore del testo utilizzando le funzioni SetAPen, SetBPen e SetDrMd. Il corpo del testo è disegnato con il colore APen.

Se il modo di disegno è settato a JAM1, questo è il solo colore usato quando viene stampato un carattere del testo.

Il rettangolo di contenimento del testo è disegnato con la BPen se il modo grafico è settato a JAM2.Per evidenziare alcune parti di un testo viene spesso usata la tecnica dell'inversione dei colori. In altre parole, il colore di APen è usato per lo sfondo e quello di BPen per i caratteri. Non c'è bisogno di scambiare le penne per ottenere questo. Si deve soltanto settare il modo di disegno (SetDrMd) in modo tale da includere l'INVERSVID come mostrato di seguito:

```
SetDrMd(rp,JAM1+INVERSVID);      /* mostra il testo nel colore */
                                 /* dello sfondo e il rettangolo */
                                 /* di contenimento nel colore */
                                 /* della penna primaria */

SetDrMd(rp,JAM2+INVERSEVID);     /* mostra il testo nel colore */
                                 /* dello sfondo e lo sfondo */
                                 /* nel colore della penna *
                                 /* primaria */
```

Testi in bold, in italic, e in underline. Si selezionano queste caratteristiche del testo mediante l'uso della funzione SetSoftStyle. Si può chiedere al sistema di operare su un font in modo tale da mostrare i caratteri del font stesso in uno o più di questi stili particolari. Per richiamare SetSoftStyle si scriva:

```
SetSoftStyle(rp,style,mask);
```

dove rp è un puntatore a una RastPort; style è un valore che contiene i bit che identificano lo stile che si desidera; e mask è un set di bit che informa il sistema su quali bit di stile si vuole operare.

Il valore mask è utile se, per esempio, si è selezionato italic e si vuole mutarlo in underline senza interferire con l'adattamento dello stile italic. (I bit di flag sono specificati nel file Include di sistema di nome graphics/text.h). Si può settare l'Underline senza alterare gli altri bit specificando uno stesso valore per la mask e lo style, cioè:

```
SetSoftStyle(rp,FPF_ITALICS,FPF_ITALICS);
```

Se si desidera resettare tutti bit di stile senza curarsi di come sono settati, si usi un valore di 0xFF per la mask:

```
SetSoftStyle(rp,FPF_ITALICS,0xFF);

/* non si cura di come fossero settati i vari Bit */
/* pone il tutto a italic */
```

Se un font è stato progettato in partenza di tipo bold, e si chiede al sistema di operare su di esso un bold, tale operazione non verrà eseguita poiché il bit di bold è già parte integrante della struttura di base del testo (nei flag di font dove viene descritto il font stesso). La stessa cosa accade per gli altri due stili. Se si vogliono determinare le caratteristiche che possono essere legittimamente settate per il font corrente, si usi la funzione AskSoftStyle. Per richiamare tale funzione si scriva:

```
enable = AskSoftStyle(rp);
```

dove rp è un puntatore a una RastPort. Il valore restituito, enable, è un byte che contiene i flag che mostrano all'utente quali stili generati algoritmicamente possono essere scelti per queto font.

Ecco una parte di programma che lista tali stili:

```
UBYTE enable;

enable = AskSoftStyle(rp);

if(enable & FPF_UNDERLINED)
   printf("Si può richiedere lo stile Underline\n");
if(enable FPF_BOLD)
   printf("Si può richiedere lo stile bold\n");
if(enable & FPF_ITALICS)
   printf("Si può richiedere lo stile italic\n");
```

Ecco una nota finale sull'underline. Quando il sistema genera una sottolineatura per il testo, esso la pone sulla linea sottostante la linea di base del testo. Alcuni font possono essere stati progettati in modo da non avere parti che vanno al di sotto della linea di base, il sistema non può creare una sottolineatura per questi caratteri poiché essa andrebbe posta al di fuori del rettangolo di contenimento dei caratteri stessi.

Si può vedere se è possibile effettuare una sottolineatura confrontando l'altezza del testo con il valore della linea di base del testo che si trova nella RastPort. Ecco un modo per fare ciò:

```
if(rp->TxHeight-rp->TxBaseline ==1)
printf("Non è possibile sottolineare!");
```

La linea più alta del testo è la linea zero, così il valore della linea più bassa sarà uguale a TxHeight meno uno.

Cancellare e operare uno scroll sulle aree di disegno

Si può cancellare un'intera area di disegno di una RastPort usando la funzione SetRast. Questa funzione setta un'intera area di disegno in modo che sia di un solo colore. Per richiamare SetRast si scriva:

```
SetRast(rp,color);
```

dove rp è un puntatore a una RastPort, e color è il colore al quale deve essere settata la RastPort.

Si può utilizzare questa funzione sulla RastPort ottenuta da Intuition nella fase di apertura della window. Se non si specifica il flag GIMMEZROZERO tra i flag della window, usando la funzione SetRast si cancellerà la propria window per intero cioè non solo l'area di disegno interna ma anche il gadget e i bordi. Se invece si setta GIMMEZROZERO, allora solo l'area di disegno risentirà della funzione SetRast.

Le funzioni ClearEOL e ClearScreen sono disponibili nella graphics.library. Queste funzioni sono orientate ad un uso nell'ambito dei testi e sono utilizzate dalla device console per cancellare una linea e per cancellare una schermata sino in fondo. Poiché sono usate in ambiente testo, il loro modo di funzionamento dipende dal font correntemente selezionato per quella RastPort.

ClearEOL comincia dalla posizione corrente della penna di disegno e cancella un'area rettangolare larga sino al margine destro estremo della RastPort e alta quanto il valore della variabile TxHeight della struttura RastPort e posizionata come se il testo fosse stato disegnato. ClearScreen opera un ClearEOL, e poi cancella tutta l'area della RastPort che si trova al di sotto della linea cancellata.

Per richiamare tali funzioni si scriva:

```
ClearEOL(rp);
ClearScreen(rp);
```

dove rp è un puntatore alla RastPort che deve essere cancellata.

Quando si usano i testi per applicazioni come un programma di terminale o un Word Processor, si devono poter scrollare in alto delle linee per far spazio per una nuova linea alla fine oppure scrollare in basso alcune linee per far spazio a una linea all'inizio dello schermo. Una azione di scroll che risulti fluida è certamente preferibile al dover ridisegnare tutte le linee dello schermo. La funzione ScrollRaster può essere usata per ottenere ciò. Per richiamare ScrollRaster si scriva:

```
ScrollRaster(rp,dx,dy,xmin,ymin,xmax,ymax);
```

dove rp è un puntatore a una RastPort; xmin, ymin, xmax, e ymax sono le coordinate degli angoli superiore sinistro e inferiore destro di un rettangolo che racchiude la zona della RastPort che si desidera scrollare; e dx e dy sono il numero di pixel (orizzontali e verticali) dei quali deve essere mossa questa zona delimitata in riferimento alle coordinate 0,0 dell'area in questione.

Per esempio, per scrollare in su un set di linee per fare spazio ad una nuova linea in fondo si userà un valore di dx uguale a 0 (nessuno spostamento orizzontale) e di dy uguale a 8 (8 pixel in su rispetto a 0,0).

ScrollRaster risulta comoda se non si sta usando una window GIMMEZEROZERO nella quale si può limitare l'area da scrollare all'interno dei bordi della window, lasciando gli stessi bordi indisturbati.

Combinare oggetti per formare delle immagini

In questo paragrafo si vedrà come creare la propria area di disegno fuori dello schermo. Fin qui, i programmi di questo capitolo hanno utilizzato una RastPort ottenuta dall'Intuition mediante una chiamata di OpenWindow. Ora, si vedranno le funzioni che servono a creare un'area di disegno e la sua RastPort direttamente. Si disegneranno alcuni oggetti in una RastPort off-screen, e poi si copierà l'oggetto composito all'interno di una window, nella quale sarà visibile.

Per comprendere come fare queste cose, si deve sapere:

- Come inizializzare una Bitmap e allocare la memoria per i suoi piani di bit
- Come inizializzare una RastPort
- Come copiare i dati da una Bitmap ad un'altra.

Si esamineranno tutti questi argomenti, uno alla volta.

Inizializzare una Bitmap

La struttura dati Bitmap contiene le variabili che definiscono le dimensioni di un'area di disegno e la locazione della memoria che è dedicata per il suo uso. Si utilizza la funzione InitBitMap per inizializzare la struttura dati stessa, e poi, a seconda del numero di piani che la Bitmap contiene, si deve allocare una quantità di memoria appropriata all'immagazzinamento dei bit che rappresentano l'area di disegno.

Per esempio, si immagini di voler avere un'area di disegno off-screen di 320 per 200, che possa contenere quattro colori. Questo richiede una profondità di 2, cioè due piani di bit dedicati a questa Bitmap. Ecco un esempio di inizializzazione:

```
#define DEPTH 2
#define WIDTH 320
#define HEIGHT 200

struct BitMap myBitM;
int i;
estern PLANEPTR AllocRaster();
```

```
initBitMap(myBitM,DEPTH,WIDTH,HEIGHT);

for(i=0; i<DEPTH; i++)
{
   myBit.Planes[i] = AllocRaster(WIDTH,HEIGHT);
   if(myBitM.Planes[i] == 0)
   {
               /* elaborazione dell'errore...manca memoria */
   }
}
```

E questo è tutto quello che serve. Più tardi, terminato l'uso della Bitmap off-screen, si dovrà liberare la memoria che è stata allocata, di solito con la seguente sequenza:

```
for(i=0; i<DEPTH; i++)
{
   if(myBitM.Planes[i] != 0)
   {
         FreeRaster(myBitM.Planes[i],WIDTH,HEIGHT);
   }
}
```

Inizializzare una RastPort

Ora che è stato inizializzato lo spazio per i piani di bit, risulta facile inizializzare la RastPort:

```
struct RastPort myRast;

InitRastPort(myRast);

myRast.BitMap = &myRast
   /* linka la Bitmap dentro questa RastPort */
```

Ora, usando un puntatore a questa RastPport, si possono settare i numeri di colore di penna e il modo di disegno, muovere le penne, e fare tutto ciò che si desidera con la grafica. Le due linee seguenti definiscono un puntatore a una RastPort off-screen e stabiliscono il valore di tale puntatore.

```
struct RastPort *offscreeenrp;   /* un puntatore a questa RastPort */

offscreenrp = &MyRast;
```

In questa RastPort off-screen si possono disegnare figure complesse che non si desidera siano viste sino a quando non sono state completate. Poi si può copiare questa figura dall'area off-screen in una RastPort che sia visibile.

Copiare i dati da una Bitmap ad un'altra

Amiga fornisce tre differenti routine per copiare dei dati da una Bitmap ad un'altra: BltBitMap; ClipBlit, e BltBitMapRastPort.

BltBitMap copia i dati da una Bitmap ad un'altra, oltrepassando completamente l'uso della RastPort. Questa routine è potenzialmente la più pericolosa e può far si che il sistema si blocchi perché non opera alcun check per controllare se tutti i bit sono stati inseriti nell'area di destinazione. I movimenti dei dati (chiamati Blit) al di fuori dei confini della Bitmap inevitabilmente distruggono dei dati che magari non dovevano essere alterati e così il sistema potrebbe essere danneggiato.

ClipBlit copia i dati da una RastPort ad un'altra. Essa ritaglia un rettangolo di dati ad una particolare posizione X,Y nella RastPort sorgente e lo piazza in una posizione X,Y selezionata nella RastPort di destinazione. Lo svantaggio di ClipBlit è che, se si hanno due oggetti che devono ricoprirsi nell'area di destinazione, saranno i rettangoli in toto a ricoprirsi e non solo gli oggetti. Se un oggetto è circondato da uno spazio vuoto, per esempio, la parte vuota di questo oggetto cancellerà una porzione del primo oggetto che è stato piazzato nell'area di destinazione.

BltBitMapRastPort è leggermente più veloce di ClipBlit poiché la copia avviene dalla Bitmap sorgente alla RastPort di destinazione. Questo significa che non vi sono operazioni di distinzione tra Layer sull'area sorgente, e si assume che tale area non sia composta da Layer sovrapposti.

Per richiamare ClipBlit si scriva:

```
ClipBlit(src_rp,src_x,src_y,dest_rp,dest_x,dest_y,
                            size_x,size_y,minterm);
```

I parametri di ClipBlit sono i seguenti:

- src_rp e dest_rp sono dei puntatori alle RastPort sorgente e di destinazione
- src_x e src_y sono le coordinate X,Y dell'angolo superiore sinistro del rettangolo contenente l'area sorgente.
- dest_x e dest_y sono le coordinate X,Y nell'area di destinazione alle quali deve essere piazzato il rettangolo di dati copiato
- size_x e size_y sono le dimensioni orizzontale e verticale del rettangolo da copiare
- minterm è il valore che indica esattamente come devono essere trattati i dati durante la fase di copiatura

Un valore di minterm di 0xC0 opera una copia diretta dalla sorgente alla destinazione. Un valore di 0x30 inverte la sorgente - dovunque vi era un bit-0 viene posto un bit-1 e viceversa. Un valore di 0x50 ignora totalmente la sorgente e semplicemente inverte l'area di destinazione.Per richiamare BltBitMapRastPort si scriva:

```
BltBitMApRastPort(src_bm,src_x,src_y,dest_rp,dest_x,dest-y,
                          size_x,size_y,minterm);
```

dove tutti i parametri sono gli stessi di quelli della routine ClipBlit, a eccezione di src_bm che rappresenta la Bitmap sorgente dalla quale devono essere copiati i dati.

Utilizzare le routine di spostamento dati

Il listato 4.17 apre una window Simple-Refresh e una RastPort off-screen. Nella prima window vengono disegnate delle rette colorate. Nella RastPort off-screen, viene disegnata una serie di rettangoli e viene spostata nell'area visibile mediante la ClipBlit prima e BltBitMapRastPort dopo.

Nel prossimo capitolo, si vedrà molto di più circa l'Intuition, compreso come interpretare la posizione del mouse e la pressione dei suoi tasti. Per questo esempio è stato fornito un semplice controllo temporizzato.

Copiare con trasparenza

Si sarà notato che con ClipBlit e BltBitMapRastPort viene riportato l'intero rettangolo contenente gli oggetti da copiare. Per evitare questo, si potrebbe sviluppare un oggetto off-screen con un colore di sfondo 'trasparente'.

```
/* offscreen.c */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "mydefines.h"
#include "window1.h"
#include "graphics/gfxmacros.h"

#define DEPTH 2
#define WIDTH 640
#define HEIGHT 200

int intuitionBase, GfxBase;

int i,j;
extern PLANEPTR AllocRaster();

struct BitMap myBitM;         /* la Bitmap dell'area off-screen */
struct RastPort myRast;       /* la RastPort dell'area off-screen */
struct RastPort *offscreenrp; /* un puntatore a questa RastPort */

struct RastPort *rport;
            /* il puntatore alla RastPort visibile */
struct Window *w;  '
            /* il puntatore alla window visibile */
extern struct Window *OpenWindow();

main()
{
      GfxBase = OpenLibrary("graphics.library",0);
      if(gfxBase == 0)
      {
            printf("graphics.library non si apre\n");
            exit(10);
      }
      IntuitionBase = OpenLibrary("intuition.library",0);
      if(IntuitionBase == 0)
      {
            printf("intuition.library non si apre\n");
            exit(15);
      }

      w = OpenWindow(&myWindow);
```

```
if(w == 0)
{

      printf(" La window non si è aperta\n");
      CloseLibrary(GfxBase);
      exit(20);
}
rport = w->RPort;

InitBitMap(myBitM,DEPTH,WIDTH,HEIGHT);

for(i=0; i<DEPTH; i++);
{
            myBitM.Planes[i]=AllocRaster(WIDTH,HEIGHT);
            if(myBitM.Planes[i] == 0)
            {
            /* elaborazione errore.... non basta la memoria */
            }
}
InitRastPort(nyRast);

myRast.BitMap = &myBitM;
/* linka la Bitmap dentro questa RastPort */
offscreenrp = &myRast;

/* disegna alcune rette nell'area visibile così si */
/* potrà vedere cosa succede quando i dati off-screen */
/* saranno spostati */

SetAPen(rport,3);
SetDrMd(rport,JAM1);

j = 10;
for(i=0; i<30; i++)
{
            Move(rport,j,0);
            Draw(rport,j,100);
            j += 10;
}

/* disegna qualcosa nell'area off-screen */

SetRast(offscreenrp,0); /* prima la cancella */

SetAPen(offscreenrp,1);
SetDrMd(offscreenrp,JAM1);
RectFill(offscreenrp,30,30,50,50);

SetAPen(offscreenrp,2);
RectFill(offscreenrp,40,40,60,60);

SetAPen(offscreenrp,3);
RectFill(offscreenrp,50,50,70,70);

/* poi copia tutto dentro l'area visibile in due modi */
```

```
        /* differenti... una volta con ClipBlit (l'intero */
        /* rettangolo) e una volta con BltBitMapRastPort */
        /* (solo le porzioni colorate) scelte mediante minterm */

        ClipBlit(offscreenrp,30,30,rport,10,30,40,40,0xc0);

        Delay(150);

        BltBitMapRastPort(myBitM,30,30,rport,90,30,40,40,0xc0);

        Delay(300);

cleanup:
        if(w)
                CloseWindow(w);
        if(IntuitionBase)
                CloseLibrary(IntuitionBase);
        if(GfxBase)
                CloseLibrary(GfxBase);

        for(i=0; i<DEPTH; i++)
        {
                        if(myBitM.Planes[i] != 0)
                        {
                                FreeRaster(myBit.Planes[i],WIDTH,HEIGHT);
                        }
        }
}               /* fine del main */
```

Listato 4.17

Il listato 4.18 implementa una ricopiatura con trasparenza. Questo listato usa una nuova funzione chiamata CliBlitTransparent e tre nuove funzioni ad essa associate che servono per allocare e cancellare ciò che serve per essa.

ClipBlitTransparent trasforma la routine ClipBlit in un generatore di Bobs (Blitter objects). Ogni punto colorato con il colore 0 nella sorgente diviene trasparente. Per richiamare ClipBlitTansparente si scriva:

```
        ClipBlitTransparent(src_rp,src_x,src_y,dest_rp,dest_x,
                        dest_y,size_x,size_y,shadow_rp,makeShadow);
```

La RastPort di maschera denominata Shadow punta ad una Bitmap contenente un solo piano di bit della stessa dimensione dell'oggetto che deve essere spostato. I bit di questo piano vengono qui posizionati mediante la routine CreateShadowRP. Per ogni bit contenete un colore diverso dal colore 0 nell'oggetto sorgente, vi sarà un bit-1 nella corrispondente posizione della maschera Sha-

dow. Durante l'operazione di spostamento, la maschera è utilizzata per creare un buco nell'area di destinazione. Poi l'oggetto sorgente viene posto in tale buco. I colori dell'oggetto appaiono nell'area dell'oggetto, e i colori dello sfondo appaiono dove non sono usati i colori dell'oggetto.

Si può utilizzare CreateShadowBM e CreateShadowRP per allocare e preparare le struttura dati per la routine ClipBlitTransparent. Esse, però, non creano la Shadow direttamente. Per fare questo, si deve settare il valore di makeShadow a TRUE. Se questa routine è stata eseguita già una volta, per una specifica Bitmap Shadow, makeShadow può essere settata a FALSE. ClipBlitTransparent restituisce un valore nullo se ha funzionato correttamente, un valore non nullo se non ha funzionato.

La seguente parte di programma mostra come si utilizza ClipBlitTransparent. Il listato 4.18 contiene l'esempio completo.

```
struct BitMap *sbm;
struct RastPort *srp;
srp = NULL;
sbm = CreateShadowBM(depth,width,height);
srp = CreateShadowRP(sbm,srp);
/* si deve controllare che il valori di ritorno di sbm e */
/* srp siano nulli */
ClipBlitTransparent(sourceRP,sourceX,sourceY,destRP,
   destX,destY,sizeX,sizeY,srp,TRUE);
DeleteShadowRP(srp);
DeleteShadowBM(sbm);
```

Una funzione che crea una Bitmap utilizzabile da ClipBlitTransparent è chiamata CreateShadowBM.

```
/* cliptransparent.c */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "exec/memory.h"

#define WINDOWFLAGS {WINDOWSIZING|WINDOWDRAG|WINDOWDEPTH|WINDOWCLOSE}

struct NewWindow nw =
{
      100,80,           /* posizione iniziale */
      340,110,          /* larghezza, altezza */
```

```
        -1,-1,              /* detail pen, block pen */
        0,
                            /* Flag IDCMP */
        WINDOWFLAGS
                            /* Flag di window */
        NULL,         /* puntatore al primo Gadget */
        NULL,         /* puntatore al Checkmark */
        "ClipBlitTransparent",
                            /* nome della window */
        NULL,         /* puntatore allo schermo */
        NULL,               /* puntatore alla SuperBitmap */
        60,60,640,200       /* dimensioni min/max */
        WBENCHSCREEN        /* tipo di schermo nel quale aprirla */
};

        struct GfxBase *GfxBase;
        struct IntuitionBase *IntuitionBase;

        extern struct RastPort *CreateShadowRP();
        extern struct BitMap *CreateShadowBM();
        extern struct Window *OpenWindow();

main()
{
        short i;
        int x2, y2, error;

        struct RastPort testRP;
        struct BitMap saveBM;

        struct RastPort *rp;
        struct Window *w;

        struct RastPort *imageShadowRP;
        struct BitMap *imageShadowBM;

        SHORT x,y;


        GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
        if(gfxBase == NULL)
        {
                printf("Non riesco ad aprire graphics.library\n");
                exit(1000);
        }

        IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("Intuition.library",0);
        if(IntuitionBase == NULL)
        {
                CloseLibrary(GfxBase);
                printf("Non riesco ad aprire Inuition.library\n");
                exit(1000);
        }
```

```
w = OpenWindow(nw);           /* apre una window */
if(w == NULL)
      goto cleanup;
rp = w_.RPort;

InitBitMap(testBM,2,00,150);
InitBitMap(saveBM,2,00,150);

error = FALSE;
for(i=0; i<2; i++)
{
      testBM.Planes[i] = (PLANEPTR)AllocRaster(300,150);
      if(testBM.Planes[i} == 0)
                  error = TRUE;
}
if(error)
      goto cleanup;
for(i=0; i<2; i++)
{
      saveBM.Planes[i] = (PLANEPTR)AllocRaster(300,50);
      if(saveBM.Planes[i] == 0)
                  error = TRUE;
}
if(error)
      gotocleanup;
InitRastPort(&testRP);
testRP.BitMap = &testBM;

InitRastPort(saveRP);
saveRP.BitMap = saveBM;
SetAPen(rp,1);
Move(rp,0,0);
Draw(rp,200,100);       /* nella window principale */
SetAPen(rp,2);
Move(rp,6,0);
Draw(rp,206,100);       /* nella window principale */
SetAPen(rp,1);
Move(rp,12,0);
Draw(rp,212,100);       /* nella window principale */
SetAPen(&testRP,2);
RectFill(testRP,0,0,130,55);
SetAPen(testRP,0);
                  /* crea un buco nel centro del rettangolo */
RectFill(&testRP,10,10,120,45);

x = 20;
y = 10;
x2 = 2;
y2 = 1;

imageShadowBM = CreateShadowBM(2,10,55);
if(imageShadowBM == 0)
            goto cleanup;
imageShadowRP = CeateShadowRP(imageShadowBM,NULL);
```

```
        if(imageShadowRP == 0)
        goto cleanup;

        /* INIZIALIZZAZIONE */

        ClipBlit(rp,x,xy,0,0 130,55, 0xc0);/* salva */

        /* questo (TRUE) crea la maschera Shadow per la prima volta, */
        /* in modo che le volte seguenti la possiamo direttamente */
        /* usare */
        ClipBlitTransparent(&testRP,0,0,
                                    rp,x,y,
                                    130,5,
                                    imageShadowRP,TRUE);
        ClipBlit(&saveRP,0,0, rp,x,y, 10,5, 0xc0);/* restore */


        for(i=0; i<60; i++)
        {
                    ClipBlit(rp,x,y,&saveRP,0,0,130,55,0xc0);
                          clipBlitTransparent(&testRP,0,0,
                                    rp,x,y,
                                    130,5,
                                    imageaShadowRP,FALSE);
                                    /* Usa Shadowmask */
                    Delay(5);
                    ClipBlit(&saveRP,0,0,rp,x,y,130,55,0xc0); /* restore */
                    x += x2;
                    y += y2;
                    if(y < 10 | y > 45)
                    {
                              y2 = -y2;
                              x2 = -x2;
                    }
        }
cleanup:
        for(i=0; i<2; i++);
        {
                    if(testBM.Planes[i])
                          FreeRaster(testBM.Planes[i],300,150);
                    if(saveBM.Planes[i])
                          FreeRaster(saveBM.Planes[i],300,150);
        }
        if(w) CloseWindow(w);
        DeleteShadowRP(imageShadowRP);
        DeleteShadowBM(imageShadowBM);

        if(IntuitionBase)
              CloseLibrary(IntuitionBase);
        ifGfxBase)
              CloseLibrary(GfxBase);


}           /* fine del main */
```

```
ClipBlitTransparent(srp,sx,sy,drp,dx,dy,width,height,
                        shadowRP,makeShadow)
struct RastPort *srp, *drp, *shadowRP;
SHORT sx,sy,dx,dy;
SHORT width,height,makeShadow;
{

      /* FORMA UNA MASCHERA SHADOW OPERANDO UN OR TRA UNA MASCHERA */
      /* SINGOLA E UNA MULTIPLANARE */

if(makeShadow)
      /* la maschera deve essere fatta per la prima volta */

            ClipBlit(srp,sx,sy,shadowRP,0,0,width,height,0xe0);

            /* il valore esadecimale 0 significa B and C + C not B. */
            /* cioè B + C (B or C), cioè significa */
            /* "poni un 1 ovunque ci sia un 1 nella sorgente o */
            /* nella destinazione". Permette che si possa operare */
            /* un Merge su tutti i piani */

      /* USA LA MASCHERA SHADOW PER CREARE */
      /* UN BUCO NELL'AREA DI DESTINAZIONE */

      ClipBlit(shadowRP,0,0,drp,dx,dy,width,height,0x30);

      /* RIEMPIE IL BUCO CON MATERIALE DELLA SORGENTE */

      ClipBlit(srp,sx,sy,drp,dx,dy,width,height,0xe0);
      return(0);
}

struct BitMap

*CreateShadowBM(depth,width,height)
SHORT depth, width, heigth;
{

      /* la maschera Shadow ha un solo piano di Bit, grande solo */
      /* quanto l'area rettangolare che vogliamo spostare. Essa */
      /* preleva un puntatore per un piano. Ma sembra che possieda */
      /* 5 piani di Bit */

      SHORT i;
      struct BitMap *shadowBM;
      if((shadowBM = (struct BitMap *)
            AllocMem(sizeof(struct BitMap),MEMF_CHIP)) = NULL);
            return(NULL);

      InitBitMap(shadowBM,depth, width,height);

      if(shadowBM->Planes[0] = (PLANEPTR)
            AllocMem(RASSIZE(WIDTH<HEIGHT),
                  MEMF_CHIP | MEMF_CLEAR)) == NULL)
      {
```

```
            FreeMem(shadowBM,sizeof(struct BitMap)); return (NULL);
      }

      for(i=1; i<depth; i++)
      {
            shadowBM->Planes[i] = shadowBM->Planes[0];
      }
      return(shadowBM);
}     /* fine di CreateShadowBM */

DeleteShadowBM(sbm)
struct BitMap *sbm;
{

      if(sbm != NULL)
      {

            if(sbm->Planes[0] != NULL)
                  FreeMem(sbm->Planes[0],
                        RASSIZE(( *(sbm->BytesPerRow),sbm->Rows));

                              FreeMem(sbm,sizeof(struct BitMap));
      }
      return
}
struct RastPort
*CreateShadowRP(shadowBM, oldshadowRP)
struct BitMap *shadowBM;
struct RastPort *oldahadowRP;
{

      struct RastPort *shadowRP;
      /* se shadow non è nulla, stiamo semplicemente linkando */
      /* una nuova BitMap all'interno di una struttura dati già */
      /* esistente */
      if(oldshadowRP == NULL)
      {
      if((shadowRP = (struct RastPort *)
            AllocMem(sizeof(struct RastPort), MEMF_CHIP))
                  == NULL)
            return(null);
      InitRastPort(shadowRP);
      }
      else
      {
            shadowRP = oldshadowRP;
                  /* utilizza il vecchio valore, se ne esiste uno */
      }
      /* linka la Bitmap e la RastPort */

      shadowRP->BitMap = shadowBM;
      return(shadowRP);
}

DeleteShadowRP(srp)
```

```
struct RastPort *srp;
{

      if(srp != NULL)
            FreeMem(srp,sizeof(struct RastPort));
            return(0);
}
```

Listato 4.18

Questa funzione alloca lo spazio per una Bitmap e per una maschera che contenga un solo piano di bit di una data larghezza e di una data altezza (tutti i piani sono settati in modo tale che ovunque vi sia un bit-1 in un piano c'è un bit-1 nella maschera). Lo spazio viene utilizzato per maneggiare ogni sottofigura monoplanare costituente l'oggetto, per permettere che possa essere disegnato con trasparenza all'interno di un'area di destinazione mediante ClipBlitTransparent.

Gli Input necessari per CreateShadowBM sono la profondità (il numero di piani dell'area dei destinazione), la larghezza (la larghezza massima dell'oggetto espressa in pixel), e l'altezza dell'oggetto. Essa restituisce un puntatore alla struttura BitMap se tutto ha funzionato. La struttura contiene un puntatore a un piano di bit di una parte dinamicamente allocata di memoria, che può essere usato con CreateShadowRP per creare l'attuale sottofigura dell'oggetto in questo piano di bit. La funzione ritorna un valore nullo se non vi era abbastanza memoria disponibile per effettuare tutte le operazioni.

ClipBlitTransparent ha bisogno di un'area nella quale immagazzinare la maschera dell'oggetto. La funzione CreateShadowRP fornisce ciò. CreateShadowRP crea una struttura dati di disegno che può essere usata con oggetti mono o pluriplanari. Per richiamare CreateShadowRP si scriva:

```
shadowRastPort = CreateShadowRP(shadowBitMap,oldShadowRP);
```

dove shadowBitMap è il puntatore ritornato da CreateShadowBM, e oldshadowRP è il puntatore ritornato da una precedente chiamata di CreateShadowRP. La prima chiamata deve specificare un valore NULL per questo parametro per chiedere al sistema di allocare dinamicamente una struttura dati RastPort. Il parametro viene fornito in luogo di una routine separata che semplicemente linka le nuove BitMap all'interno di una RastPort esistente.

CreateShadowRP ritorna un puntatore a una RastPort che può essere usato per disegnare nella sottofigura oppure fornisce un valore nullo se non c'è abbastanza memoria.

Questo capitolo ha trattato la maggior parte delle funzioni della graphics.library di Amiga. Le funzioni di visualizzazione grafica qui trattate sono tutte compatibili con Intuition.

Nel prossimo capitolo, si vedrà come Intuition costruisce le proprie immagini e come essa interagisca con il programmatore. Si forniranno durante la spiegazione dei tool di base che permetteranno a chiunque di costruire le proprie applicazioni in Intuition.

Capitolo 5

Intuition

In questo capitolo si vedrà molto di più riguardo al modo di interagire con Intuition rispetto a quanto è stato detto sinora nel libro. Gli screen e le window Intuition che sono stati aperti nel capitolo precedente saranno qui spiegati approfonditamente.

Molte delle specifiche di Intuition su come disegnare e rappresentare le cose richiedono che si forniscano i valori delle posizioni relative ad altri oggetti. Per esempio, una window viene posizionata in relazione allo schermo nel quale è contenuta; un requester è posizionato in relazione alla window alla quale è allegato; i testi e i gadget sono posizionati in relazione al requester al quale sono allegati; le voci dei menu sono posizionate in relazione al menu a cui appartengono; e le sottovoci sono posizionate rispetto alle voci.

Questa relatività fornisce allIntuition una grande flessibilità e risparmia al programmatore gran parte del lavoro. Per esempio, se non si è soddisfatti della posizione alla quale si è posto un requester, lo si sposta dove si vuole, e tutti i testi e i gadget ad esso allegati si sposteranno con lui. Una volta che si ha chiara l'idea di questa relatività sarà molto facile creare window, requester, gadget, e così via.

Comunicare con Intuition

Il modo più diretto per comunicare con Intuition è usare l'IDCMP (Intuition Direct Communications Message Port). Si dice a Intuition a quale tipo di evento si è interessati, e ogniqualvolta questo evento si verifica, Intuition invierà un messaggio per riferire su tale evento. I messaggi che si ottengono dalla IDCMP sono chiamati IntuitionMessage.

Si possono ottenere messaggi che riguardano:

Window — Si può essere informati quando un utente attiva, disattiva, ridimensiona e chiude una finestra. Così, si può sapere se una determinata finestra necessita di un refresh (deve essere ridisegnata) quando il sistema ne oscura o ne mette a nudo una parte. Si può utilizzare SIZEVERIFY per assicurarsi che qualcosa di speciale sia fatto prima che il sistema permetta l'attuazione di

un'operazione di ridimensionamento. (Nonostante la richiesta di ridimensionamento dell'utente, si può impedire che questo avvenga nel caso che non ci sia abbastanza memoria per operare un particolare movimento di dati sul quale si basa il ridimensionamento stesso).

Dischi — Il Software in uso potrebbe voler sapere se qualcuno ha tolto un disco che si doveva usare o sul quale c'è un file già aperto. L'AmigaDos richiederà poi che quel disco venga reinserito se un file è realmente aperto. Ma è utile che un programma sappia il problema al quale può andare incontro.

Menu — Se un utente seleziona una voce di un menu, il messaggio contiene il numero del menu e, il numero della voce scelta, e della sottovoce, se esiste.

Gadget — Nel progettare un gadget, si definisce un rettangolo o all'interno di una window all'interno di un requester. Se l'utente schiaccia o rilascia il pulsante di selezione del mouse all'interno del rettangolo che è definito per uno dei gadget, il programma riceverà un messaggio che lo informa su quale gadget è stato premuto o rilasciato.

Tastiera — Quando la window in azione è quella aperta dal programmatore, ogni Input dalla tastiera viene inviato a tale window. Vi sono due tipi di input da tastiera richiedibili: le sequenze di pressione e rilascio dei tasti, dove la traduzione dei tasti premuti viene effettuata dal programma in uso, e VANILLAKEY, dove viene inviata come messaggio solo una traduzione dei tasti premuti (di solito in codice ASCII). (Si veda per questo il sesto capitolo).

Timer — L'Intuition è in grado di generare eventi temporizzati ognuno da 1/30 di secondo. Nonostante il device timer (discussa nel sesto capitolo) sia accessibile indipendentemente, è conveniente che il un task risponda a questo evento, comunemente accessibile, piuttosto che prendersi il disturbo di settare una comunicazione separata con il device timer.

Requester	Si può ottenere un messaggio una volta che il Requester è posizionato (REQSET) e una volta che il Requester stesso è stato cancellato dall'area di disegno (REQCLEAR). Se si setta REQVERIFY, si può essere informati che un requester sta per essere disegnato e si può fare qualcosa, come salvare l'area di sfondo della window Simple-Refresh, prima di rispondere. Quando si risponde al messaggio, si da al sistema il permesso di disegnare il requester.
Mouse	L'Intuition tiene traccia della posizione del mouse all'interno della struttura IntuitionBase. Essa inoltre contiene la posizione del mouse relativamente a tutte le finestre del sistema e riporta anche la posizione del mouse in relazione a determinati eventi di Intuition. In particolare, si può settare il Flag FOLLOWMOUSE nella struttura gadget e mentre il gadget è selezionato, il proprio IntuiMessage sarà informato sui movimenti del mouse. Oppure, si può settare il flag REPORTMOUSE per una window e ricevere informazioni sui movimenti del mouse. O, ancora, si può si può settare DELTAMOVE, che converte i movimenti del mouse dalle coordinate assolute alle coordinate relative all'ultimo spostamento riferito.

Si troveranno implementate in questo capitolo la maggior parte di queste caratteristiche.

Messaggi da Intuition

Ogni volta che si riceve un messaggio da Intuition, si cerchi di rispondere al più presto. Ogni volta che Intuition rileva che sta avvenendo uno degli eventi di uno dei tipi richiesti, essa guarda se c'è un messaggio che può riutilizzare, riempiendo la struttura IntuiMessage con le informazioni che le sono state richieste. Se non ci sono messaggi da riutilizzare, Intuition alloca un nuovo blocco di messaggio vuoto per questo messaggio e lo invia qui.

Se il task del programmatore aspetta troppo a rispondere al messaggio di Intuition, Intuition continua a mangiarsi memoria finché, magari, la memoria disponibile finisce. Se il task del programmatore è lento a rispondere, una volta che

Intuition si è riservata un'area di memoria per un blocco di messaggio, l'area occupata non sarà liberata finché il sistema non riceverà un ReBoot. Una volta che alloca la memoria per un messaggio, Intuition lascia tale memoria assegnata al messaggio nel caso debba essere usata nuovamente per lo stesso compito. Così, è meglio essere sicuri di rispondere molto velocemente e, possibilmente, non farsi sorpassare da Intuition.

Un metodo per rispondere velocemente consiste nel copiare l'IntuiMessage dall'area di messaggio di Intuition all'interno della propria struttura dati IntuiMessage locale e rispondervi immediatamente, prima di effettuare qualsiasi elaborazione del messaggio stesso. Questa tecnica è utilizzata più avanti in questo capitolo.

I contenuti di un IntuiMessage

Vi sono molti campi di dati in una struttura IntuiMessage. Il più importante è il campo Class, che contiene esattamente la definizione del tipo di messaggio che si sta tentando di interpretare.

Ecco una lista dei contenuti di una tipica struttura IntuiMessage così come è definita nel file include Intuition/Intuition.h:

```
struct IntuiMessage
{
   ULONG Class;
   USHORT Code;
   USHORT Qualifier;
   APTR IAdress;
   SHORT MouseX,mouseY;
   ULONG Seconds,Micros;
   struct Window *IDCMPWindow;
   struct IntuiMessage *SpecialLink;
         /* SpecialLink è riservato al sistema */
};
```

Il campo Class. Il campo Class identifica il tipo di messaggio contenuto in questo IntuiMessage. I valori accettabili per Class sono gli stessi a quelli dei Flag individuali che si possono settare per i messaggi IDCMP che si vogliono riceve-

re. Quando si forniscono i valori dei parametri di Class, si possono specificare gli stessi nomi usati per settare i Flag IDCMP.

Il campo Code. Il campo Code è usato per maneggiare le voci dei menu. Se il campo Class contiene MENUPICK, il campo code riferisce al programmatore su quale voce del menu è stata selezionata. Se il campo class contiene MENUVERIFY, il campo code riferisce quale risposta Intuition si aspetta di ricevere.

Il campo Qualifier. Il campo Qualifier è semplicemente una copia del campo Qualifier dell'evento corrente di input. (Gli IntuiMessagge in generale sono copie degli eventi di sistema con alcune traduzioni di Intuition o con delle variabili di sistema di Intuition inserite). Il campo Qualifier sarà ricevuto dal device GamePort per un evento di tipo mouse, dal device Keybord per eventi che riguardano la tastiera, o dal device Timer per eventi temporizzati. I possibili valori del campo Qualifier sono elencati nel file Include devices/inputevent.h.

Tra i possibili modificatori vi è la pressione del tasto Shift, del tasto Alt di sinistra, e del tasto Ctrl, che permettono di definire in precedenza gli eventi di Input che si sta cercando di elaborare. Per esempio, in un programma di WordProcessing, la combinazione dei tasti Shift—<freccia destra> potrebbe essere un segnale per muovere il cursore all'estremità destra dello schermo, mentre la semplice <freccia> potrebbe spostare a destra di un carattere il cursore. Il campo Qualifier permette di giudicare cose come queste, senza dover tenere traccia dello status dei singoli eventi dei modificatori.

Il campo IAdress. Il campo IAdress si riferisce ad eventi in relazione ai gadget; controlla l'indirizzo della struttura dati gadget che definisce il gadget selezionato. Utilizzando questo indirizzo, si può accedere ad ognuna delle variabili interne, come il campo identificatore di gadget (GadgetID), e si possono eseguire operazioni basate su questo valore.

I valori Mouse X e Mouse Y. Questi valori, per tutti gli eventi, contengono le coordinate correnti del mouse in relazione all 'angolo superiore sinistro dello schermo all'interno del quale il mouse è correntemente localizzato. L'Intuition contiene la posizione del mouse all'interno di una struttura IntuitionBase con una risoluzione possibile di 640 posizioni orizzontali per 400 (512 per la versione PAL) verticali. A causa della risoluzione del cursore non è possibile di fatto posizionare il cursore stesso in più di 320 diverse posizioni orizzontali. Il conto inter-

no è mantenuto in una risoluzione doppia rispetto a quella in uso, così da permettere future espansioni e per disegnare in alta risoluzione.

Se la classe di evento è DELTAMOVE, allora MouseX e MouseY specificano la lunghezza del movimento e la direzione dello stesso. Valori di movimento positivi indicano un movimento a destra, o un movimento verso il basso; valori negativi indicano un movimento verso sinistra o verso l'alto. I movimenti DELTAMOVE non si fermano neanche se il puntatore del mouse è a ridosso dell'estremità dello schermo tentando di andare oltre i suoi limiti. Il puntatore del mouse, e il contatore di posizione interno di Intuition, non procederanno oltre i limiti dello schermo anche se i valori DELTAMOVE continuano a incrementarsi o decrementarsi.

Se la classe dell'evento è MOUSEMOVE, allora MouseX e MouseY indicano la posizione attuale del mouse. Si noti che i valori del mouse sono validi a prescindere dal tipo di Input che viene riportato. Questo continua ad essere vero persino se l'evento è un MENUPICK o un GADGETDOWN o un INTUITICKS. Questo significa che l'evento che si riceve è legato direttamente all'esatta posizione del mouse al momento dell'accadimento dell'evento stesso.

Per esempio, mettiamo che si progetti un gadget che deve essere selezionato dall'utente con il mouse. Non solo si può ottenere il rilevamento di GADGETDOWN da Intuition, ma utilizzando i valori di MouseX e MouseY per quell'evento, si può sapere se l'utente ha clickato in cima, in basso, in mezzo, o da qualsiasi altra parte del gadget. Si deve semplicemente controllare la posizione del mouse rispetto a valori di LeftEdge, TopEdge, Width, e Height della struttura gadget per vedere dove l'utente ha premuto il tasto.

Il listato 5.1 è un interprete di questa situazione. Si da per scontato che il messaggio sia già stato ricevuto da Intuition e passato a questa routine per l'interpretazione.

Come alternativa possibile alla creazione di una schermata con moltissimi gadget presenti, si può, se necessario (magari se c'è poca memoria), fare un unico gadget (o nessuno) e interpretare gli IntuiMessage MouseX e MouseY come mostrato nel listato.

I valori Seconds e Micros. Ogni struttura IntuiMessage ha i propri campi Seconds e Micros, che contengono un solo valore di tempo per il messaggio. Non possono esservi due valori di tempo, perciò questo campo determina un parametro per identificare in modo univoco un IntuiMessage.

Il campo IDCMPWindow. Questa variabile contiene un puntatore alla window responsabile della generazione di questo IntuiMesagge. In particolare, è possibile a volte che IntuiMessage provenienti da window diverse pervengano alla stessa MessagePort, magari comune a molte finestre. Il campo IDCMPWindow permette a Intuition di identificare univocamente la window che ha generato il messaggio.

Una routine di gestione messaggi IDCMP

Il listato 5.2 è una versione estesa della routine di gestione degli eventi fornita nel quarto capitolo. Essa comprende tutti i possibili eventi di Input che un IDCMP può generare ed esegue una copia del messaggio ricevuto in modo tale da rispondere velocemente allntuition. Le subroutine all'interno del main possono essere eliminate se non sono necessarie. Il modo efficiente per fare ciò, naturalmente, consiste nell'eliminare le istruzioni case per ogni voce alla quale non si desideri rispondere. Comunque, il listato 5.2 gestisce tutti gli eventi base con un unica routine.

Alcune delle classi di messaggi nel listato 5.2 sono mutualmente esclusive. Cioè, se si specifica che si attende un particolare tipo di messaggio, si impedisce al task di prelevare ogni altro tipo di messaggio anche se vi sono più richieste. Le esclusioni sono:

- Se si seleziona DELTAMOVE, non si riceverà alcun evento MOUSEMOVE. (DELTAMOVE modifica il modo in cui devono essere riferiti i movimenti del mouse relativi all'ultima posizione).
- Se si seleziona VANILLAKEY, non si riceveranno eventi RAWKEY.(VANILLAKEY modifica il modo in cui vengono riferiti gli eventi di tastiera).

```
WhereInGadget(im)
            struct IntuiMessage *im;
{
            struct Gadget *g;

            g = (struct Gadget *)im->IAdress;

            if(im->MouseX - g->LefteEdge) < 4)
                        printf("Click vicino al bordo sinistro\n");
            else if((g->LeftEdge + g->Width-1 - im->MoseX) < 4)
                        printf("Click vicino al bordo destro\n");
            else
                        printf("Click vicino al centro sull'asse X\n");

            if((im->MouseY - g->TopEdge) < 4)
                        printf("Click vicino all'estremo superiore\n");
            else if(((g->LeftEdge + g->Height-1 - im->MouseY) < 4)
                        printf("Click vicino all'estremo inferiore\n");
            else
                        printf("Click vicino al centro sull'asse Y\n");
}
```

Listato 5.1

- Se si seleziona MOUSEBUTTONS, non si riceverà nessun evento GADGETDOWN o GADGETUP per i gadget allegati alle window. Comunque si riceveranno questi eventi per i gadget allegati ai Requester delle window. Il pulsante di selezione del mouse è normalmente usato per selezionare i gadget. L'evento consiste nella pressione o rilascio dei pulsanti del mouse. MOUSEBUTTONS ha la precedenza sulle informazioni relative ai gadget.

Il listato 5.3 contiene alcune routine per disabilitare quelle opzioni che non sono richieste per il programma di Paint.

Progettare un programma di Paint

Per progettare un programma di Paint, vi sono molte cose che bisogna saper controllare e generare. L'effetto principale del semplice programma di Paint che si trova in questo capitolo consiste nel far apparire un punto colorato nella posi-

zione in cui si trova il cursore nel momento in cui l'utente preme il pulsante del mouse stesso. Se il mouse viene mosso senza che sia premuto il pulsante non deve accadere nulla.

In più, permette di richiamare un Requester che può posizionare dei testi colorati in qualsiasi punto dell'area di disegno.

Questo programma non permette di salvare o caricare le figure create. Potrebbe essere interessante per il lettore creare un estensione di questo programma che faccia proprio questa operazione. Il gadget di chiusura della window permette di uscire dal programma.

```
/* event2.c */

extern struct Window *w;

GadgetDown(m) struct IntuiMessagge *m; { return(1); }

CoCloseWindow(m)
            struct IntuiMessage *m;
{
      return(FALSE);    /* permette al main() di chiudere la window */
}

HandleEvent(ms)
            struct IntuiMessage *ms;
{

      struct IntuiMessage localms;
      int i;
      UBYTE *s, *d;
      int result; /* result è ciò che viene ritornato dalla */
                  /* Routine e determina se il programma */
                  /* ha chiuso tutto e esce. Ritorna zero se */
                  /* è tutto chiuso e esce, ritorna un valore */
                  /* diverso da zero in caso contrario */
      s = (UBYTE *)ms;        /* sorgente...IntuiMessage */
      s = (UBYTE *)localms;   /* destinazione...IntuiMessage */

      for(i=0; i<sizeof(struct IntuiMessage); i++)
      {
            *d++ = *s++;      /* lo copia */
      }
      ReplyMsg(ms);           /* e gli risponde velocemente */

      /* si può usare la copia locale per elaborare il contenuto */
      /* del messaggio */
```

```
      /* Le Routine Response sono scritte per accettare l'indirizzo */
      /* dell'IntuiMessage in modo tale che ogni Routine possa */
      /* ritornare informazioni addizionali sul messaggio se */
      /* fosse richiesto. Se un utente è realmente a corto di tempo */
      /* potrebbe evitare di fare la copia */
      /* dell' IntuiMessage, e */
      /* passare l'indirizzo del messaggio originale */

switch(localms,Class)
      {
                  case SIZEVERIFY:
                        result = SizeVerify(&localms);
                        break;
                  case NEWSIZE:
                        result = NewSize(&localms);
                        break;
                  case REFRESHWINDOWS:
                        result = RefreshWindow(&localms);
                        break;
                  case MOUSEBUTTONS:
                        result = MouseBottons(&localms);
                        break;
                  case MOUSEMOVE:
                        result = MouseMove(&localms);
                        break;
                  case GADGETDOWN:
                        result = GadgetDown(&localms);
                        break;
                  case GADGETUP:
                        result = Gadgetup(&localms);
                        break;
                  case REQSET:
                        result = ReqSet(&localms);
                        break;
                  case MENUPICK:
                        result = MenuPick(&localms);
                        brea;
                  case CLOSEWINDOW:
                        result = DoCloseWindow(&localms);
                        break;
                  case RAWKEY:
                        result = RawKey(&localms);
                        break;
                  case REQVERIFY:
                        result = ReqVerify(&localms);
                        break;
                  case MENUVERIFY:
                        result = MenuVerify(&localms);
                        break;
                  case NEWPREFS:
                        result = NewPrefs(&loaclms);
                        break;
                  case DISKINSERTED:
                        result = DiskInserted(&localms);
```

```
                    break;
                case DISKREMOVED:
                    result = DiskRemoved(&localms);
                    break;
                case WBENCHMESSAGE:
                    result = WBenchMessage(&localms);
                    break;
                case ACTIVEWINDOW:
                    result = ActiveWindow(&localms);
                    bvreak;
                case INACTIVEWINDOW:
                    result = InactiveWindow(&localms);
                    break;
                case DELTAMOVE:
                    result = DeltaMove(&localms);
                    break;
                case VANILLIAKEY:
                    result = VanillaKey(&localma);
                    break;
                case INTUITICKS:
                    result = IntuiTicks(&localms);
                    break;
                default:
                    result = 1;
                    break;
                };
            return(result);

            /* Come per la versione precedente presentata nel libro */
            /* di HandleEvent, il valore ritornato da essa sarà */
            /* utilizzato per determinare se chiudere o no la */
            /* window e terminare il programma */
}
```

Listato 5.2

```
#define IM struct IntuiMessagge
      int   SizeVerify(ms)     IM *msg; { return (1); }
      int   NewSize(msg)       IM *msg; { return (1); }
      int [RefreshWindow(msg) IM *msg; ( return (1); }
      int   MouseMove(msg)     IM *msg; { return (1); }
      int [ReqSet(msg)         IM *msg; { return (1); }
      int   RawKey(msg)        IM *msg; { return (1); }
      int   VanillaKey(msg)    IM *msg; { return (1); }
      int   ReqVerify(msg)     IM *msg; { return (1); }
      int [ReqClear(msg)             IM *msg; { return (1); }
      int   MenuVerify(msg)    IM *msg; { return (1); }
      int [NewPrefs(msg)             IM *msg; { return (1); }
```

```
int     DiskInserted(msg)       IM *msg; { return (1); }
int     DiskRemoved(msg)        IM *msg; { return (1); }
int     WBenchMessage(msg)      IM *msg; { return (1); }
int     ActiveWindow(msg)       IM *msg; { return (1); }
int     InactiveWindow(msg)     IM *msg; { return (1); }
int     DeltaMove(msg)          IM *msg; { return (1); }

/* fine di stubs1.c */
```

Listato 5.3

Tra le altre cose bisogna sapere:

- Come selezionare lo schermo da usare
- Come determinare la posizione del mouse
- Come leggere lo status dei pulsanti del mouse
- Come progettare e utilizzare i menu
- Come specificare un colore
- Come progettare e usare i gadget.

Selezionare uno schermo

Naturalmente è necessaria di una certa varietà di colori per il programma di Paint. Il numero di colori che si possono utilizzare dipende dalla profondità (in termini di piani di Bit) dello schermo . Se si fa girare il programma di Paint sullo schermo del Workbench, si potrà scegliere solo tra quattro colori. Nel programma listato, invece, si apre uno schermo personalizzato. Con 16 colori tra i quali scegliere. Il listato 5.4 fornisce le istruzioni per aprire una window in uno schermo personalizzato.

```
/* myscreen2.h */

struct TexAttr TestFont = { "topaz.font",8,0,0};

struct NewsCreen ns = {
```

```
0,0,                    /* posizione iniziale */
320,200,4,              /* larghezza, altezza, profondità */
0,1,                    /* Detail pen, Block pen */
0,                      /* modo di visualizzazione */
CUSTOMSCREEN,           /* tipo di schermo */
&TestFont,              /* Font da utilizzare */
"Custom Screen",        /* nome dello schermo */
NULL};                  /* puntatore a Gadget addizionali */

struct NewWindow nw = {
0,10,                   /* posizione iniziale */
WIDTH,WEIGHT,           /* larghezza, altezza */
0,1,                    /* Detail pen, Block pen */
INTUITICKS | GADGETUP | GADGETDOWN |
MOUSEBUTTONS |MENUPICK |CLOSEWINDOW,
                        /* Flag IDCMP */
WINDOWCLOSE | ACTIVATE,
                        /* erano Flag di window BACKDROP | BORDERLESS */
NULL,                   /* puntatore al primo Gadget */
NULL,                   /* puntatore al CheckMark */
"Tiny Paint",           /* nome della window */
NULL,                   /* puntatore allo schermo */
NULL,                   /* puntatore alla superBitmap */
0,0,0,0,                /* ignorati/ non essendo una window */
                        /* dimensionata non si devono specificare i */
                        /* valori min/max permessi */
CUSTOMSCREEN };         /* tipo di schermo nel quale aprirla */

/* fine di myscreen2.h */
```

Listato 5.4

La struttura NewScreen

Nel quarto capitolo, gli schermi sono stati utilizzati ma non pienamente discussi. Poiché è importante creare degli schermi per qualsiasi programma di Intuition, è bene dare un'occhiata più da vicino ai vari campi della struttura dati NewScreen.

Una struttura NewScreen è definita come segue:

```
struct NewScreen
{
   SHORT LeftEdge,TopEdge,Depth,Width,Height;
   UBYTE DetailPen,BlockPen;
   USHORT ViewModes;
   USHORT Type;
```

```
    struct TextAttr *Font;
    UBYTE *DefaultTitle;
    struct Gadget *gadgets;
    struct BitMap *CustomBitMap;
};
```

LeftEdge e TopEdge per la maggior parte degli schermi sono settate a 0,0. Questo significa che lo schermo ha il suo angolo superiore sinistro coincidente con l'angolo superiore sinistro dell'area di visualizzazione. Height di solito è settata a 200 linee se il modo di visualizzazione è non interlacciato o a 400 se è interlacciato (256 e 512 per la versione PAL) .

Comunque, se si vogliono creare schermi suddivisi in modo particolare, si può manipolare lo schermo come si desidera. Per esempio, si possono aprire due schermi, uno che occupa la parte superiore dell'area di visualizzazione (LeftEdge e TopEdge settati a 0,0),e l'altro che occupa la parte inferiore dell'area stessa (LeftEdge e TopEdge settati a 0,100), ciascuno dei quali può usare il proprio set di colori e qualsiasi risoluzione sia disponibile.

Depth specifica il numero massimo di colori utilizzabili contemporaneamente, tra i quali si dovrà scegliere per disegnare sullo schermo. Depth può andare da 1 a 5 per selezionare rispettivamente 2, 4, 8, 16, o 32 colori.

DetailPen è il numero del registro contenente il colore usato per scrivere il nome dello schermo, BlockPen è quello usato per tracciare i gadget di profondità.

ViewModes è un gruppo di Bit molto importante che serve a specificare il modo di visualizzazione. In questo libro, viene utilizzata solo la bassa risoluzione (che corrisponde ad un valore di ViewModes uguale a 0), l'alta risoluzione (HIRES), e il modo interlacciato (LACE). In bassa risoluzione si possono aprire schermi ampi fino a 354 Pixel orizzontali. In alta risoluzione si possono raggiungere i 704 Pixel. Inoltre in bassa risoluzione in verticale si utilizzano 200 linee (a meno che non si specifichi un diverso valore). Se si specifica LACE come parte dei ViewModes, si può creare uno schermo che contiene circa 400 (512 per la versione PAL) linee verticali. Questo aumento di risoluzione è controbilanciato da uno sfarfallio insopportabile con alcune combinazioni di colori.

Il campo Type determina se lo schermo deve essere un WBENCHSCREEN (come quello del Workbench) o un CUSTOMSCREEN. Di solito c'è un solo schermo di Workbench nel sistema. Si noti che il campo Type nella struttura dati NewScreen ha effetto sul modo in cui verranno utilizzati i restanti campi di dati.

Il campo Font descrive il tipo di caratteri che dovrà essere usato come default per tutti i menu, i nomi delle finestre, e qualsiasi altro testo. DefaultTitle è un puntatore a una stringa contenente il nome dello schermo, visibile sulla barra superiore. Il puntatore ai gadget non è utilizzato - gli schermi non hanno altri gadget oltre a quelli previsti dal sistema.

Il puntatore CustomBitMap permette di allocare e definire una speciale area di memoria che il sistema dovrà utilizzare per visualizzare il nuovo schermo. Se non si definisce una Bitmap propria, il sistema allocherà automaticamente la memoria necessaria per lo schermo. Quando lo schermo si chiude, questa memoria allocata viene automaticamente restituita al sistema; in questo libro, tale parametro è sempre impostato a NULL.

La struttura NewWindow. Un'altra importante struttura di Intuition è la NewWindow, ed è definita come segue:

```
struct NewWindow
{
   SHORT LeftEdge,TopEdge;
   SHORT Width,Height;
   UBYTE DetailPenBlockPen;
   ULONG IDCMPFlags;
   USHORT Flags;
   struct Gadget *FirstGadget;
   struct Image *CheckMark;
   UBYTE *Title;
   struct Screen *Screen;
   struct BitMap *BitMap;
   SHORT MinWidth,MinHeight;
   SHORT MaxWidth,MaxHeight;
   USHORT Type;
};
```

LeftEdge e TopEdge precisano la posizione della window rispetto all'angolo superiore sinistro dello schermo. Width e Height precisano quanto deve essere larga e alta la window, inclusi i bordi e la barra. Come per gli schermi, DetailPen e BlockPen specificano come debbano essere disegnati i gadget (il gadget di chiusura, i gadget di profondità, e il gadget di ridimensionamento).

I Flag IDCMP sono stati discussi nel quarto capitolo.

I valori dei Flag specificano i gadgets che devono essere acclusi alla window quando viene disegnata e altre sue caratteristiche, e sono:

WINDOWSIZING	Se questo Flag è settato la window può essere ridimensionata. Si devono anche specificare i limiti del ridimensionamento nelle variabili MinWidth, MaxWidth, MinHeight, e MaxHeight. Attenzione: Width deve essere un valore compreso tra MinWidth e MaxWidth; Height deve essere compreso tra MinHeight e MaxHeight.
WINDOWDRAG	Se questo Flag viene settato, la window può essere riposizionata.
WINDOWCLOSE	Se questo Flag viene settato, la window possiede un gadget di chiusura. Si noti che se si vuole ricevere un messaggio sulla selezione del gadget di chiusura da parte dell'utente, si deve settare anche CLOSEWINDOW nei parametri dei Flag IDCMP.
WINDOWDEPTH	Se questo Flag viene settato, la window può essere spostata in profondità. Se è stata creata una window BACKDROP, quando l'utente utilizzerà questo gadget non accadrà nulla.
BACKDROP	Se questo Flag viene settato, la window viene aperta dietro a tutte le altre window e non è possibile portarla in primo piano. In questo modo si risparmia un pò di memoria poiché i disegni all'interno di questa window utilizzerebbero la Bitmap dello schermo. Le window che vengono aperte di fronte a una window BACKDROP allocano uno spazio a parte nella memoria.
REPORTMOUSE	Se questo Flag viene settato, si verrà informati da Intuition su tutti i movimenti del mouse: questo implica un gran numero di eventi in Input e potrebbe essere poco desiderabile.
GIMMEZEROZERO	L'intera area all'interno dei confini della window è considerata l'area di disegno. Il punto 0, 0 è nell'angolo superiore sinistro sotto la barra del titolo. Dopo che una window è stata aperta, vi sono due gruppi di posizioni del mouse che vengono mantenuti da Intuition. Il primo

è costituito dai valori MouseX e MouseY. Questi rappresentano le posizioni del mouse rispetto alla window completa, compresi i bordi. Il secondo set è composto dalle variabili GZZMouseX e GZZMouseY. Questo secondo set tiene traccia del mouse se è stato selezionato GIMMEZEROZERO. Esso corregge le coordinate in modo che si riferiscano all'angolo superiore sinistro della reale area di disegno attuale (esclusi il titolo, i bordi ed i gadget).

BORDERLESS — Se questo flag viene settato la window non avrà alcun bordo. Se però si seleziona WINDOWDRAG i bordi vi saranno comunque perchè il sistema li reputa necessari per lo sposatamento della window.

ACTIVATE — Se questo Flag viene settato, la window non appena aperta diviene quella attiva. Si potrebbe voler mantenere attiva una diversa window mentre viene aperta una nuova finestra che magari non attende nessun Input.

RMBTRAP — Se viene settato questo Flag, gli eventi riguardanti il pulsante destro del mouse vengono deviati dal Intuition e inviati al task dell'utente attraverso gli IDCMP. Diversamente, tutte le informazioni riguardanti i pulsanti del mouse sono direttamente inviati a Intuition che li utilizza per controllare i propri menu.

I seguenti Flag sono mutualmente esclusivi. Si può selezionare solo uno di essi.

SMART_REFRESH — Intuition salva il contenuto della window e, automaticamente, si preoccupa di ridisegnarla (Refresh) nel caso in cui venga coperta e poi scoperta da altre window.

SIMPLE_REFRESH — L'Intuition non salva nulla di ciò che è stato disegnato nelle aree coperte. Di ciò si incarica il task utente. Se si vuole utilizzare questo metodo, si devono richiedere gli eventi REFRESHWINDOW nei Flag IDCMP.

SUPER_BITMAP — C'è una superBitmap associata a questa window. Le window superBitmap sono state discusse nel quarto capitolo.

Il parametro FirstGadget è un puntatore ad una lista di gadget da allegare alla window. Quando uno di questi gadget viene selezionato, si verrà informati da In-

tuition sempre che siano stati settati i parametri GADGETDOWN e GADGETUP nei Flag IDCMP. I gadget sono esaminati più a fondo più avanti in questo stesso capitolo.

Il parametro CheckMark è un puntatore a una struttura dati Image di Intuition che descrive la forma di una immagine personalizzata del simbolo di selezione per elementi dei menu CheckMark. Se è stata settata la striscia di menu (SetMenuStrip) per la window dell'utente, il menu apparirà sulla barra dello schermo quando tale window è attiva e il pulsante destro è premuto. All'interno di quel menu, ovunque si debba visualizzare un CheckMark, se ne potrà utilizzare uno personalizzato. Se questo valore è NULL, il sistema usa un CheckMark di default.

Il parametro Type è critico. Se si specifica WBENCHSCREEN viene ignorato qualsiasi valore del parametro Screen nella struttura NewScreen e la propria window sarà aperta in uno schermo Workbench. Se si specifica CUSTOMSCREEN, si deve avere un puntatore valido a uno schermo memorizzato nel parametro Screen. Poi, per aprire la propria window nello schermo personalizzato, si usi questa sequenza:

```
struct Screen *s;
struct Window *w;
s = OpenScreen(&myNewScreen);
if(s)
{
   w = OpenWindow(myNewWindow);
   if(w)
   {
                                   /* si continua */
   }
}
```

La posizione del mouse

Si può trovare la posizione corrente del mouse nella struttura dati della window che è stata aperta. Se w è un puntatore alla window, allora si possono utilizzare queste istruzioni:

```
CurrentMouseX = w->MouseX;
CurrentMouseY = w->MouseY;
```

Comunque, questa non sarà necessariamente la posizione aggiornata del mouse. Al posto della formula precedente, si può ottenere la posizione del mouse dai messaggi ricevuti dalla porta IDCMP, (im è un puntatore ad un IntuiMessage):

```
CurrentMouseX = im->MouseX;
CurrentMouseY = im->MouseY;
```

Nel programma di Paint, se il pulsante di selezione del mouse è premuto, ogni volta che un evento di tempo si verifica (INTUITICKS), sarà disegnato un punto colorato. Questo è ciò che fa la routine IntuiTicks del listato 5.5. Una variabile globale di nome selectbutton tiene traccia dello status di pressione (down = 0) e di rilascio (up = 1). Le variabili w e rp sono i puntatori alla window e alla RastPort.

I pulsanti del mouse

Selezionando il Flag IDCMP MOUSEBUTTONS nella struttura NewWindow, si ottengono messaggi circa lo status del pulsante di selezione del mouse. Il pulsante di menu del mouse è di solito riservato allIntuition per utilizzare i menu. Se si necessita anche delle informazioni circa il tasto di menu, allora si dovrà specificare RMBTRAP tra i Flag della NewWindow e si avranno sotto controllo entrambi i pulsanti del mouse.Il listato 5.6 è la subroutine mousebuttons. Essa riporta le informazioni sullo status del pulsante di selezione del mouse.

Progettare e utilizzare i menu

Per progettare un menu a discesa per il programma di Paint, si devono specificare le caratteristiche del menu desiderato, le sue voci e le sottovoci.

```
/* ticks.c */

extern int, selectbutton;
int CurrentMouseX, CurrentMouseY;
```

```
extern struct RastPort *rp;

IntuiTicks(im)
            struct IntuiMessage *im;
{
            if(!selectionbutton) return(1);
                        /* nessuna azione se il pulsante è sollevato */
            CurrentMouseX = im->MouseX;
            CurrentMouseY = im->MouseY;

            WritePixel(rp, CurrentMouseX, CurrentMouseY);
            return(1);
}
```

Listato 5.5

```
/* mousebuttons.c */

int selectbutton; /* global - viene premuto il pulsante? */

MouseButtons(im);
struct IntuiMessage *im;
{
            if(im->Code == SELECTDOWN )
            {
                  /* il pulsante è appena stato premuto */
                  selectbutton = TRUE;
                  IntuiTicks(im);   /* disegna un Pixel */
                  return(1);
            }
            if(im->Code == SELECTUP )
            {
                  /* il pulsante è appena stato rilasciato */
                  selectbutton = FALSE;
                  return(1);
            }
            return(1);  /* nessuna gestione del pulsante destro */
}
```

Listato 5.6

Il menu appare al di sopra della barra dello schermo, rimpiazzando localmente la barra stessa quando è premuto il pulsante di menu del mouse.

Ogni menu è completamente specificato dalla struttura dati menu. I menu sono posizionati in modo assoluto non relativo. Si specifica a quale posizione orizzontale e verticale si vuole che cominci l'estremità destra del menu. Ognuna delle voci di menu è posizionata relativamente al menu al quale appartiene. Così se si muove ciò a cui è allegato il menu, esso stesso si muoverà di conseguenza. Se una voce di un menu a discesa ha una sottovoce ad essa allegata, tale sottovoce sarà posizionata relativamente alla voce alla quale è allegata. Così se si sposta la voce del menu, la sottovoce si sposterà di conseguenza.

Il listato 5.7 crea un menu per il programma di Paint. Vi sono due parti principali. La prima permette all'utente di selezionare il colore di scrittura, la seconda permette all'utente di digitare un testo all'interno della propria figura. Il programma fornisce un requester che chiede all'utente quale è il testo da aggiungere e dove posizionarlo nella figura.

Voci e sottovoci

Per meglio chiarire il programma del listato 5.7, ecco spiegato come si collegano i menu e le relative voci:

- Per ogni singola parte del menu si forma una lista che specifica una certa quantità di testo selezionata e dimensionata dal programmatore. Tale stringa rimpiazza il nome dello schermo sulla barra quando viene premuto il pulsante di menu del mouse.

```
/* initmenu.c */

char *menunames[2];

extern struct Menu      menu[2];
struct MenuItem         textitem;
extern struct MenuItem  coloritem[32];
extern structImage      colorImage[32];

/* un valore di 32 supporta una profondità fino a 5 BitPlane */
```

```
IntuiMenu()
{
            struct Menu *menuptr;

            int n;
            int leftside;
            leftside = 2;

            menunames[0]="Color";
            menunames[1]="Text";

            menuptr = menu[0];

            for (n=0; n<2; n++)
            {
                  menuptr->LeftEdge = leftside;
                  menuptr->TopEdge = 0;
                  menuptr->Width = 9 * strlen(menunames[n]);
                  leftside += (menuptr->Width + 2);
                  menuptr->Height = 10;
                  menuptr->Flags = MENUENABLED;
                  menuptr->MenuName = menunames[n];

                  if(n == 0)
                  {
                        menuptr->FirstItem = coloritem[0];
                        menuptr->NextMenu = menu[1];
                  }
                  else
                  {
                        menuptr->FirstItem = textitem;
                        menuptr->NextMenu = NULL;
                  }

            menuptr->JazzX = 0; menuptr->JazzY = 0;
            menuptr->BeatX = 0; menuptr->BeatY = 0;
            menuptr++;
            }
            InitTextItem();
            InitColorItems(DEPTH);
            return(0);
}

/* Ogni Menu deve avere almeno una voce. Si crea una singola */
/* voce da allegare al menu di testo */

struct IntuiText textitemtext =
            { 1,0,JAM2,0,0,NULL,"Insert Text",NULL};

            /* penna principale, penna di sfondo, modo di disegno */
            /* posizione (sinistra, in alto) relativamente alla */
            /* posizione (sinistra, in alto) del rettangolo */
            /* che racchiude il menu */
            /* Font (NULL significa che verrà usato il Font */
```

```
            /* di default), una stringa di caratteri */
            /* IntuiText, linka la seguente IntuiText se c'è; */
            /* in questo caso è NULL */

IntuiTextItem()

            textitem.NextItem = NULL;/* una sola voce */

            /* quando i menu contengono testi */
            /* si deve puntare a IntuiText per riempirli */

            textitem.ItemFill = (APTR)&textitem;

            textitem.LeftEdge = 0;  /* limite sinistro, */
            textitem.TopEdge = 8;   /* linea successiva, */
                                    /* immediatamente sotto */
                                    /* il menu stesso. */
            textitem.Width = 9*14;  /* "Insert Text" */
                                    /* più spazio per */
                                    /* l'acceleratore */
            textitem.Height = 10;

            /* Questa voce di menu contiene un testo */

            textitem.Flags= HIGHCOMP | ITEMTEXT | COMMSEQ | ITEMENABLED;

            /* Non si cerca di escludere altre selezioni */
            textitem.MutualExclude = 0;

            /* Niente da visualizzare quando questo rettangolo */
            /* viene selezionato */
            textitem.SelectFill = NULL:

            /* Invece di utilizzare il testo del menu per selezionare */
            /* questa voce, un'utente dovrebbe essere in grado */
            /* di usare il tasto di comando */
            /* (il tasto Amiga di sinistra) con */
            /* un altro tasto (qui una t minuscola) per operare */
            /* una interruzione. Questo programma rileva l'evento */
            /* proprio come se fosse stato selezionato la voce di */
            /* menu, senza che il menu nemmeno appaia. Questo */
            /* funziona se e solo se è settato il Flag COMMSEQ */
            /* per questa MenuItem */

            textitem.Command = 't';
            /* <left-Amiga>t crea l'interruzione di comando */
            /* non vi sono sottomenu di cui preoccuparsi */
            textitem.SubItem = NULL;
            textitem.NextSelect = 0;
}

/* Fornisce tante immagini a colori in questo menu di selezione */
/* quanti sono i colori disponibili a secondo della profondità */

InitColorItems( depth )
```

```
        SHORT depth;
{
        SHORT n,colors;
        struct Image *colorimageptr;
        struct MenuItem *coloritemptr;
        struct MenuItem *nextcoloritemptr;
        colors = palette[depth-1];

        colorimageptr = coloimage[0];
        coloritemptr = coloritem[0];

        nextcoloritemptr = coloritemptr;

        for(n=0; n<colors; n++)
        {
                /* ora... setta i puntatori delle voci */
                nextcoloritem++;

                coloritemptr->NextItem = nextcoloritemptr;
                coloritemptr->ItemFill = (APTR)colorimageptr;
                coloritemptr->LeftEdge = 2 + CW*(n % 4);
                coloritemptr->TopEdge = CH * (n/4);
                coloritemptr->Width = CW;
                coloritemptr->Height = CH;
                coloritemptr->Flags = ITEMSTUFF;
                coloritemptr->MutualExclude = 0;
                coloritemptr->SelectFill = NULL;
                coloritemptr->Command = 0;
                coloritemptr->SubItem = NULL;
                coloritemptr->NextSelect = 0;

                /* centra l'immagine */
                /* nel rettangolo selezionato */
                colorimageptr->LeftEdge = 1;
                colorimageptr->TopEdge = 1;

                /* rende l'immagine un po' più piccola */
                /* del rettangolo nel quale si trova */

                colorimageptr->ImageData = NULL;
                colorimageptr->PlanePick = 0;
                colorimageptr->Depth = depth;

                /* Non fornisce un'immagine da BitMap da */
                /* inserire nel rettangolo. Invece, traendo */
                /* vantaggio dall'uso di PlanePick e */
                /* PlaneOnOff lo fa direttamente */

                colorimageptr->ImageData = NULL;
                colorimageptr->PlanePick = 0;
                colorimageptr->PlaneOnOff = n;

                coloritemptr++;
                        /* il successivo in ogni Array */
                colorimageptr++;
```

```
            }
            coloritemptr--;   /* dopo il Loop, questo puntatore */
                                  /* punta un elemento dell'Array */
                                  /* che si trova oltre quello che */
                                  /* vogliamo modificare di fatto */
            coloritemptr->NextItem = NULL;/* fine della catena */
            return(0);
}
```

Listato 5.7

- In aggiunta al puntatore alla successiva voce del menu, ogni singola parte del menu ha anche un puntatore alla voce del menu che scende quando quell'area attiva viene selezionata.
- Le voci del menu formano una lista che descrive il menu a discesa. Ogni voce del menu punta anche a un sottomenu opzionale che si apre quando è selezionata la voce stessa. è permesso un solo livello di sottomenu.

Inizializzare i menu

I menu non devono sovrapporsi. L'Intuition controlla la posizione del mouse nella sequenza che è stata fornita nella lista linkata delle strutture dati menu. I controlli vengono fatti a livello dei rettangoli forniti con LeftEdge, TopEdge, Width, e Height. Se vengono creati due menu che si sovrappongono, quello che è più vicino all'inizio della lista ha la precedenza. Per esempio, si supponga di creare due menu - uno per i colori (Color) e uno per i testi (Text) - e di specificarli nel modo mostrato in figura 5.1. Se l'ordine dei menu è:

```
menu[0].Next = &menu[1];
menu[1].Next = NULL;
```

allora se il menu Text è collegato al menu[0], l'utente può utilizzarlo. Ma se esso è linkato al menu[1], ogniqualvolta l'utente si trova all'interno del rettangolo di selezione di Color, il menu Color viene selezionato. Allora non è assolutamente possibile selezionare il menu Text. L'esempio sopra listato usa le variabili locali a sinistra per impedire una sovrapposizione.

Il flag MENUABLED è usato per entrambi i menu del programma. Questo fornisce un aspetto tipico dei menu di testo. Se non viene specificato nessuno dei flag di menu (i parametri flag sono tutti settati a zero), il menu non può essere selezionato e appare retinato.

Un menu deve contenere dei testi, diversamente dalle voci dei menu stessi che possono contenere anche immagini interessanti. Però si può essere creativi ed utilizzare dei font bizzarri per visualizzarne il testo invece di quello standard specificato al momento dell'apertura dello schermo. Il Font dello schermo può contenere caratteri bizzarri, e caratteri normali per visualizzare altri testi. Sta tutto al programmatore.

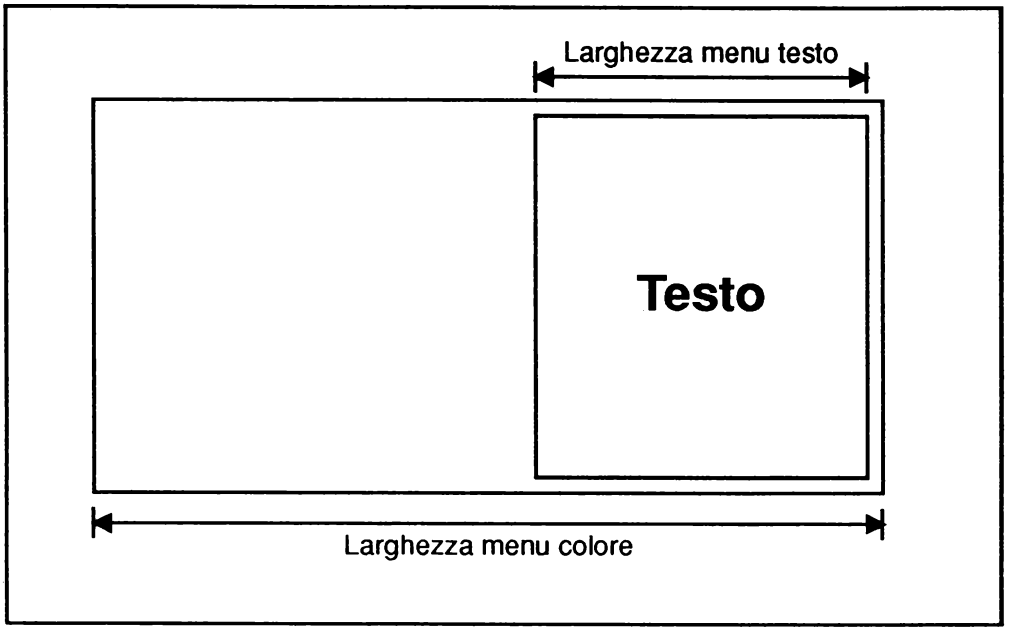


Figura 5.1

Inizializzare le voci dei menu

A differenza di quanto accade per i menu, oltre alle sovrapposizioni bisogna stare attenti alle mancate sovrapposizioni quando si posizionano i vari rettangoli che descrivono le voci dei menu. Questo potrebbe sembrare contraddittorio, ma quando si crea una lista di voci di menu, Intuition valuta le liste delle aree rettangolari. Per conto suo, Intuition crea un contenitore rettangolare grande abbastanza per contenere interamente tutti i rettangoli specificati dal programmatore, sia i rettangoli di selezione (LeftEdge, TopEdge, Width, e Height) sia la regione rettangolare che potrebbe essere definita nel visualizzare un testo su di una immagine.

Per esempio, se si specifica che una voce di menu deve essere larga solo 12 pixel, ma poi si dice al sistema di visualizzare in quello spazio la parola Text in font Topaz, il tipo 80 colonne di larghezza 32 pixel, Intuition crea una voce di menu che è larga almeno 32 pixel in modo che possa starci qualsiasi cosa si visualizzi.

In più, Intuition si regola mediante dei canoni suoi propri per decidere dove una voce di menu possa realmente essere posizionata rispetto al menu al quale è allegata. Il rettangolo che viene creato dad Intuition per contenere le voci scelte dal programmatore comincia almeno tanto a sinistra quanto l'estremità sinistra del suo menu e si estende almeno tanto a destra quanto l'estremità destra del suo menu, a prescindere dai valori di posizionamento e di larghezza che si erano richiesti. Si può chiedere ad Intuition di posizionare una voce in un luogo al di fuori della normale zona ammessa, ma come mostrato in figura 5.2, Intuition disegna il rettangolo di contenimento secondo le proprie regole.

Perfino se si richiede che le parole Insert Text inizino alla seconda posizione mostrata nella figura, Intuition estenderà ancora il rettangolo di contenimento a sinistra in modo che l'utente possa cadere direttamente all'interno dell'area della voci del menu. Altrimenti, non ci sarebbe il modo di scendere nell'area delle voci del menu e di proseguire lungo le voci stesse. Si noti che le voci non possono essere selezionate finché il mouse non discende all'interno della loro area, e poi non scorre lungo il rettangolo di selezione.

Quando si progetta un menu o un sottomenu, si deve fare in modo che ci sia sempre una certa quantità di sovrapposizione tra le voci adiacenti. Quando l'u-

tente sta selezionando dal menu almeno una voce sarà sempre attiva evitando confusioni all'utente.

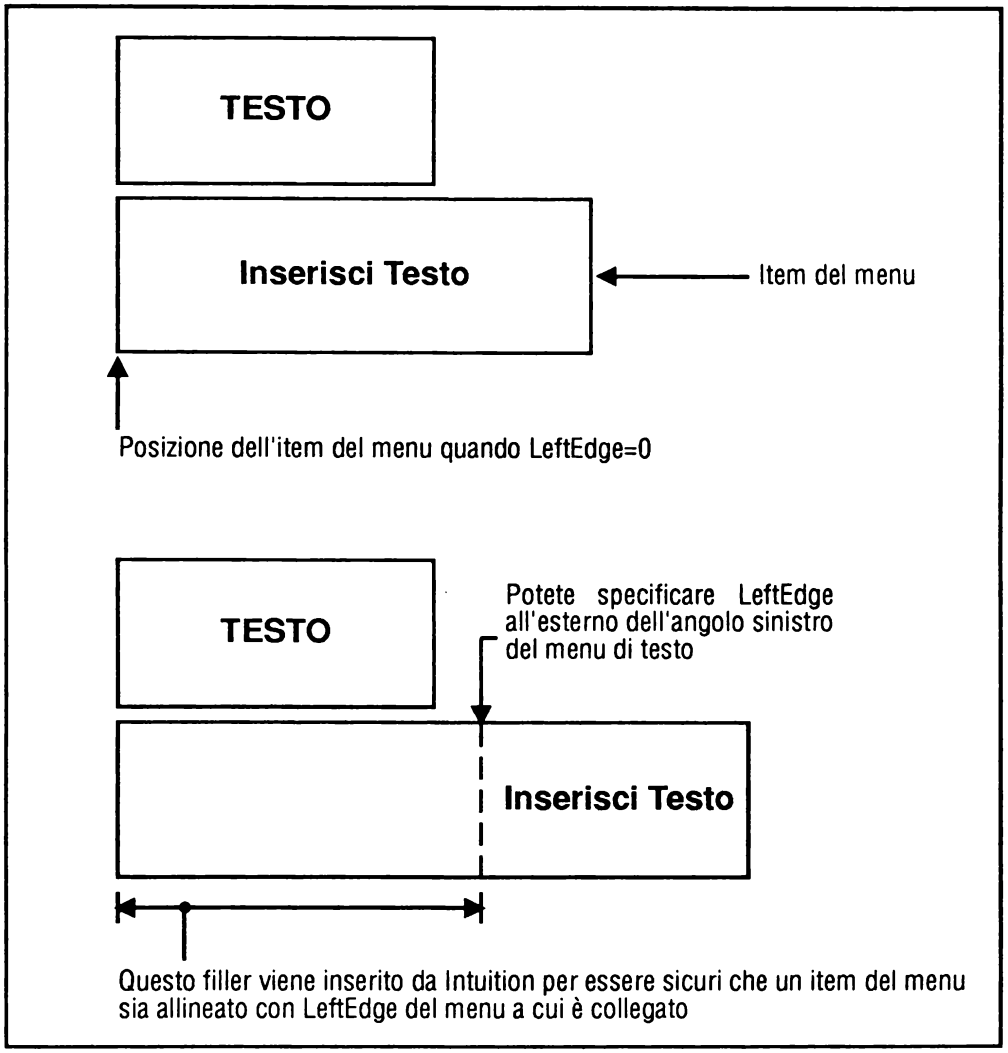


Figura 5.2

Testi o Immagini. Quando si progettano le voci dei menu, si può scegliere se in esse devono apparire dei testi o dei dati che formino delle immagini. Il testo prende la forma della struttura dati IntuiText; esso richiede che i flag della struttura MenuItem contengano il parametro di flag ITEMTEXT. Se questo flag viene usato, sia ItemFill sia SelectFill sono specificati come puntatori a una voce testo.

Se ITEMTEXT non viene specificato tra i flag, sia ItemFill sia SelectFill possono puntare a dati di un'immagine. questo fornisce una maggiore libertà per progettare ciò che deve essere posto nelle voci di menu. ItemFill dice ad Intuition cosa deve essere disegnato all'interno della voce del menu non appena essa appare. SelectFill dice ad Intuition cosa far apparire quando la voce del menu viene messa in evidenza.

Nel listato 5.7 le strutture dati ColorItem e ColorImage sono inizializzate contemporaneamente, poiché sono usate solo immagini in questa parte del programma di Paint. La struttura dati Image permette di specificare dove mettere l'immagine relativamente all'angolo superiore sinistro del suo rettangolo definito, e specifica le dimensioni della immagine in pixel. Permette inoltre di risparmiare della memoria fornendo dei controlli specifici su come deve essere disegnata l'immagine (nei parametri PlanePick e PlaneOnOff).

Si può per esempio, creare un'immagine a quattro colori da visualizzare in varie posizioni su di uno sfondo a 16 colori. In varie posizioni differenti, magari, si vuole utilizzare un diverso set dei 16 colori disponibili per visualizzare la propria immagine a quattro colori. L'Intuition fornisce un aiuto che viene spiegato nella discussione che segue.

Immagini a colori. I colori vengono formati sullo schermo mediante una combinazione binaria di bit situati in diversi piani, e questo per ogni pixel dello schermo. Per selezionare il colore di pixel numero 1 (a prescindere da quale sia il colore correntemente assegnato al registro numero 1), si usi SetAPen(rp,1) e WritePixel(rp,x,y). Questo implica il coinvolgimento di uno o più piani di bit, a secondo del numero di piani di bit usato per costruire l'immagine. Se si utilizzano quattro piani di bit (cioè 16 colori disponibili), allora la combinazione dei bit che viene scritta per il singolo pixel per il colore numero 1 è la seguente:

```
Plane          3 2 1 0
Bit            0 0 0 1
```

per il colore numero 10:

```
Plane           3 2 1 0
Bit             1 0 1 0
```

Immaginiamo di definire un'immagine a quattro colori così fatta:

```
1100330011

1100330011

0110330110

0011331100

0022332200

0220330220

2200330022

2200330022
```

Ognuno dei valori numerici rappresenta un pixel di uno dei quattro possibili colori di questa immagine. Due piani di bit combinati formeranno il pattern desiderato. I pattern di bit per due piani sono mostrati nella figura 5.3.

Se si prendono questi bit, come mostrato, e si disegnano sempre nei BitPlane di sistema 0 e 1, allora si otterrà una figura che avrà sempre come mappa di colori quella formata con il colori 1, 2, 3, e 4. Intuition fornisce un'alternativa: si possono scegliere i piani all'interno dei quali visualizzare la propria immagine. Se si scelgono i piani 3 e 1, allora il piano 0 della propria immagine viene posto all'interno del più basso dei piani scelti (1) il piano 1 dell'immagine viene posto nel più basso dei piani successivi che sono stati scelti (3 in questo caso).

In più, per i piani non scelti, PlaneOnOff dice ad Intuition cosa farne, all'interno del rettangolo di contenimento dell'immagine. Se un Bit di un piano non scelto è zero, allora l'intero rettangolo immagine è azzerato, altrimenti viene interamente impostato ad 1.

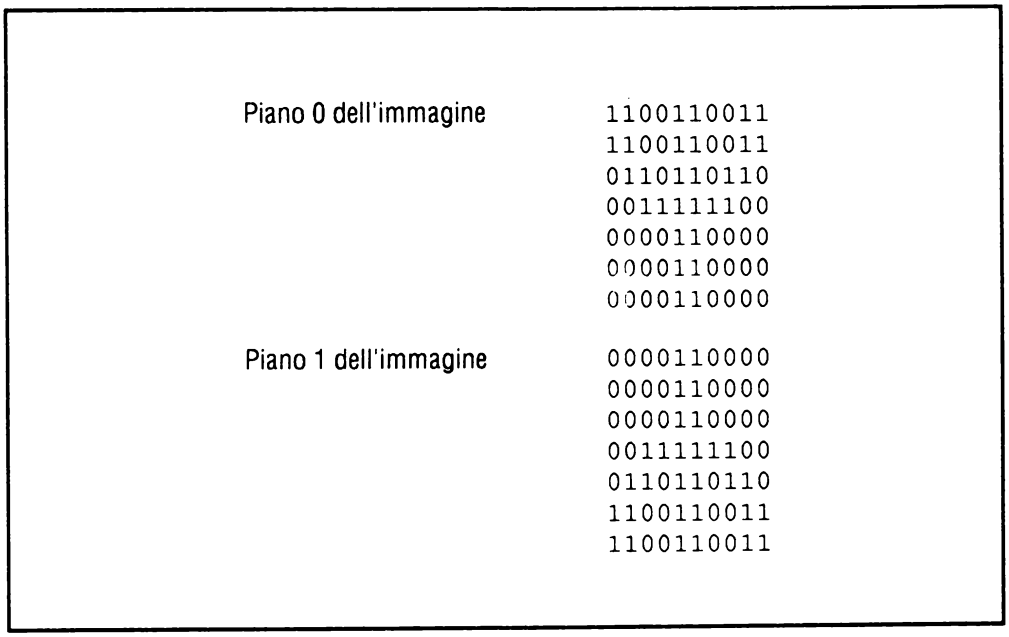


Figura 5.3

Per esempio dove sono prelevati i piani 3 e 1 non lo sono i piani 2 e 0. I bit in quelle posizioni all'interno di PlaneOnOff determinano ciò che accade. Mettiamo che questi due bit siano entrambi a 1. Ecco cosa succede:

```
myImage.PlanePick = 0x0a;        /* vengono prelevati i Bit 3 e 1 */
myImage.PlaneOnOff = 0x05;       /* i Bit 2 e 0 sono ON... essi */
                                 /* non sono stati prelevati, */
                                 /* così questa volta contano */
```

Così, i dati dell'immagine vanno nei piani 3 e 1 e degli uno binari vanno nei piani 0 e 2 all'interno del rettangolo di definizione dell'immagine. I colori risultanti quando viene disegnata l'immagine non sono più 0, 1, 2, e 3, ma sono invece quattro dei colori di sistema disponibili. PlaneOnOff forza i piani 2 e 0 a essere sempre uno:

```
Plane          3 2 1 0
Bit            1 X 1 X
```

Così, questi sono i colori che l'immagine può contenere:

```
1 0 1 0 = colore 10

1 0 1 1 = colore 11
```

```
1 1 1 0 = colore 14

1 1 1 1 = colore 15
```

Il programma di Paint possiede un menu di colori che produce solo rettangoli colorati, così esso ignora i dati dell'immagine, poiché che non c'è nessuna immagine di fatto. Il programma specifica PlanePick = 0, cioè non disegna l'immagine dentro alcun piano. Questo significa che il colore è determinato esclusivamente da PlaneOnOff per ognuno dei rettangoli, mettendo gli zero e gli uno nei piani appropriati e quindi settando il colore del rettangolo senza specificare alcuna immagine da usare.Se si vogliono poi utilizzare vere immagini, il formato per immagazzinarle è un Array di uword.

Il dato del bit viene immagazzinato in modo Left-Justified in una matrice rettangolare di word. L'Array è largo abbastanza da contenere l'oggetto in tutta la sua larghezza e lungo abbastanza da contenere l'oggetto in tutta la sua altezza. I Bit che si trovano oltre il margine destro rimangono semplicemente inutilizzati. Per esempio, se si ha un oggetto da visualizzare largo 29 bit, esso occupa due word (32 Bit) di larghezza nel formare l'immagine. Quando l'immagine viene disegnata, solo i bit a sinistra della 29esima posizione vengono usati in fase di disegno. Gli altri tre bit sono totalmente ignorati. Il prospetto dell'immagine dato sopra può essere inserito come mostrato nel listato 5.8; il resto dell'immagine sarà definito come segue:

```
struct Image myImage = {
   1,1,10,8,2,myimageshape[0],0x05,0x0a};

                              /* LeftEdge, TopEdge, Width, Height */
                              /* Depth, <image>, PPick, POnOff */
```

I Checkmark. Intuition permette di utilizzare nuovi simboli di Checkmark che appaiono a fianco delle voci quando vengono selezionate. I Checkmark permettono all'utente di sapere quale delle opzioni è correntemente attiva. Si può rendere un'opzione attiva settando il flag CHECKIT nella struttura dati MenuItem. Per usare l'immagine di Checkmark, ci si assicuri di lasciare abbastanza spazio lungo il margine sinistro della voce di menu all'interno del quale Intuition visualizzerà il Checkmark (ogniqualvolta viene selezionata questa opzione).

Il flag CHECKIT rappresenta un modo conveniente per fa si che Intuition tenga traccia dei vari flag. Nelle routine di elaborazione degli eventi, si può voler sapere lo stato corrente del flag CHECKED per le varie voci di menu prima di

procedere. Si potrebbe voler specificare il Flag MENUTOGGLE, che dice ad Intuition di inserire lo stato di CHECKED, ogniqualvolta la voce viene selezionata.

```
/* myimmageshape.h */

SHORT myimmageshape[] = {     /* piano 0 dell'immagine */
                        0xccc0,
                        0xccc0,
                        0x5d80,
                        03xf00,
                        0x0c00,
                        0x0c00,
                        0x0c00,
                                    /* piano 1 dell'immagine */
                        0x0c00,
                        0x0c00,
                        0x0c00,
                        0x0c00,
                        0x3f00,
                        0x5d80,
                        0xccc0,
                        0xccc0
                  };

      struct image myImage = {
                        1,1,10,8,2, &myimageshape[0],0x05,0x0a };

            /* LeftEdge, TopEdge, Width, Height */
            /* Depth, <image>, PPick,POnOff */
```

Listato 5.8

Cioè, se attualmente è CHECKED, inverte il suo stato, e viceversa. Si possono definire delle immagini personali da usare come Checkmark nella struttura dati NewWimdow. (Ogni window può avere i propri menu così come i propri Checkmark personalizzati). Il parametro Checkmark nella struttura dati NewWindow punta a NULL (se si desidera usare il Checkmark di default) oppure a una struttura dati Image.

Perché si dovrebbe voler definire un proprio Checkmark? Bene, il Checkmark di default del sistema è largo circa otto pixel e alto altrettanto. Si potrebbe volere un immagine completamente differente. Si potrebbe voler progettare il proprio Checkmark, magari un punto o una freccia, o magari invisibile - largo e alto un solo pixel e dello stesso colore del menu - in modo che non appaia nemmeno.

Progettando un proprio Checkmark invisibile, si può trarre vantaggio dalla manipolazione del flag CHECKED di Intuition. Il Checkmark potrebbe essere disegnato e cancellato senza alcun coinvolgimento dell'utente, sapendo però attraverso lo stato corrente del flag CHECKED se un'opzione è stata scelta o no. Si potrebbe, per esempio, usare lo stato di questo flag per decidere quale immagine presentare all'utente alla prossima visualizzazione del menu. Si potrebbe anche voler cambiare con alternanza l'immagine per ItemFill e SelectFill basandosi sullo stato del flag.

Mutue esclusioni. MutualExclude, nella struttura dati MenuItem, è una word long (32 Bit) che contiene un bit di posizione per ogni possibile voce di menu. Se c'è un bit settato a uno in una particolare posizione, allora quella voce deve essere esclusa quando è selezionata questa voce.

Le mutue esclusioni sono desiderabili in quei menu dove una sola voce può essere scelta tra le varie disponibili. Intuition permette una situazione ideale nella quale viene dichiarato "se selezioni me allora devi disattivare il seguente gruppo di voci". Così l'esclusione di altre opzioni è pienamente specificabile.

Mettiamo che vi siano queste quattro voci in un menu:

```
PATTERNFILL:

   BLANK
   SOLID COLOR
   CROSS HATCH
   STRIPED
```

Se si utilizzano solamente i Checkmark, nel ricevere il messaggio che una di queste voci di menu viene selezionata, si dovrà scorrere l'intera lista delle voci di menu, resettare il flag CHECKED per quelle voci che non sono più attive, e settarlo per quella che è selezionata.

Utilizzando MutualExclude si può risparmiare questo lavoro. Quando viene selezionata una nuova voce, il task non dovrà preoccuparsi di fare nulla sulle restanti voci della lista. Tutte quelle voci che sono settate nei parametri di MutualExclude vengono disattivate automaticamente. Il listato 5.9 è un esempio che mostra come settare i dati di mutua esclusione per il menu PATTERNFILL. Viene mostrata solo MutualExclude per chiarezza.

Il sistema fa attenzione solo ai bit che si trovano nel range del numero di voci fornite. Per esempio, se vi sono soltanto quattro voci in un determinato menu, è giusto specificare MutualExclude in modo che abbia tutti i bit a uno tranne gli ultimi quattro. Il sistema si occupa solo di tanti bit quante sono le voci con cui operare. Infatti, questo è ciò che si fa qui -si prende un valore che contiene tutti bit posti a uno (un valore con tutti uno lascerà inattive tutte le voci) e si resetta un Bit specifico utilizzando un OR esclusivo. Così, quando viene selezionata una certa voce, tutte le altre vengono disattivate contemporaneamente.

Evidenziare una voce di menu. Quando un utente sta scorrendo lungo un menu con il puntatore del mouse, si potrebbe desiderare che sia chiaro che il sistema sa quale voce sarebbe selezionate se fosse lasciato il pulsante di selezione del mouse. Normalmente, se viene lasciato il pulsante di selezione del mouse mentre il puntatore non è posizionato su nulla che si trovi nell'area del menu, non vi saranno invii di messaggi al task di controllo. Intuition da la possibilità di scegliere dove posizionare una potenziale scelta - utilizzando i flag di Highlighting.

```
/* setexclude.c */

#define EXCLUDEALL 0xffffffff

SetExclude(mi,howmany)
            struct MenuItem *mi;
            int howmany;
      {
            for(i=0; i<4; i++)
            {
                        miMutualExclude =
                                    EXCLUDEALL ^ (1 < i);
            /* resetta a zero un solo Bit specifico; quel Bit */
            /* corrisponde all'identificatore di questa particolare */
            /* voce di menu. */

            mi++; /* punta a quello successivo per settare mut.ex */
            }
}
```

Listato 5.9

Se si specifica HIGHBOX, allora, come nel listato (ITEMSTUFF = MENUENABLED + HIGHBOX), un rettangolo viene disegnato attorno a ognuno dei rettangoli di selezione mentre il mouse si muove.

Se si specifica HIGHCOMP, allora il colore del rettangolo viene cambiato creando il complemento dei colori all'interno del rettangolo di selezione. Si vi è un testo, allora si otterrà l'effetto di Reverse qundo il rettangolo viene selezionato. Si è preferito usare HIGHBOX invece di HIGHCOMP per evitare di cambiare i colori. Dopo tutto, si tratta di un programma di Paint e si desidera proprio scegliere un determinato colore.

Se si specifica HIGHIMAGE, si dice che vi è un puntatore valido all'interno del parametro SelectFill nella struttura dati MenuItem e che questa immagine (o questo testo) alternativa deve essere usata al posto dell'immagine (o del testo) che viene specificata in ItemFill.

Se si specifica HIGHNONE, questo lascia l'utente al buio. Non vi saranno indicazioni da parte del sistema riguardo quale voce sarà selezionata quando l'utente rilascerà il pulsante di menu del mouse. Il proprio task potrebbe ricevere dei messaggi circa selezioni operate involontariamente dall'utente. Questo non è una buona opzione da scegliere.

I requester

Un requester appartiene a una window. Esso viene visualizzato nella window, riferendosi all'angolo superiore sinistro della window stessa. I requester vengono implementati come Layers separati dell'area di visualizzazione. Questo significa che, anche se la window è talmente piccola da non poter contenere il requester, il requester sarà visualizzato nella sua totalità, sovrapponendosi ai bordi della window se risultasse necessario.

Si inizializza una struttura dati requester richiamando InitRequester e puntando a uno scheletro 'vuoto' di requester. Poi si specifica dove deve essere posizionato il suo angolo superiore sinistro (LefteEdge, TopEdge) relativamente all'angolo superiore sinistro della sua window. Si specificano anche la larghezza e l'altezza del requester.

Un requester viene visualizzato riempiendo un rettangolo con un colore specifico (BackFill) e disegnandone i bordi utilizzando segmenti di retta che sono stati definiti (ReqBorder). ReqBorder permette di disegnare un requester in modo

che appaia come qualcosa che sta fluttuando sopra lo sfondo dello schermo, lasciando un'ombra sopra lo schermo stesso. Questo viene chiamato Drop-Shadow. Poi viene visualizzato il testo (IntuiText) che si desidera. Se vi è qualche gadget in un requester (di solito ce n'è almeno uno - OK o CANCEL), si dovrà includere l'indirizzo del primo di essi di una lista linkata di gadget che si trovi nella propria struttura dati requester. Il listato 5.10 è una routine di inizializzazione di un requester.

```
/* inittr.c */

struct Requester textrequest;
struct TextAttr modfontattr;

InitTextRequest()
{
            BYTE *s, *t;
            InitRequest(textrequest);
            textrequest.LeftEdge = 20;
            textrequest.TopEdge =20;
            textrequest.Width = 280;
            textrequest.Height = 130;
            textrequest.ReqGadget = trg[0];
            textrequest.ReqText = textreqtext[0];
            textrequest.BackFill = 1;
            textrequest.ReqBorder = NULL:

            s = textstring[0];/* copia la stringa di Default come testo */
            t = defaulttesxt;
            while((*s++ = *t++) != '\0')
                              /* non fa altro che copiare */

            /* setta il modo di testo iniziale e lo stile */
            txfont = 80;      /* usa un Font a 80 colonne */
            txmode = 1;       /* colore 1 jam */

            modfontattr.ta_Name = "topaz.font";
            modfontattr.ta_YSize = 8;
            modfontattr.ta_Style = 0;
            modfontattr.ta_Flags = 0;

            return (0);
}
```

Listato 5.10

I gadget

Il requester del programma di Paint contiene dei gadget per leggere testi, cancellare un'operazione, e selezionare la posizione di un testo nell'area di disegno. Questo requester contiene tre differenti tipi di gadget: boolean, string e proportional. Essi sono descritti uno per volta. Ma per prima cosa vediamo ciò che è comune a tutti i tipi di gadget.

Posizionamento dei gadget. Se un gadget è stato allegato a una window, allora sarà disegnato facendo riferimento all'angolo superiore sinistro della window. Se lo si allega a un requester, allora sarà disegnato facendo riferimento all'angolo superiore sinistro del requester. Il posizionamento è controllato dai parametri LeftEdge e TopEdge della struttura gadget.

Dimensioni dei gadget. Le dimensioni di un gadget sono definite dai parametri Width e Height della struttura gadget. Questi parametri, come per le voci dei menu, definiscono le dimensioni del rettangolo di selezione. Cioè definiscono l'area all'interno della quale l'utente può clickare con il mouse al fine di selezionare il gadget. Come per le voci di menu, più vicino si trova un gadget all'inizio della lista di gadget, maggiore è la sua priorità nei riguardi della selezione dell'utente. Se due gadget si sovrappongono l'un l'altro, l'area sovrapposta apparterrà a quello che si trova prima nella lista di gadget, cioè esso avrà la priorità in quel luogo.

Aspetto dei gadget. Come per le voci dei menu, i gadget hanno un aspetto quando vengono creati, e un'altro possibile aspetto quando vengono selezionati. GadgetRender e SelectRender corrispondono a ItemFill e SelectFill delle voci di menu. Si può controllare l'evidenziazione dei gadget mediante dei Flag che si trovano tra i parametri della struttura gadget. Come per le voci di menu, i gadget possono essere complementati (GADGHCOMP), racchiusi in rettangoli (GADGHBOX), mostrati con un'immagine alternativa puntata da SelectRender (GADGHIMAGE), o possono non essere evidenziati per niente dalla selezione (GADGHNONE).

Diversamente dalle voci di menu, l'immagine puntata da SelectRender può essere o una struttura dati Image o una struttura dati Border. Si setta un particolare Flag (GADGIMAGE) per dire al sistema che si tratta realmente di un'immagine. Usando i border di solito si risparmia memoria.

Identificazione dei gadget. Quando si usano i menu, in generale, i messaggi che si ricevono dad Intuition vengono formulati dalla struttura dati dei menu stessi. Il controllo che si ha sul numero di codice che si riceve dopo una selezione consiste nel posizionamento della voce di menu all'interno della lista linkata. Il numero di codice dipende strettamente dalla posizione.

Per i gadget, quando si riceve una notificazione di selezione o rilascio, si ottiene l'indirizzo della struttura dati che descrive il gadget. All'interno di questa struttura dati vi è un campo di nome GadgetID al quale si può assegnare il valore che si desidera (soggetto alle mutue esclusioni di cui si necessita). Così, a differenza delle voci di menu che possono essere al massimo 32, si può usare qualsiasi schema si desideri per i GadgetID, così come si è liberi di linkarli all'interno di una lista di gadget in qualsiasi ordine si voglia. Le informazioni riguardanti la selezione di un gadget non sono dipendenti dalla posizione.

Mutue esclusioni. Le mutue esclusioni sono disponibili anche per i gadget. Il parametro MutualExclude viene formato nello stesso modo dell'omonimo parametro delle voci di menu, ma esso dipende dal campo GadgetID della struttura gadget più che dall'ordine nel quale i gadget sono linkati all'interno della lista.

Tipi di gadget. I gadget vengono allegati alle window o ai requester. Nella struttura dati gadget, nel parametro chiamato GadgetType, il Flag di nome REQGADGET, se è settato, dice che questo è un gadget di un requester, altrimenti si tratta di un gadget di window. Il campo GadgetType contiene molti altri flag di definizione del gadget, ma la maggior parte di essi appartengono a Intuition.

Flag di gadget. Oltre ai Flag che specificano l'aspetto del gadget, sono disponibili i seguenti Flag della struttura gadget:

GADGDISABLED	Scurisce il gadget e lo rende inselezionabile.
SELECTED	Visualizza il gadget nel suo stato di selezione. Si può guardare questo flag per sapere se un gadget è correntemente selezionato.
GRELBOTTOM	Interpreta il parametro TopEdge come se fosse Bottom Edge. Questo permette di attaccare un gadget in fondo a una window o a un requester. Se l'utente ridimensiona la window, il gadget si muove con essa lungo la destra, rimanendo sempre invisibile.

GRELRIGHT — Interpreta il parametro LeftEdge come se fosse un RightEdge. Anche esso, permette di lasciare il gadget visibile anche dopo un ridimensionamento della window da parte dell'utente.

Attivazione dei gadget. I parametri di attivazione dei gadget dicono a Intuition cosa fare quando un gadget viene selezionato, mentre l'utente sta ancora tenendo premuto il pulsante. Ecco alcune caratteristiche sull'attivazione:

GADGHIMMEDIATE — Dice a Intuition di inviare un messaggio nel momento in cui l'utente preme il pulsante mentre si trova all'interno del rettangolo del gadget. Si deve anche aver selezionato GADGETDOWN tra i Flag IDCMP per poter ricevere il messaggio.

RELVERIFY — Dice a Intuition di inviare un messaggio GADGETUP quando l'utente rilascia il pulsante di selezione. Ma si riceverà il messaggio solo se il puntatore del mouse si trova ancora sopra il gadget. Questo significa che se si vuole semplicemente ricevere GADGETDOWN e GADGETUP, si deve compiere un po' più di lavoro, poiché non si può controllare dove l'utente posizionerà il puntatore finché non rilascerà il pulsante di selezione. Si noti che GADGETUP deve essere settato tra i flag IDCMP della window per ottenere il messaggio.

FOLLOWMOUSE — Dice a Intuition di fornire informazioni sulla posizione del mouse mentre il gadget è clicato dall'utente. Se GADGIMMEDIATE è stato settato, si sa quando il gadget è stato clickato. Persino se sono stati richiesti i movimenti del mouse settando FOLLOWMOUSE, si deve settare il flag MOVEMOUSE tra i flag IDCMP della window. Se FOLLOWMOUSE è settato, allora il Loop del programma può leggere qualunque evento legato al mouse oltre a MOUSEMOVE, rendendosi conto che il pulsante di selezione è stato finalmente rilasciato.

ENDGADGET — Se un gadget è installato in un requester, quando l'utente lo seleziona, esso automaticamente pone fine al requester, come se fosse stata richiamata EndRequester. ENDGADGET viene usato nel programma di Paint per il pulsante di cancellazione.

FLAG DI BORDO — Vi sono quattro flag riguardanti i bordi tra i Flag Activation: RIGHTBORDER, TOPBORDER, BOTTOMBOR-

DER, e LEFTBORDER. Se viene settato ognuno di questi flag, il corrispondente lato viene posizionato per costruire il corpo del gadget. Questi flag vengono usati per esempio per le barre di scroll. Per una window GIMMEZEROZERO, non si desidera generalmente che gli oggetti grafici si sovrappongano alla barra di scroll o ad altri gadget. Questi flag chiedono al sistema che vi sia uno spazio protetto entro il quale disegnare i gadget.

Il listato 5.11 è un'insieme di istruzioni che definisce tutti i gadget del requester di testo.

Gadget booleani. Il primo gadget del listato 5.11 è un gadget booleano. Un gadget booleano ha solo due stati, o è selezionato o non lo è. Per il gadget Cancel del requester, i flag specificano GADGHCOMP, che significa "sostituisci ai colori i loro complementari quando il gadget viene selezionato".

I Flag Activation contengono GADGIMMEDIATE | ENDGADGET; si tratta della combinazione OR di entrambi i flag. Quando viene premuto Cancel, la sola cosa che il sistema deve sapere è ENDGADGET. ENDGADGET fa si che il requester scompaia automaticamente, così l'informazione circa GADGETDOWN generata da GADGIMMEDIATE è solo di convenienza. Più di un gadget in un requester può avere il Flag ENDGADGET settato. Se si desidera che il proprio programma sappia quale gadget ha fatto si che il requester svanisse, si setti sia GADGIMMEDIATE sia ENDGADGET. Altrimenti, Intuition non genererà alcun messaggio e non si sarà in grado di dire quale gadget ha terminato il requester.

Il parametro GadgetType contiene REQGADGET | BOOLGADGET: questo è un gadget booleano istallato in un requester. Non vi è alcuna descrizione dei bordi né visualizzazione speciale quando il gadget viene selezionato. Se fosse stato necessario visualizzare un'immagine alternativa, si sarebbe dovuto settare il Flag GADGHIMAGE.

Il gadget Text punta all'IntuiText che contiene la parola Cancel. Le mutue esclusioni non vengono usate. Vi sono informazioni particolari riguardanti i gadget di stringa e quelli proporzionali e saranno trattate più avanti in questo paragrafo.

```
/* trgadgets.c */

struct Gadget trg[= {                    /* CANCEL */
{ &trg[1],                          /* indirizzo del Gadget successivo */
205,115,60,9,                       /* dimensioni del rettangolo di */
                                    /* selezione */
GADGHCOMP,                          /* Flag */
GADGIMMEDIATE | BOOLGADGET,
                                    /* ci informa solo quando l'utente */
                                    /* rilascia il pulsante del mouse */
                                    /* e solo se è sopra il Gadget */
REQGADGET | BOOLGADGET,             /* Requester, Booleano */
NULL,                               /* descrittore del BORDER */
NULL,                               /* descrittore di SELECT */
&textreqtext[11],                   /* Cancel */
0,                                  /* mutue esclusioni */
NULL,                               /* info speciali*/
TEXTWRITEGADGETS + 1,               /* identificatore del Gadget, utente */
NULL},                              /* puntatore dati dell'utente */

{ &trg[2],                          /* modo testo */
190,3,40,9,
GADGHCOMP,
RELVERIFY | GADGIMMEDIATE,
REQGADGET | BOOLGADGET,
NULL,
NULL,
textreqtext[],
0,
NULL,
TEXTWRITEGADGETS + 2,
NULL},
{ &trg[3],                          /* stile del testo */
190,13,80,9,
GADHCOMP,
RELVERIFY | GADGIMMEDIATE,
REQGADGET | BOOLGADGET,
NULL,
NULL,
textreqtext[9],
0,
NULL,
TEXTWRITEGADGET + 3,
NULL},
 &trg[4]                            /* questo è un Gadget String */
55,60,140,10,                       /* posizione e dimensioni del */
                                    /* rettangolo di selezione */
GADGHCOMP,                          /* Flag, modo complementare */
RELVERIFY | ENDGADGET,              /* Flag Activation, quando l'utente */
                                    /* attiva Return, termina l'Input */
                                    /* (avviene anche un RELVERIFY) e */
```

```
                                /* disseleziona il Gadget */
REQGADGET | ENDGADGET,          /* è un Requester, String */
NULL,                           /* descrittore BORDER */
NULL,                           /* descrittore SELECT */
NULL,                           /* IntuiText da scrivere qui */
0,                              /* mutue esclusioni */
(APTR)&textstringstuff,         /* info speciali */
NULL},                          /* puntatore dati dell'utente */

                                /* un gadget Dual-Proportional */
{NULL,                          /* gadget successivo in questa lista = nessuno*/
190,25,                         /* estremità sin., estremità sup.*/
                                /* del rettangolo di selezione */
80,42,                          /* dimensioni del rettangolo */
GADGIMAGE | GADGHNONE,          /* Flag */
GADGIMMEDIATE | RELVERIFY,      /* Flag di attivazione */
PROPGADGET | REQGADGET,         /* è un gadget proporzionale */
(APTR)&textimage,               /* visualizzazione normale con questa */
                                /* immagine */
(APTR)&textimmage,              /* visualizzazione evidenziata con */
                                /* questa immagine */
NULL,                           /* nessun testo per questo Gadget */
0x0,                            /* mutue esclusioni */
TEXTWRITEGADGET +4,             /* questo è l'identificatore del */
                                /* Gadget per me */
NULL } };
```

Listato 5.11

Tutti i GadgetID sono basati su di un valore preassegnato chiamato TEXTWRITEGADGETS. Se si modifica questo programma per molti gadget, esso rappresenta un modo facile per identificare univocamente un gruppo di gadget.

Il secondo e il terzo gadget definiti nel listato servono a cambiare lo stile e a visualizzare il testo. Per lo stile, si può utilizzare sia JAM1 sia JAM2. I due font residenti in ROM, topaz-60 e topaz-80, sono le possibili scelte per visualizzare il testo. Anche questi sono gadget booleani. Si possono, naturalmente, installare altre scelte utilizzando le informazioni sui testi che si trovano nel quarto capitolo.

La differenza tra questi e il gadget Cancel sta nel fatto che questi non sono ENDGADGETS. Il requester rimane presente. Ciò che bisogna fare per elaborare questi gadget è fornire un testo alternativo per essi e cambiare la variabile Flag di conseguenza, in modo da scegliere quale testo si dovrà usare.

Gadget String. Il quarto gadget del listato 5.11 è un gadget String. Esso è utilizzato per ottenere informazioni dall'utente. Il rettangolo di selezione è dimensionato in modo da mostrare più caratteri possibili di una stringa. Si sceglie di accettare una stringa solo se è lunga quanto il rettangolo, ma si può selezionare un rettangolo stretto e una stringa molto lunga.

Questo gadget ha come parametro flag GADGHCOMP, il quale è richiesto per i gadget String. Il Flag Activation ha settato ENDGADGET in modo tale che il requester se ne vada automaticamente nel caso l'utente selezioni il gadget di stringa, e poi prema Return. Premere Return diventa un segnale di accettare il risultato a prescindere di ciò che è successo all'interno del gadget String (persino se non è stato scritto nulla).

Nel GadgetType, sono settati REQGADGET | STRGADGET. Essendo settato STRGADGET, Intuition sa come interpretare il puntatore SpecialInfo all'interno delle struttura gadget. Specificamente, Intuition sa che questo è un puntatore a una struttura dati StringInfo mostrata nel listato 5.12.

La struttura GadgetInfo fornisce informazioni, tra l'altro, sulla struttura gadget stessa, compreso un parametro textstring che dice dove posizionare il testo che l'utente ha inserito in questo Gadget String, e un parametro textundo che fornisce un buffer Undo. Nell'usare il gadget l'utente può servirsi di un UNDO per ripristinare il testo di default o il testo precedentemente inserito (schiacciando Return).

I gadget String operano automaticamente uno scroll per permettere la visualizzazione di tutti i caratteri permessi dalla grandezza del rettangolo di contenimento. Quando vengono disegnati, i gadget String contengono un cursore che è chiaramente visibile quando il gadget è attivo. Per controllare l'aspetto iniziale del gadget, la struttura:

```
/* stringinfostuff.c */

UBYTE textstring[10];
UBYTE textundo[10];
UBYTE *defaulttext = "test";

struct StringInfo textstringstuff = {
&textstring[0],              /* stringa di Default e finale */
&textundo[0],                /* buffer opzionale Undo */
90,                          /* posizione del carattere nel Buffer */
10,                          /* numero max di caratteri nel Buffer */
0,                           /* posizione nel Buffer del primo */
                             /* carattere visualizzato */
0,0,0,0,0,NULL,0 };          /* variabili locali di Intuition */
```

Listato 5.12

StringInfo definisce anche:

- La posizione alla quale deve trovarsi il cursore all'interno del Buffer quando viene selezionato il gadget per la prima volta (0).
- Il massimo numero di caratteri che può accettare il gadget (10).
- La posizione all'interno del Buffer del primo carattere da visualizzare (0).

Le variabili usate da Intuition per tenere traccia di ciò che sta facendo l'utente con questo gadget sono state tutte settate a zero.

Le variabili globali che sono mostrate come parte del listato 5.12 rappresentano lo spazio per la stringa di testo e per il buffer di Undo. Il testo di default viene copiato all'interno dell'area della stringa prima che il requester venga presentato per la prima volta. Così, se l'utente seleziona il gadget e preme Return. viene usata la stringa di default.

Gadget Proportional. Per illustrare i gadget proporzionali, è stato fornito un gadget che contiene l'immagine di una punta di freccia, chiamata slider, che può muoversi sia orizzontalmente che verticalmente. Per specificare completamente i

gadget proporzionali, oltre alla struttura gadget stessa sono necessarie tre struttura dati addizionali:

- Una struttura dati PropInfo (chiamata textslider nel listato 5.1) che descrive sia gli allegati del gadget proporzionale sia il comportamento del gadget, e anche la sua posizione iniziale.
- Una struttura dati Image (chiamata textimage nel listato 5.1) che descrive le dimensioni dello slider e dove possono essere trovati i dati che lo riguardano. Questi dati devono trovarsi nella memoria accessibile ai Chip Custom (MEMF_CHIP). Se si usa semplicemente la struttura dati definita da textsliderimage, lo slider non sarà visibile se si fa girare il programma su di una macchina che possiede espansioni di memoria.
- Un Array di uword senza segno che contiene l'immagine stessa. Il puntatore chiamato clipsliderimage, all'interno del programma attuale, è settato in modo da puntare a una zona di memoria accessibile ai Chip Custom. Lo slider viene poi copiato all'interno di questa memoria per un uso futuro.

Il listato 5.1 contiene le strutture dati e le funzioni che definiscono lo slider che si usa per posizionare il testo.

```
/* slider.c */

      /* questa immagine contiene la figura di un diamante */
      /* la useremo però solo come una punta di freccia */
      /* specificando l'uso soltanto dei 4 Bit a sinistra */
      /* dell'immagine */

      UWORD textsliderimage[] = {
                        0x03c0,
                        0x0ff0,
                        0x3ffc,
                        0xffff,
                        0x3ffc,
                        0x0ff0,
                        0x03c0 },
      UWORD *chipsliderimage  /* l'immagine dello slider deve */
                              /* trovarsi nella memoria */
                              /* accessibile ai Chip Custom */
                              /* il programma alloca tale */
                              /* memoria e poi vi copia */
                              /* lo sider */
      struct Image textimage = {
                        /* l'immagine è larga 16 Bit ma ne */
```

```
                        /* usiamo solo 8 per far si che la freccia */
                        /* dica dove posizionare il testo */
            0,0,8,7,1,&textsliderimage[0],0x1,0 };

      struct PropInfo textslider = {
            FREEHORIZ | FREEVERT,
                        /* Flag...può muoversi liberamente */
                        /* in entrambe le direzioni */
            0,0         /* HorizPot, VertPot...settano */
                        /* le posizioni iniziali */
                        /* orizzontale e verticale */
                        /* e poi legge queste variabili */
                        /* mentre l'utente sta operando *
                        /* per vedere dove si trova il */
                        /* posizionato correntemente */
                        /* il Knob. */
            0xffff,     /* HorizBody..non si usa l'autoKnob */
                        /* così può non essere necessario */
                        /* settare a un valore non nullo */
            0xffff,     /* VertBody */
            0,0,0,0,0,0, };   /* variabili di Intuition */

InitChipSliderImage()
{
      int i;
      UWORD *s,*d;

      chipsliderimage = (UWORD*)AllocMem(14, MEMF_CHIP);
      if(chipsliderimage == NULL)
      {
            return(FALSE);
      }
      s = Textsliderimage;    /* dati Static del programma */
      d = chipsliderimage;    /* un'area sicuramente */
                              /* accessibile ai Chip Custom */
      for(i=0; i<7; i++)
      {
            *d++ = *s++       /* copia di dati */
      }
      return(TRUE);
}

DeleteChipSliderImage()
{
      FreeMem(chipsliderimage, 14);
}
```

Listato 5.13

Il parametro dei flag all'interno della struttura PropInfo è settato in modo da permettere sia movimenti orizzontali sia movimenti verticali del Knob di controllo (Slider). I valori iniziali di HorizPot e VertPot sono settati a 0,0 in modo da piaz-

zare lo slider all'angolo superiore sinistro del contenitore. Questi valori cambieranno quando l'utente sposterà lo Slider verticalmente o orizzontalmente. Qui si stanno usando delle strutture dati static, con valori iniziali determinati all'atto della compilazione. Se il requester viene richiamato più volte e l'utente muove lo slider di controllo, allora ogniqualvolta compare il requester, la posizione dello slider rispecchia la posizione settata dall'utente l'ultima volta.

Quando l'utente sposta lo slider, i valori di HorizPot e VertPot varieranno da 0 a FFFF (Hex). Il valore di tali variazioni dipende dalla grandezza del rettangolo all'interno del quale si muove lo slider. Per esempio, se si permettono 20 linee di movimento verticale, cioè 20 possibili posizioni verticali alle quali l'utente può posizionare lo slider, allora ci sono 20 possibili valori per VertPos. Ogni posizione adiacente corrisponderà a un valore che dista circa FFFF (hex) diviso 20 (decimale).Per interpretare i valori orizzontali e verticali, si usi la seguente formula:

```
myrange = mymaxvalue-myminvalue

   /* determina il Range dei valori attuali ammessi */

myactualvalue = myminvalue +
   ((ULONG)myrange * (ULONG)textslider.VertPot)
   /0xFFFF;

         /* sopra si è usata un poco di aritmetica */
         /* degli interi, creando una frazione */
         /* per la quale viene moltiplicato myrange */
         /* (la moltiplicazione precede */
         /* la divisione per */
         /* mantenere la maggior precisione possibile) */
```

Intuition fornisce anche una possibilità di autoKnob, che qui non è usata, che permette di creare automaticamente l'immagine dello slider. Le dimensioni del Knob generato automaticamente rappresentano la proporzione della piccola immagine che viene usata qui. Ad esempio, un gadget di scroll allegato a una window potrebbe avere un autoKnob le cui dimensioni sono uguali a quelle del contenitore totale se tutto il contenuto della window è visibile, o solo a una parte del contenitore se si vede solo una parte del contenuto della window. In questo caso, si setterà il flag AUTOKNOB nella struttura PropInfo, e non si punterà ad alcuna immagine.

Se si è interessati ai rimanenti parametri della struttura PropInfo, si veda la spiegazione nell'Amiga Intuition Manual. Il resto è soprattutto riservato ad un uso

interno di Intuition e non è indispensabile conoscerlo per usare un gadget proportional.

Elaborazione dei menu

Per elaborare una selezione di menu (MENUPICK), si usano le macro di sistema MENUNUM, ITEMNUM, e SUBNUM. Esse trasformano il codice contenuto in un IntuiMessage in valori numerici riguardanti il menu nel quale è avvenuta la selezione (i valori vanno da 0 a 30), il numero della voce di menu all'interno del menu (da 0 a 63), e la sottovoce, se c'è, che è allegata alla voce di menu.

Può avvenire che un utente guardi un menu ma non effettui alcuna selezione. Questo viene tradotto in CODE MENUNULL e CLASS MENUPICK. Si può sapere cosa è accaduto interpretando il menu, le voci del menu, e le sottovoci del menu individualmente. In questo caso, MENUNUM (codice) ritorna NOMENU, ITEMNUM (codice) ritorna ITEMENU, e SUBMENU (codice) ritorna NOSUB. Il giusto modo di operare sta nell'elaborare tutti i segnali ammessi che si possono riconoscere e come default ignorare totalmente gli altri.

Pertanto, i limiti imposti da Intuition per le operazioni sui menu sono i seguenti:

- 31 possibili selezioni, al massimo, all'interno della striscia dei menu.
- 63 possibili selezioni di immagini o di testi all'interno del gruppo delle voci di menu allegate a ogni singola selezione.
- 31 selezioni di immagini o di testi all'interno del gruppo delle sottovoci allegate a ogni voce del menu.

Si vede quindi che vi sono tante possibilità.

Il listato 5.14 fornisce una routine che elabora l'IntuiMessage contenente un evento MENUPICK. Richiamatala, il menu Color permette di prelevare un colore da usare per disegnare, e il menu Text apre un requester che permette di specificare a quale posizione si desidera piazzare il testo. L'elaborazione dei menu ri-

sulta molto semplice se si elaborano i valori riconosciuti legali dell'IntuiMessage e si ignora ogni altra cosa.

Elaborazione di eventi legati ai gadget

La routine del listato 5.15 è leggermente complicata, poiché gestisce dei gadget proporzionali, dei gadget string, e dei gadget booleani, ma un'istruzione di Switch in questi casi si rivela un trucco brillante.

Il listato 5.16 comprende tre routine che di fatto cambiano il modo in cui viene visualizzato il testo - modo di testo, stile, e scrittura del testo (visualizzazione). Le prime due routine sono relativamente semplici. La terza contiene delle istruzioni che interpretano il valore attuale dei gadget proporzionali per determinare dove posizionare il testo all'interno dell'area di disegno. Si noterà che, in textstyle, il sistema va direttamente a guardare i valori dei puntatori di identificazione testo. Questo viene fatto in luogo di una funzione strcmp. Se i puntatori hanno lo stesso valore, entrambi stanno puntando alla stessa stringa. L'IntuiText per il requester si trova in questo parte del programma, così come le routine di elaborazione.

Il programma di Paint

Infine, ecco nel listato 5.17 il corpo principale del programma di Paint che raccoglie tutto quanto abbiamo fatto nel paragrafo precedente.

```
/* menupick.c */

#define COLORMENU 0
#define TEXTMENU 1

#define FIRSTITEM 0

MenuPick(im)
      struct IntuiMEssage *im;
{
      USHORT code, k;
```

```
code = im->Code;

switch(MENUNUM(code) {

            case COLORMENU:
                  /* setta un nuovo colore di penna */
            k = ITEMNUM(code);

            if(K >= 0 && k <= 15)/* è nel Range? */
            {
                  SetPen(rp, ITEMNUM(code));
                  SetDrMd(rp, !JAM);
            }
            break;

            case TEXTMENU:    /* apre il requester */
                              /* nella window */

            Request(&textreq,w);
                  break;

            default:
                  break;
            }
            return(TRUE);
}
```

Listato 5.14

```
/* gadgetup.c */

#define GADGETID ((struct Gadget *)IAdress)->GadgetID
GadgetUp(ms)
            struct IntuiMessage *ms;
{
            SHORT id;
            struct Gadget *g;

            g = (struct Gadget *)(ms->IAdress);
            id = g->GadgetID;

            /* che numero di Gadget era? */

            switch(id) {

            case TEXTWRITEGADGETS:
```

```
                textwrite();      /* scrive il testo */
                break;

        case TEXTWROTEGADGETS + 1:

                break;      /* Cancella il Gadget, in modo che */
                            /* non vi sia bisogno di alcuna */
                            /* azione. Poi Cancel contiene un */
                            /* ENDGADGET che si sbarazza del */
                            /* Requester */

        case TEXTWRITEGADGETS + 4:

                break;      /* eventi legati ai Gadget */
                            /* proporzionali. Non ci si occupi */
                            /* di ciò che accade al Gadget prop. */
                            /* finché l'utente non vuole */
                            /* realmente scrivere un testo */
                            /* ALLORA lo si interpreti */

        case TEXTWRITEGADGETS + 2:

                textstyle();      /* topaz-60, topaz-80 */
                break;

        case TEXTWRITEGADGETS + 3:

                textmode();       /* JAM1, JAM2 */
                break;

        default:
                break;
        }
return(TRUE);
}
```

Listato 5.15

```
/* textstuff.c */

int txfont, txmode;          /* var. Global per il controllo del testo */

/* seguono delle dichiarazioni che renderanno felice il compilatore */

extern struct Window *w;
extern struct TextAttr TestFont;
extern struct PropInfo textslider;
extern struct TextAttr modfontattr;
```

```
extern struct Gadget trg[];
extern struct Requester textrequest;

struct IntuiText textrequest[] = {

            /* requester IntuiText per questo Requester colorato */

            {0,1,JAM1, 5,3, &TestFont, "Click per cambiare modo:",
                        &textrequest[1]},
            {0,1,JAM1, 5,13, &TestFont, "Click per cambiare stile:",
                        &textrequest[2]},
            {0,1,JAM1, 5,3, &TestFont, "Posiziona Text",
                        &textrequest[3]},
            {0,1,JAM1, 5,80, &TestFont,
                  "Click nel rettangolo di Text per scrivere:",
                        &textrequest[4]},
            {0,1,JAM1, 25,90, &TestFont,
                  "Entra nell'area di disegno:",
                        &textrequest[5]},
            {0,1,JAM1, 5,105, &TestFont, "Return per disegnare Text:",
                        &textrequest[6]},
            {0,1,JAM1, 5,115, &TestFont, "Click su CANCEL per uscire:",
                        NULL},

            /* il Gadget di Text non è linkato nell'IntuiText sopra */
            /* mostrato... è solo un posto comodo dove */
            /* immagazzinarlo */

            /* Gadget IntuiText per questo Requester a colori */

            { 1,0,JAM2,1,0,&TextFont,"JAM1",
                        NULL},

            { 1,0,JAM2,1,0,&TextFont,"JAM2",
                        NULL},

            { 1,0,JAM2,1,0,&TextFont,"Topaz-80",
                        NULL},

            { 1,0,JAM2,1,0,&TextFont,"Topaz-60",
                        NULL},

      { 1,0,JAM2,1,0,&TextFont,"CANCEL",
                        NULL},
            };

textstyle{}
{
            if( trg[1].GadgetText == &textreqtext[7])
            {
                        trg[1].GadgetText = textreqtext[8];
                        txmode = 2;
            }
            else
            {           if( trg[1].GadgetText == &textreqtext[8])
```

```
                    {
                              trg[1].GadgetText = textreqtext[7];
                              txmode = 1;
                    }
          }
          /* ne abbiamo cambiato uno in modo da operare un */
          /* Refresh su tutti loro */
          RefreshGadgets(&trg[1],w,&textrequext);
}

textmode()
{
          if( trg[2].GadgetText == &textreqtext[9])
          {
                    trg[2].GadgetText = textreqtext[10];
                    txfont =60;
          }
          else
          {         if( trg[1].GadgetText == &textreqtext[10])
                    {
                              trg[2].GadgetText = textreqtext[9];
                              txfont = 80;
                    }
          }
          RefreshGadgets(trg[1],w,textrequext);
}
textwrite()
{
          ULONG temp1, temp2;

          struct TextFont *oldfontsave;
          struct TextFont *myfontptr;

          /*mette in scala la posizione secondo la dimensione */
          /* attuale dello schermo, si poteva fare per una */
          /* window e/o altro */

          /* usa gli 8 Bit alti di ogni valore... c'è abbastanza */
          /* risoluzione */

          temp1 = ((textslider.HorizPot >> 8) * (WWIDTH-1)) >> 8;
          temp2 = ((textslider.VertPot >> 8) * (WHEIGHT-1)) >> 8;

          /* sopra converte la posizione dello slider in un valore */
          /* che sia all'interno del Range della window */
          Move(rp,temp1,temp2);

          if(txmode == 1)
                SetDrMd(rp,JAM1);
          else
                SetDrMd(rp,JAM2);

          if(txfont == 80)
                modfontattr.ta_YSize = 8;
          else
```

```
            modfontattr.ta_YSize = 9;

        /* salva il Font corrente di Intuition */
        oldfontsave = rp->Font;

        /* seleziona il font desiderato dall'utente */
        myfontptr = (struct TextFont *)OpenFont(&modfontattr);

        if(myfontptr == 0)
        {
              printf("\nil Font non può essere aperto");
              /* non disegna il testo se non trova il Font */
              return(0);
        }
        SetFont(rp,myfontptr);

        /* non disegna il testo se il Font non va bene */

        Text(rp,&textstring[0],strlen(textstring));

        /* ripristina il vecchio Font*/
        SetFont(rp,oldfontsave);

        /* chiude il nuovo Font */
        CloseFont(myfontptr);
        return(0);
}
```

Listato 5.16

```
#define DEPTH 4
#define WWIDTH 320
#define WHEIGHT 190

#include "exec/types.h"
#include "Intuition/Intuition.h"
#include "exec/memory.h"
#include "imageedit.h"

#include "myscreen2.h"
#include "event2.c"
#include "stubs1.c"
#include "ticks.c"
#include "mousebuttons.c"
#include "stringinfostuff.c"
#include "slider.c"
#include "textstuff.c"
#include "trgadgets.c"
#include "inittr.c"
```

```
struct Window *w;
struct RastPort *rp;
struct ViewPort *vp;
struct Screen *screen;
struct Image colorimage[32];  /* fornisce il numero max */

long IntuitionBase=0;
struct MenuItem coloritem[32];
      /* solo ne caso sia richiesta una profondità di 5 piani */

long GfxBase=0;
struct Menu menu[2];
      /* è presente una voce principale */

extern struct Screen *OpenScreen();
extern struct Window *OpenWindow();

#define ITEMSTUFF (ITEMENABLED | HIGHBOX)
#define CW 40     /* dimensioni del singolo rettangolo nella */
                  /* tavolozza di scelta dei colori */
#define CH 25

SHORT palette[] = { 2, 4, 8, 16, 32, 64 };

#include "initmenu.c"
#include "menupick.c"
#include "gadgetup.c"

main()
{
      struct IntuiMessage *mess;
      int havevalidimage;

      GfxBase = OpenLibrary("graphics.library", 0);
      if (GfxBase == NULL)
      {
            printf("Non riesco ad aprire graphics.library\n");
            exit(1000);
      }
      IntuitionBase = OpenLibrary("Intuition.library", 0);
      if (IntuitionBase == NULL)
      {
            printf("Non riesco ad aprire Intuition.library\n");
            exit(1000);
      }
      screen = OpenScreen(&ns);
      if (screen == NULL)
      {
            exit(1);
      }
nw.Screen = screen;

      w = OpenWindow(&nw);    /* apre una Window */
      rp = w->RPort;
      vp = &w->WScreen->ViewPort;
```

```
        InitMenu();

        SetMenuStrip(w, menu);

        InitTextRequest();

        havevalidimage = FALSE;

        if(InitChipSliderImage())
        {
              textimage.ImageData = (USHORT *)chipsliderimage;
              havevalidimage = TRUE;
        }
        /* deve fare un InitChipSliderImage nel caso il sistema */
        /* usato per far girare il programma abbia più di 512K */
        /* L'immagine dello Slider deve trovarsi nella memoria */
         /* accessibile ai Chip Custom */
        while(1)
        /* "per sempre" ... attende che finisca il messaggio */
        {
              /* è possibile che l'Intution invii più di un */
              /* messaggio mentre il task è inattivo */
              /* si deve svuotare la porta prima di porsi */
              /* nuovamente in stato di inattività. Ciò che */
              /* segue gestisce ogni messaggio ricevuto e si pone */
              /* in stato di inattività solo quando la porta è */
              /* stata svuotata */

              mess = (struct IntuiMessage *)GetMsg(w->UserPort);

              if(mess == NULL)
                    WaitPort(w->UserPort);
              else
                    if(HandleEvent(mess) == FALSE)
                          break;
        }
        /* cancella la striscia, e poi chiude ciò che */
        /* è stato aperto */

        if(havevalidimage)
        {
              DeleteChipSliderImage();
        }

        ClearMenuStrip(w);
        CloseWindow(w);
        CloseScreen(screen);
        CloseLibrary(GfxBase);
        CloseLibrary(IntuitionBase);

}       /* fine del main() */
```

Listato 5.17

Ulteriori opzioni

Il programma di Paint che è stato costruito passo per passo in questo capitolo fornisce molti Tool utili che possono essere usati in tutti i programmi in Intuition. Naturalmente, esso non comprende tutte le opzioni disponibili. Ci sono un paio di opzioni qui accluse che possono fornire alcune buone idee circa le cose che si possono fare mediante Intuition per ottenere ancora una maggior versatilità.

Immagini e testi combinati

L'Intuition permette di settare il flag GADGIMAGE per i gadget per indicare che GadgetRender e SelectRender puntano a un'immagine e non a un testo (IntuiText). Allo stesso modo Intuition permette di settare il flag ITEMTEXT per le voci dei menu per indicare che i puntatori ItemFill e SelectFill puntano a un IntuiText e non a un'immagine. Ma cosa si deve fare se si vuole che la propria voce di menu o il gadget contengano una combinazione di immagini e testi?

Si può creare questo effetto semplicemente combinando due voci o due gadget, uno contenente un IntuiText, l'altro contenente i dati di un'immagine.

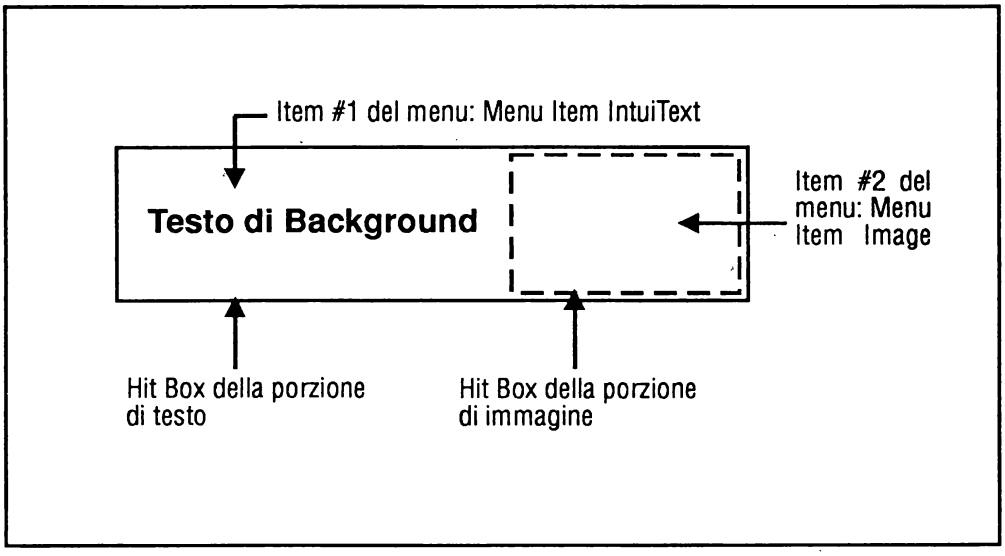


Figura 5.4

Si faccia in modo che i rettangoli di selezione siano dimensionati in modo tale che uno comprenda totalmente l'altro. Si faccia in modo che sia più esterno il rettangolo appartenente alla voce di menu o al gadget che sono più vicini all'inizio della lista linkata dei gadget o delle voci di menu. La Figura 5.4 mostra un'immagine a colori inserita all'interno di una voce di menu (IntuiText).

La voce 1 setta il Flag ITEMTEXT ed è molto vicino all'inizio della lista di voci. Essa costituisce il testo di fondo della voce di totale. La voce 2 non setta il Flag ITEMTEXT ed è più lontana dall'inizio della lista di voci di quanto non lo sia la 1. Poiché il rettangolo di selezione della voce 1 circonda totalmente la voce 2, essa non potrà mai essere selezionata direttamente. La voce totale, nonostante sia composta in realtà dall'unione di due voci distinte, appare come un'unica voce di menu, persino quando viene selezionata.

Liste delle voci di menu

L'Intuition non restringe il campo d'uso della lista delle voci di menu. Una volta che la lista è stata costruita, oltre alla possibilità di poterla linkare a un menu, da anche quella di poterla linkare a una o più voci di menu creando un sottomenu. Per esempio, nel programma di Paint, si poteva costruire il primo menu nel seguente modo:

```
Color

   Penna di disegno

   Testo in primo piano
   Testo sullo sfondo
```

Poi si poteva linkare il pannello delle immagini Color come un sottomenu all'interno delle tre principali voci del menu. Ecco il linkaggio in pseudocodifica:

```
<color>.FirstItem = <penna.di.disegno>

<penna.di.disegno>.NextItem = <text.foreground>
<penna.di.disegno>.SubItem = <primo.item.nella.colorimage.list>

<testo.in.primo.piano>.NextItem = <text.background>
<testo.sullo.sfondo>.SubItem = <primo.item.nella.colorimage.list>
```

```
<testo.in.primo.piano>.NextItem = NULL;
<testo.sullo.sfondo>.SubItem = <primo.item.nella.colorimage.list>
```

Poi ognuna delle voci del menu gestirà uno stesso set di pannelli di colori dai quali si potranno scegliere i colori stessi.

Se si usa questa tecnica, si può anche considerare una combinazione mediante la tecnica precedente che dava l'apparenza del testo misto a immagini così come si può mostrare il colore correntemente scelto per quella voce. Per fare ciò, si tralasci il Flag CHECKIT, poiché la sottovoce sarà spartita da tre differenti voci del menu. La figura 5.5 mostra come apparirà il tutto quando la terza voce verrà selezionata e aprirà la sua sottovoce.

Si noti che poiché la sottovoce comincia in una posizione data relativamente alla voce menu alla quale è allegata, il gruppo di rettangoli colorati comincerà esattamente nella stessa posizione in rapporto a ognuna delle voci di menu.

Si noti anche che si può specificare un valore negativo per LeftEdge e TopEdge per la voce di menu e anche in questo caso avere l'oggetto posizionato correttamente. La figura 5.6 mostra come funziona questa cosa.

Questo capitolo ha mostrato le caratteristiche principali di un'interfaccia utente che bisogna sapere per poter creare programmi in Intuition: gli schermi, le window, i gadget, i requester, e i menu. Ecco alcune note che riassumono quanto esposto:

- Gli schermi permettono di definire la risoluzione e il numero di colori all'interno dell'area di disegno.
- Le window permettono di suddividere gli schermi in una serie di aree sovrapponibili.
- I requester, i gadget, e i menu permettono al programma di ottenere informazioni su ciò che desidera l'utente.

Se si desiderano ulteriori informazioni su uno degli argomenti trattati in questo capitolo, si suggerisce di consultare approfonditamente l'Amiga Intuition Manual, che descrive tutte le strutture dati e le routine in dettaglio.

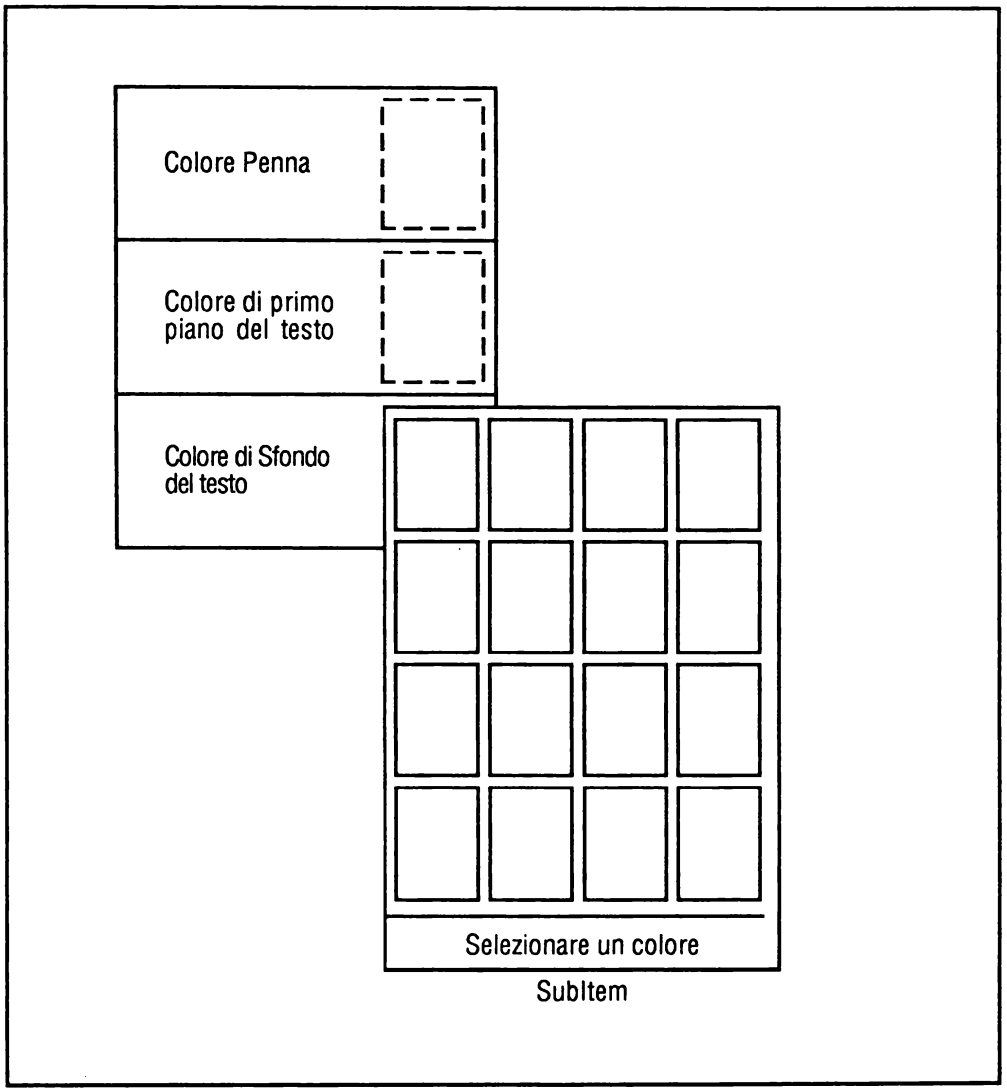

Colore Penna
Colore di primo piano del testo
Colore di Sfondo del testo
Selezionare un colore
SubItem

Figura 5.5

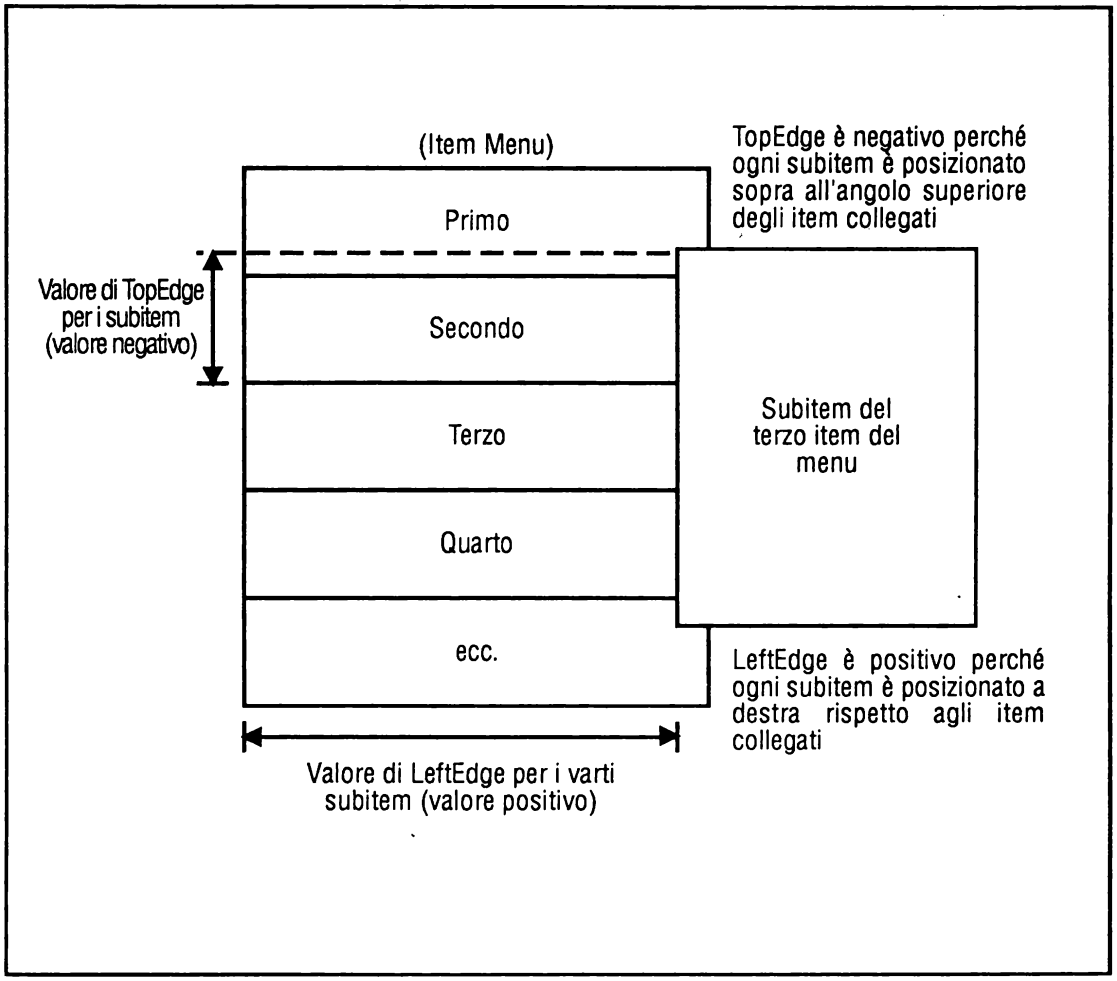

(Item Menu)
TopEdge è negativo perché ogni subitem è posizionato sopra all'angolo superiore degli item collegati
Primo
Valore di TopEdge per i subitem (valore negativo)
Secondo
Terzo
Subitem del terzo item del menu
Quarto
ecc.
LeftEdge è positivo perché ogni subitem è posizionato a destra rispetto agli item collegati
Valore di LeftEdge per i varti subitem (valore positivo)

Figura 5.6

Capitolo 6

I device

Nel terzo capitolo si è detto che l'Exec fornisce un protocollo standard per l'Input-Output. L'Input-Output assume la forma di una richiesta (un messaggio) che viene passata da un programma (task) a un driver di un apparecchio esterno. Tale driver si occupa dell'interazione a basso livello, che di fatto realizza l'Input-Output. Questo significa che un programma può utilizzare un codice ad alto livello per comunicare con l'esterno invece di preoccuparsi di controllare ciò che avviene a livello hardware.

Bisogna fare una nota cautelativa. Una volta che è stata formulata una richiesta di Input-Output per un device e tale richiesta è stata inviata, il blocco di memoria che è stato passato al device non appartiene più al task che lo ha inviato. Non si tenti di leggere o scrivere sulla memoria che costituisce l'IORequest finché non è stata completata la richiesta. Molti device usano le strutture dati contenute nell'IORequest per un proprio uso locale. Si potrebbe bloccare il sistema tentando di leggere o di scrivere su strutture dati che non si trovano sotto il controllo esclusivo del task del programmatore.

Questo capitolo tratta della comunicazione con il device Timer, Console, Input, Keyboard, e Gameport. I device Serial e Printer sono tra quelli che qui non sono trattati. Per informazioni su tali argomenti, si veda l'Amiga ROM Kernel Manual. Si noti che con la versione 1.2 del sistema operativo, il programma Preferences può essere usato per operare i cambiamenti più importanti alle caratteristiche del Serial device. Così, è diventato inutile per un programma cambiare direttamente tali caratteristiche.

Se è stato fatto girare il programma di Paint del quinto capitolo, sarà stato notato che la velocità di tracciamento era piuttosto bassa. Infatti, Amiga è in grado di operare molto più velocemente come potete aver notato. Il tracciamento lento deriva dalla scelta operata su quando disegnare un nuovo pixel. Si ricorderà che il pixel veniva disegnato con il colore desiderato quando veniva premuto il pulsante di selezione del mouse e viene quindi ricevuto il messaggio di INTUITICK. Poiché tale messaggio arriva solo una volta ogni decimo di secondo, la velocità di tracciamento potrà essere al massimo di dieci pixel al secondo.

Vi sono molti modi per aumentare la velocità di tracciamento. Uno consiste nel settare i Flag REPORTMOUSE e MOUSEMOVE nella struttura dati NewWindow. Quando il pulsante di selezione è premuto, il programma in corrispondenza dell'arrivo dell'informazione circa le coordinate del mouse disegna un pixel. Un altro modo consiste nell'utilizzare un timer più veloce.

Il device Timer

Vi sono due unità all'interno del device Timer. Una governa un Timer molto accurato, che misura intervalli di 16.67 millisecondi, con la quale è possibile ottenere sino a 60 eventi al secondo. Questo è molto meglio del INTUITICKS, ma l'INTUITICKS esiste per una certa convenienza d'uso, non per la sua velocità. Questo Timer di precisione è detto VBLANK.

L'altra unità è chiamata Timer MICROHZ, perché può essere programmato sino ai microsecondi. Invece di usare tale Timer per la sua precisione, probabilmente lo si preferirà usare per la sua velocità. Si capisce che, a causa del mutitasking di Amiga, un task può rimanere in stato di inattività in attesa di un messaggio di Timer che deve giungere alla sua porta di messaggio. Con gli altri task che girano, il task che attende il messaggio di Timer potrebbe non svegliarsi subito quando è passato il lasso di tempo desiderato. Pertanto, quando si setta una richiesta di I/O temporizzato, e si rende il proprio task inattivo in attesa che trascorra il tempo desiderato, esso resterà effettivamente inattivo almeno per il tempo che è stato richiesto. Ma il task potrebbe non attivarsi per un lasso di tempo molto più lungo, a secondo di ciò che sta avvenendo nel sistema quando scade il periodo di attesa richiesto.

Per usare il device Timer, si devono fare le seguenti cose:

1. Selezionare quale delle unità del Timer si vuole usare e aprirla

2. Creare un blocco IORequest di Timer e scrivervi i valori del Timer stesso

3. Fissare una porta di risposta per ricevere il messaggio dal device Timer quando il tempo è trascorso

4. Scrivere nel blocco del messaggio l'indirizzo di questa porta di risposta

5. Mandare il blocco del messaggio al device Timer

6. Attendere che arrivi la risposta alla propria porta di risposta oppure fare qualcos'altro e controllare più tardi se è arrivata la risposta

7. Chiudere il Timer al termine del nuovo utilizzo

Molti dei device di sistema possono essere aperte e quindi accessibili a un solo task alla volta. Il device Timer, invece, può operare con più task contemporaneamente.

Quando si trasmette una richiesta al device Timer, lo stesso modifica il contenuto della richiesta e opera un sort sulla lista delle richieste organizzandole in una sequenza temporale. In altre parole, se tre task inviano messaggi al Timer, del tipo:

- task 1: Fammi attendere 5 minuti
- task 2: Fammi attendere 1 minuto
- task 3: Fammi attendere 2 minuti

La device modificherà le richieste di temporizzazione, riorganizzandole in modo che appaiano così:

- task 2: Fammi attendere 1 minuto
- task 3: Fammi attendere 2 minuti
- task 1: Fammi attendere 5 minuti

Così modificato l'insieme delle richieste verrà esaudito sequenzialmente dallo hardware del Timer. I blocchi delle richieste di temporizzazione vengono restituiti a ogni task, a turno, dopo che il tempo da essi specificato è trascorso.

Ogni device usa la sua versione del blocco di IORequest. Comunque si possono semplificare le cose per quanto riguarda il Timer e utilizzare la funzione di supporto standard di sistema CreateStdIO per creare un blocco di richiesta per il Timer stesso. Nel blocco di richiesta ritornato da questa funzione, i campi posso-

no essere rinominati (con un'istruzione appropriata) in modo da utilizzare dei nomi appropriati per settare i valori di tempo:

```
#define SECONDS io_Actual
#define MICROSECONDS io_Length
```

I valori di io_Actual e di SECONDS in una normale struttura dati Timeval sono definiti come dati long privi di segno, pertanto queste assegnazioni di nomi forniscono un modo conveniente per allocare della memoria per una richiesta di Timer senza dover fornire una funzione addizionale definita localmente.

Il listato 6.1 è una funzione che si può usare per generare una richiesta di Timer. Si noti che questa funzione crea anche una porta di messaggio nella quale ricevere il messaggio del Timer che viene inviato una volta trascorso il tempo desiderato. L'indirizzo della porta di risposta per il messaggio fa parte dei parametri passati a CreatePort. Si usa tale routine per stabilire un blocco di IORequest per comunicare con il Timer.

La routine InitTimer che si trova nel listato 6.2 può essere usata per settare la prima temporizzazione. In un programma, si farà qualcosa quando un messaggio di Timer viene ricevuto, poi si reinizializzerà il blocco della richiesta di Timer e lo si invierà nuovamente. Le variabili del Timer, TimerSeconds e TimerMicros, sono delle variabili globali in modo che possano essere modificate da altre routine se fosse necessario. Per usare la routine timerstuff, deve essere stato salvato il puntatore al blocco IOStdReq ritornato da PrepareTimer; si passa tale puntatore a questa routine.

Quando si ha finito di utilizzare il Timer, ci si assicuri che l'ultimo messaggio indicante il trascorrimento del tempo richiesto sia stato ricevuto, poi si cancellino le strutture dati relative il Timer mediante la routine del listato 6.3.

Il listato 6.4, in pseudocodifica, rappresenta la sequenza nella quale si debbono chiamare le routine per utilizzare il Timer. Questa è un'opportunità eccellente per trarre vantaggio delle capacità dell'Exec di sospendere l'esecuzione di un task che è in attesa che avvengano uno o più eventi.

```
/* preparetimer.c */

struct IOStdReq *tr;
struct MsgPort *tp;
struct IOStdReq *CreateStdIO();
struct MsgPort *CreatePort();

struct IOStdReq *
PrepareTimer()
{
      tr = NULL;        /* comincia senza successo */
      tp = CreatePort(0,0);   /* crea una porta senza alcun */
                              /* nome e setta la sua priorità */
                              /* a zero */
      if(tp == 0)
      {
            /* trasmetti l'errore o fa qualcosa
            * se la porta non può essere creata.
            */
            return(tp); /* ritorna NULL */
      }
      tr = CreateStdIO(tp);

      if(tr == 0)
      {
            /* nuovamente fa qualcosa se non c'è abbastanza */
            /* memoria per il blocco di IOStdReq */
      }
      return(tp);

}     /* fine di PrepareTimer() */
```

Listato 6.1

```
/* timerstuff.c */

int timerSeconds, timerMicros;/* globali */

InitTimer(trq)
      struct IOStdReq *trq;
{
      OpenDevice(TIMERNAME, UNIT_VBLANK, trq);
      SendTimer(trq);
      return(0);
```

```
}

SendTimer(trq)
        struct IOStdReq *trq;
{
        trq->SECONDS = timerSeconds;
        trq->MICROSECONDS = timerMicros;
        SendIO(trq);
        return(0);
}
```

Listato 6.2

```
/* deletetimer.c */

DeleteTimer(trq)
        struct IOStdReq *trq;
{
        struct MsgPort *mp;
        mp = trq->ReplyPort;

        DeleteStdIO(trq);

        DeletePort(mp);
        return(0);
}
```

Listato 6.3

Si può dire all'Exec che il proprio task deve svegliarsi o quando scade il tempo di attesa o quando viene ricevuto un IntuiMessage. (Si noti che si potrebbe aver usato UNIT_MICROHZ al posto di UNIT_VBLANK nel listato 6.2).

Ricordiamo come nel main del programma di Paint del quinto capitolo si usò questa istruzione che permettere di porre in stato di attesa un task:

```
WaitPort(wp->UserPort);
```

```
struct IOStdReq *myTimerRequest;

inizializzazione di un altro programma...

myTimerrequest = PrepareTimer();
                              /* inizializza una richiesta */
                              /* di temporizzazione */

InitTimer(myTimerRequest);    /* inizia la prima temporizzazione */


/* più tardi.. (loop?) */

diviene inattivo finché non scade il tempo e poi

si risveglia e fa qualcosa finché non si verifica una condizione di termi-
ne

alla condizione di termine:

AbortIO(myTimerRequest) /* si assicura che l'ultima richiesta di */
                              /* Timer è stata esaudita in modo */
                              /* da poter liberare la memoria occupata */
```

Listato 6.4

C'è un altro modo per mettere un task "a dormire". Si può attendere una certa combinazione di bit di segnalazione dove ogni bit, quando è settato a 1, dice al programmatore che qualcosa è avvenuto. (Si veda il terzo capitolo per una completa discussione delle segnalazioni). Si può attendere un bit di segnalazione da due porte diverse: la IDCMP (riferita come w->UserPort) e la porta di risposta del messaggio del Timer, che può essere riferita da:

```
myTimerRequest->ReplyPort
```

C'è un bit di segnalazione specifico associato ad ogni porta. Le seguenti istruzioni define forniscono il modo per accedere a un particolare bit di segnalazione in ognuna di queste porte:

```
#define IPOORTSIGNAL w->UserPort->mp_SigBit
#define TIMERSIGNAL myTimerRequest->ReplyPort->mp_SigBit
```

Se si vuole sospendere l'esecuzione del proprio task finché non arrivi un messaggio dal Timer o un IntuiMessage, si usa Wait al posto di WaitPort e si specifica la combinazione di bit che rappresenta gli eventi dei quali si è in attesa:

```
ULONG wakebit;                    /* fornisce un luogo nel quale */
                                  /* immagazzinare lo status di */
                                  /* quei Bit settati quando */
                                  /* il task si è risvegliato */

wakebits = Wait(IPORTSIGNAL | TIMERSIGNAL);
```

Poi, quando il task si risveglia, si elabora la chiamata di riattivazione come mostrato nel listato 6.5.

```
/* multiwakeup.c */

if (wakebits & IPORTSIGNAL)
{
      /* svuota la IDCMP UserPort, elabora tutti i messaggi */
      /* come è già stato fatto in event2.c */
                        /* nel quinto capitolo */
}
if (wakebits & TIMERSIGNAL)
{
      /* Se viene inviato un solo messaggio di Timer, si potrà */
      /* ricevere un solo messaggio dal Timer stesso */
      /* Rimuove il messaggio dalla porta di risposta */
      /* e la riusa per richiedere il prossimo messaggio del */
      /* Timer. */

      GetMsg(myTimerRequest->ReplyPort);
      timer->SECONDS = timerSeconds;
      timer->MICROSECONDS = timerMicros;
      SendIO(myTimerRequest);

      /* il programma ritornerà con il Loop all'istruzione */
      /* Wait finché non saranno stati elaborati entrambi */
      /* i tipi di messaggi */

}
```

Listato 6.5

Come parte del processo finale, si può richiamare DeleteTimer, che restituisce, alla lista della memoria libera di sistema, tutta la memoria allocata da PrepareTimer.

Il Console device

Accade molto spesso che un programmatore voglia emulare le operazioni svolte da un semplice terminale ASCII. Amiga può effettuare un emulazione di terminale in una o più Window di Intuition mediante l'uso del Console device. Questo device segue la maggior parte degli standard del codice dei terminali ANSI per le sequenze di spostamento del cursore e per il controllo del display. Inoltre, vi sono parecchi codici interpretati dal Console device che sono solo caratteristici di Amiga. Si veda il sesto capitolo dell'Amiga ROM Kernel Manual per i dettagli riguardanti il modo esatto di manipolazione dei codici ANSI da parte del Console device.

La Console deve essere associata a una Window dell'Intuition. Prima di creare una Console, si deve pertanto creare una Window. Se si allega un IDCMP alla Window, l'IDCMP ha la precedenza sul Console device per quanto riguarda la ricezione dei messaggi. Per esempio, se si specifica RAWKEYS o VANILLAKEYS tra i Flag IDCMP quando si apre una Window, e poi si allega una Console a tale Window, la Console non riceverà alcun messaggio riguardante la tastiera. Pertanto, non potrà mai interpretare alcuna pressione di tasti.

Il listato 6.6 fornisce alcune routine di supporto che possono essere incluse per permettere a un programma di comunicare effettivamente con il Console device. Queste routine forniscono il modo per creare e cancellare una Console, e per leggere e scrivere dei caratteri su di una Console - magari più caratteri con una sola chiamata di funzione.

Il vantaggio di allocare della memoria e di passarvi dei dati come mostrato nella routine del listato 6.6 consiste nel fatto che ogni volta che si apre una Console device, si ottiene un unico puntatore ai blocchi IORequest della Console stessa (se tutto ha funzionato a dovere). Pertanto in seguito ci si può riferire a una Console specifica utilizzando il suo unico puntatore ai suoi blocchi di messaggio. Per rimuovere una particolare Console, si usa DeleteConsole, passandole il puntatore ai blocchi di messaggio della Console.

Codici dei caratteri della Console

Il Console device riceve i suoi Input da svariate entità Hardware e Software. La tastiera di Amiga trasmette un set di codici di tasti - un codice unico per ogni tasto, con un codice diverso nel caso di pressione o di rilascio del tasto. Il Keyboard device raccoglie le pressioni dei tasti e le fornisce come eventi di Input all'Input device che a sua volta riunisce le informazioni provenienti dal Keyboard device e dal Gameport device in unico flusso di dati che invia a Intuition.

```
/* consolestuff.c */

extern struct IOStdReq *CreateStdIO();
extern struct MsgPort *CreatePort();

struct ConIOBlocks {
      struct IOStdReq *writeReq;/* richiesta I/O write */
      struct IOStdReq *readReq;/* richiesta I/O read */
      struct MsgPort *tpr;    /* puntatore alla ReplyPort */
                              /* per la lettura della */
                              /* Console */
};

struct ConIOBlocks *
CreateConsole(window)
      struct Window *window;
{
      struct ConIOBlocks *c;

      struct MsgPort *tpw;

      int error;

      c = (struct ConIOBlocks *)AllocMem(
         sizeof(struct ConIOBlocks), MEMF_CLEAR);

      if (c == 0)       /* out of RAM */
         goto cleanup1;

      tpw = CreatePort(0,0);
                        /* Porta di risposta per la scrittura */
      if (tpw == 0)
         goto cleanup2;

      c->tpr = CreatePort(0,0);
                        /* Porta di risposta per la lettura */
      if (c->tpr == 0)
         goto cleanup3;
```

```
        c->writeReq = CreateStdIO(tpw);
        if(c->writeReq == 0)
           goto cleanup4;

        c->readReq = CreateStdIO(c->tpr);
        if(c->readReq == 0)
           goto cleanup5;

        c->writeReq->io_Data = (APTR)window;
        c->writeReq->io_Length = sizeof(struct Window);

        error = OpenDevice("console.device",0,c->writeReq,0);
        if(error != 0)
           goto cleanup6; /* non posso aprire la Console! */

        c->readReq->io_Device = c->writeReq->io_Device;
        c->readReq->io_Unit   = c->writeReq->io_Unit;

        /* Sopra viene copiato il blocco della richiesta di I/O */
        /* inizializzato da un Open che ha avuto successo */
        /* Significa che sia la lettura sia la scrittura */
        /* stanno rispondendo ad una stessa */
        /* richiesta di Console */

        return(c);  /* puntatore ai ConIOBlocks
                       /* contenenti sia i blocchi di controllo */
                       /* della */
                       /* scrittura sia quelli della lettura */

cleanup6:
        DeleteStdIO(c->readReq);
cleanup5:
        DeletePort(c->tpr);
cleanup4:
        DeleteStdIO(c->writeReq);
cleanup3:
        DeletePort(tpw);
cleanup2:
        FreeMem(c, sizeof(struct ConIOBlocks));
cleanup1:
        return(NULL);
}

DeleteConsole(c)
        struct ConIOBlocks *c;
{
        struct MsgPort *mp;
        AbortIO(c->readReq);
                    /* abortisce qualunque lettura stia avvenendo */
        CloseDevice(c->writeReq); /* chiude la device Console */

        mp = c->writeReq->io_Message.mn_ReplyPort;

        DeleteStdIO(c->writeReq);
        DeletePort(mp);
```

```
        mp = c->readReq->io_Message.mn_ReplyPort;

        DeleteStdIO(c->readReq);
        DeletePort(mp);

        FreeMem(c, sizeof(struct ConIOBlocks));
        return(0);
    }

    #define CONREAD c->readReq
    #define CONWRITE c->writeReq

    /* chiede alla Console di leggere un carattere */
                /* in modo asincrono */

    EnqueueRead(c, location)
        struct ConIOBlocks *c;
        char *location;
    {
        struct IOStdReq *conr;
        conr = c->readReq;

        conr->io_Command = CMD_READ;
        conr->io_Length = 1;
        conr->io_Data = (APTR) location;
        /* Buffer per i dati letti */
        SendIO(conr);
    /* stazione di attesa asincrona per la richiesta di lettura */

    }

Blocks
                            /* contenenti sia i blocchi di controllo */
                            /* della */
                            /* scrittura sia quelli della lettura */

    cleanup6:
        DeleteStdIO(c->readReq);
    cleanup5:
        DeletePort(c->tpr);
    cleanup4:
        DeleteStdIO(c->writeReq);
    cleanup3:
        DeletePort(tpw);
    cleanup2:
        FreeMem(c, sizeof(struct ConIOBlocks));
    cleanup1:
        return(NULL);
    }

    DeleteConsole(c)
        struct ConIOBlocks *c;
    {
        struct MsgPort *mp;
```

```
        AbortIO(c->readReq);
                    /* abortisce qualunque lettura stia avvenendo */
        CloseDevice(c->writeReq); /* chiude la device Console */

        mp = c->writeReq->io_Message.mn_ReplyPort;

        DeleteStdIO(c->writeReq);
        DeletePort(mp);

        mp = c->readReq->io_Message.mn_ReplyPort;

        DeleteStdIO(c->readReq);
        DeletePort(mp);

        FreeMem(c, sizeof(struct ConIOBlocks));
        return(0);
}

#define CONREAD c->readReq
#define CONWRITE c->writeReq

/* chiede alla Console di leggere un carattere */
            /* in modo asincrono */

EnqueueRead(c, location)
        struct ConIOBlocks *c;
        char *location;
{
        struct IOStdReq *conr;
        conr = c->readReq;

        conr->io_Command = CMD_READ;
        conr->io_Length = 1;
        conr->io_Data = (APTR) location;
              /* Buffer per i dati letti */
        SendIO(conr);
/* stazione di attesa asincrona per la richiesta di lettura */

}
```

Listato 6.6

Se Intuition non interpreta direttamente gli eventi IDCMP, essi passano attraverso il Console device e vengono tradotti in uno o più caratteri per ogni pressione di tasto.

Vi è una mappa speciale usata dalla Console per tradurre le varie combinazioni di pressioni di tasti. La mappa di default della Console è fatta in modo tale che ciò che si vede sulla tastiera di Amiga sia ciò che si ottiene quando si pre-

mono i tasti. I tasti delle lettere, per esempio, vengono tradotti in caratteri ASCII maiuscoli o minuscoli secondo lo stato del tasto Shift. I tasti dei numeri sulla tastiera e sul tastierino numerico vengono tradotti nel loro equivalente ASCII. Il tasto Enter ha lo stesso effetto del tasto Return. Del, Tab, e spazio hanno l'effetto dell'equivalente ASCII.

I tasti speciali, come le frecce, e il tasto Help, trasmettono più di un carattere alla Console quando vengono premuti. I valori per questi tasti speciali sono listati nella tabella 6.1. Il carattere di Introduzione di Sequenza di Controllo viene espresso in tale tabella da <CSI>. Si tratta di un singolo carattere e corrisponde al valore esadecimale 9B. Nell'inserimento dei caratteri da Console, un valore 9B significherà che qualcuno ha premuto uno dei tasti speciali.

Si noti che gli ultimi due valori shiftati della tabella 6.1 hanno uno spazio tra il <CSI> e il carattere. Questo spazio fa parte della sequenza che viene trasmessa.

Il settare e il cambiare la mappa della Console va oltre gli scopi di questo libro (si veda l'Amiga ROM Kernel Manual).

Eventi complessi di Input

Si possono leggere eventi complessi di Input attraverso il Console device. Alcuni di tali eventi sono i codici dei tasti, gli eventi riguardanti il mouse, la selezione dei menu, la selezione dei gadget, e l'inserimento e la rimozione dei dischi. Si può inviare una speciale sequenza di comando, scrivendola sulla Console, al Console device in modo da richiedere la ricezione di tali eventi. Comunque, ricevere eventi di Input complessi attraverso la Console non è raccomandabile. Vi sono troppe operazioni ad alto livello da compiere. Le agevolazioni dell'IDCMP delle Window di Intuition possono fornire tutto ciò che riguarda tali eventi con molte meno complicazioni.

Settando RAWKEY tra i flag IDCMP nella Window di Intuition, si può impedire effettivamente che il Console device riceva qualunque Input proveniente dalla tastiera. Si riceverà, comunque, qualunque evento riguardante la pressione e rilascio dei tasti nella Window attraverso l'IDCMP, e si potrà interpretarli e rispondervi nel modo che si ritiene più opportuno.

Si ricordi che gli eventi RAWKEY non subiscono traduzione alcuna. Si ottengono i codici che si riferiscono alla pressione o al rilascio di qualunque tasto l'utente prema sulla tastiera di Amiga. La figura 6.1 rappresenta i codici che ogni tasto trasmette. Quando si riceve un IntuiMessage con un messaggio di classe RAWKEY (di veda il quinto capitolo), il codice del messaggio riferisce quale dei tasti è stato premuto. I codici mostrati nella figura 6.1 si riferiscono solo alle pressioni dei tasti. Se i codici riportano invece il rilascio del tasto, il valore sarà uguale a quello mostrato aumentato del numero esadecimale 80.

Si noti che il campo del messaggio IDCMP di nome Qualifier contiene informazioni aggiuntive sullo stato corrente della tastiera, come ad esempio quale tasto Shift, quale tasto Amiga, o quale tasto Alt è premuto e informa anche della pressione del tasto Control. Si veda il File Include di nome devices/inputevent.h per informazioni riguardanti i bit del campo Qualifier.

Key	Codice del Carattere	
	(Shift non premuto)	(Shift premuto)
F1	<CSI>0~	<CSI>10~
F2	<CSI>1~	<CSI>11~
F3	<CSI>2~	<CSI>12~
F5	<CSI>4~	<CSI>14~
F6	<CSI>5~	<CSI>15~
F7	<CSI>6~	<CSI>16~

F8	<CSI>7~	<CSI>17~
F9	<CSI>8~	<CSI>18~
F10	<CSI>9~	<CSI>19~
Help	<CSI>?~	<CSI>?~
]]fs[[	<CSI>A~	<CSI>T~
]]fg[[	<CSI>B~	<CSI>S~
]]fd[[	<CSI>C~	<CSI> A~
]]fs[[	<CSI>D~	<CSI> @~

Tabella 6.1

Controllare il Console device

Invece di stampare semplicemente i caratteri sulla Console, si possono usare determinate sequenze di controllo per far si che la Console muova il cursore, cancelli l'area dello schermo, cambi il modo di visualizzazione dei caratteri, e così via. In più, usando OpenFont e SetFont, si possono cambiare i Font usati dalla Console.

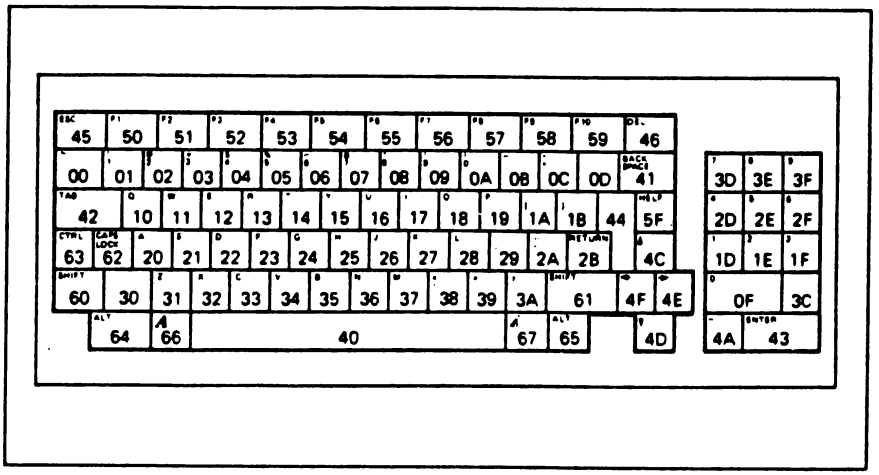


Fig. 6.1

Quando si cambia un font nella window della propria Console, il Console device si raggiusterà secondo il nuovo Font. Essa cancella l'area di disegno, poi calcola il nuovo numero di caratteri per linea e di linee per l'area della Window. Non appena si introducono nuovi caratteri da stampare nella Console, saranno già operativi i nuovi valori.

Per controllare le caratteristiche di una Console si devono utilizzare speciali serie di comandi da inviare al Console device. Essi sono listati nella tabella 6.2.

Le sequenze dei caratteri di controllo possono essere incanalate nella corrente di dati in Output, insieme al normale testo. Le sequenze da trasmettere sono rappresentate nella tabella come singoli Byte composti di due cifre esadecimali.

Nella tabella, il valore <N> e <M> possono essere rimpiazzati da uno o più numeri decimali. Per esempio, l'inserimento di <N> spazi, dove <N> è uguale a 12, sarà trasmesso al Console device come byte in tale modo:

```
9B 31 32 40
```

oppure in rappresentazione ASCII:

```
<CSI>12@
```

L'annotazione <uno o più valori> per la Selezione della rappresentazione grafica rappresenta una serie di byte che consiste nello stile di testo, la selezione del testo in primo piano, e la selezione del testo sullo sfondo. Tutte le selezioni sono opzionali, ma se vengono fatte delle selezioni multiple contemporaneamente, devono essere separate da due punti, cioè dal valore esadecimale 3B. I valori dello stile del testo sono (in esadecimale):

```
00 Plain

01 Bold

03 Italic

04 Undelined

07 Reverse
```

Comando	Sequenza di byte da trasmettere
Spazio(distruttivo)	08
Line Feed	0A
Tab Verticale	0B
Form Feed (cancella la Consolle)	0C
Return	0D
Shift In(Undo Shift Out)	0E
Shift Out	0F
Escape	1B
<CSI>	9B
Resetta allo stato Iniziale	9B 63
Inserire <N> spazi	9B <N> 40
Cursore in alto di <N> spazi	9B <N> 41
Cursore in basso di <N> spazi	9B <N> 42
Cursore a sinistra di <N> spazi	9B <N> 43
Cursore a destra di <N> spazi	9B <N> 44
Cursore alla <N>esima linea seguente (colonna 1)	9B <N> 45
Cursore alla <N>esima linea precedente(colonna 1)	9B <N> 46
Sposta il Cursore alla Riga, Colonna	9B <M> 3B <N> 48
Cancella sino alla fine della Window	9B 4A

Cancella sino alla fine della linea	9B 4B
Inserisci una linea sopra questa	9B 4C
Cancella questa linea	9B 4D
Cancella <N> caratteri	9B <N> 50
Opera uno Scroll in alto di <N> linee	9B <N> 53
Opera uno Scroll in basso di <N> linee	9B <N> 54
Set Mode(Line Feed = Return-Line Feed)	9B 32 30 68
Reset Mode (Line Feed = Line Feed)	9B 32 30 6C
Setta la lunghezza della pagina a <N> (ricalcola <N> come numero max di linee del Font corrente per cercare di riempire la Window della Console)	9B <N> 74
Setta la lunghezza della linea a <N> (considerando la larghezza del Font corrente inserisce un max di <N> caratteri per linea	9B <N> 75
Setta l'Offset di sinistra a <N> (stabilisce <N> colonne di spazio al margine sinistro della Window prima del primo carattere a sinistra)	9B <N> 78

Setta l'Offset superiore a <N> (lascia <N> lineevuote prima di iniziare il testo)	9B <N> 79
Seleziona la visualizzazione grafica	9B <uno o più valori> 6D
Riporta lo Status della device (riporta posizione del mouse)	9B 6E
Riporta lo Status della Window (riporta le dimensioni della Window)	9B 71

Tabella 6.2

I colori primo piano del testo possono essere selezionati tra i primi otto colori disponibili nella tavolozza dello schermo, con numeri esadecimali che vanno da 30 a 37 che rappresentano, appunto, i colori da 0 a 7. I colori sullo sfondo del testo possono essere scelti nello stesso modo, mediante numeri esadecimali da 40 a 47 in corrispondenza dei colori da 0 a 7.

Si possono scegliere più caratteristiche mediante l'invio di una singola sequenza di byte. Per esempio, per selezionare un testo Bold, in Italic, con un colore in primo piano numero 2 e un colore di fondo numero 3, si invii la seguente sequenza alla Console device:

```
9B 01 3B 03 3B 32 3B 43 3B 6D
```

Il comando di device Status Report ritorna la posizione corrente del cursore nella sequenza dei byte così:

```
9B <riga.cursore> 3B <colonna.cursore> 52
```

dove <riga.cursore> è una o più cifre decimali (hex 30-39) rappresentanti la riga del cursore all'interno della window della Console, e <colonna.cursore> è il numero della colonna del cursore, anch'esso trasmesso in decimale. La posizione in alto a sinistra è la (1,1).

Il comando Window Status Report fornisce informazioni riguardo a quante righe di quante colonne di testo del font corrente possono essere inserite nella Window. I valori vengono restituiti come una serie di byte del tipo:

```
9B 31 3B 31 3B <righe> 3B <colonne> 73
```

dove <righe> e <colonne> vengono trasmesse in una o più cifre decimali. Il listato 6.7 utilizza le Subroutine di Console precedentemente trattate per dimostrare alcune delle caratteristiche del Console device, compreso l'Input-Output da Console, e il controllo della Console stessa.

L'Input device

L'Input device riunisce gli eventi in Input dalla tastiera e dal mouse in un unico flusso di dati in Input. L'inserimento e la rimozione di un disco vengono ricevuti attraverso tale device così come ciò che accade alla tastiera e al mouse.

```
/* conmain.c */

/* Si possono creare Console multiple, ci si riferisce */
/* ad ognuna di esse mediante il puntatore ritornato dalla */
/* funzione CreateConsole */

/* EnqueueRead comprende una locazione nella quale devono */
/* essere posti i dati per una particolare Console */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "exec/memory.h"

ULONG IntuitionBase;

struct NewWindow nw = {
      10, 10,
      300, 100,
      -1, -1,
      0,
      GIMMEZEROZERO|ACTIVATE|SIMPLE_REFRESH|WINDOWDRAG,
      0,
      NULL,
      "Console Window",
      NULL,
      NULL,
```

```
        0,0,0,0,
        WBENCHSCREEN };

char homecursor[] = { 0x9b, '1', 0x3b, '1', 0x48 };
char backspace[]  = { 0x08 };
char linefeed[]   = { 0x0a };
char carreturn[]  = { 0x0d };
char cursorfwd[]  = { 0x9b, 0x43 };
char formfeed[]   = { 0x0c };
char insertchar[] = { 0x9b, 0x40 };
char deletechar[] = { 0x9b, 0x50 };

#define HOMECURSOR(c) WriteConsole(c,homecursor,5);
#define BACKSPACE(c)  WriteConsole(c,backspace,1);
#define LINEFEED(c)   WriteConsole(c,linefeed,1);
#define CARRETURN(c)  WriteConsole(c,carreturn,1);
#define CURSORFWD(c)  WriteConsole(c,cursorfwd,2);
#define FORMFEED(c)   WriteConsole(c,formfeed,1);
#define INSERTCHAR(c) WriteConsole(c,insertchar,2);
#define DELETECHAR(c) WriteConsole(c,deletechar,2);

/* Si possono aggiungere Array e definizione per ampliare */
/* l'esempio */
#include "console.c"

main()
{
        struct ConIOBlocks *cio, *CreateConsole();
        struct Window *w, *OpenWindow();
        int i;
        char mybuffer[1]; /* dove mettere il carattere */
        int myinput;
        char mychar[1];

        IntuitionBase = OpenLibrary("intuition.library",0);
        if(IntuitionBase == 0)
        {
                printf("L'Intuition non si apre!\n");
                exit(99);
        }
        w = OpenWindow(&nw);
        if(w == 0)
        {
                printf("La Window non si apre!\n");
                goto finish1;
        }
        cio = CreateConsole(w);
                /* allega una Console alla Window */

        if(cio == 0)
        {
                printf("Non posso creare la Console!\n");
                goto finish2;
        }
        EnqueueRead(cio,mybuffer);
```

```
/* fatto una volta, settato per la lettura */

WriteConsole(cio, "Ciao!\n\r", 13);
WriteConsole(cio, "Spazio di testo",14);

for(i=0; i<14; i++)
{
      BACKSPACE(cio);
      Delay(25);
}
LINEFEED(cio);
CARRETURN(cio);

WriteConsole(cio, "Test del cursore in avanti", 19);
CARRETURN(cio);

for(i=0; i<19; i++)
{
      CURSORFWD(cio);
      Delay(25);
}
LINEFEED(cio);
CARRETURN(cio);

WriteConsole(cio, "Test inserisci carattere", 21);
CARRETURN(cio);
for(i=0; i<8; i++)
{
      INSERTCHAR(cio);
      Delay(25);
}

LINEFEED(cio);
CARRETURN(cio);
WriteConsole(cio, "********Test cancella carattere", 29);
CARRETURN(cio);
for(i=0; i<8; i++)
{
      DELETECHAR(cio);
      Delay(25);
}
LINEFEED(cio);
WriteConsole(cio, "Test cursore in Home ", 19);
Delay(50);  /* attendi un attimo */

HOMECURSOR(cio);

Delay(100); /* 2 secondi */

FORMFEED(cio);
LINEFEED(cio);

WriteConsole(cio, "Form Feed Cleared Console\n\r", 27);
WriteConsole(cio, "Inserisci una linea...\n\r\n\r", 25);
WriteConsole(cio, "Faccio un echo dei caratteri\n\r", 28);
```

```
        WriteConsole(cio, "finché non premi RETURN\n\r", 24);

        /* I tasti Help e funzione non hanno alcun effetto qui */

        do
        {
              /* Un solo carattere alla volta */
              /* Magari si vuole creare un altro task */
              /* che preleva dei caratteri mentre la Console */
              /* fa qualcos'altro */

              myinput = CGetCharacter(cio, TRUE); /* si, aspetta */
              mychar[0] = (char)(myinput & 0xff);
              WriteConsole(cio, mychar, 1);
        }
        while(mychar[0] != '\r'); /* carattere RETURN */

    finish3:
        DeleteConsole(cio);
    finish2:
        CloseWindow(w);
    finish1:
        CloseLibrary(IntuitionBase);
        exit(0);
}
```

Listato 6.7

Il Keyboard device

Il Keyboard device fornisce dati esclusivamente all'Input device. Nella versione corrente dell'Exec, di Intuition, e dell'AmigaDOS, non è possibile escludere lo Input device per accedere direttamente alla device Keyboard.

E' consigliabile prelevare gli Input da tastiera direttamente dall'IDCMP dell'Intuition se il task usa una Window per l'Input, oppure accodarsi all'Input device e intercettare i messaggi dalla tasiera se non vi sono Window disponibili.

Il Gameport device

Anche il Gameport device fornisce dati esclusivamente all'Input device. Tutti gli eventi in Input dal mouse vengono inviati all'Input device, il quale, a turno, li rende disponibili a Intuition e a se stesso.

Ci sono un altro paio di possibilità: si può comunicare con l'Input device e assegnare il proprio mouse alla seconda porta, invece che alla porta di sinistra, come è di default. Inoltre, si può comunicare con la Gameport e usarla per i joystick e per altre periferiche.

Come per altri device, bisogna creare un blocco di IORequest per comunicare con la Gameport device. Si dice alla device che tipo di controller è stato connesso alle porte (un mouse o un joystick) e come deve rispondere al device. Il programma del listato 6.8 può essere compilato sia per utilizzare un mouse sia per utilizzare un joystick. Di default e settato per un mouse. Per compilare il programma per un joystick, si cambi la prima istruzione da #define MOUSE 1 in #define JOYSTICK 1 e lo si ricompili.

Ampliamenti della tastiera

Vi sono delle applicazioni nelle quali sono raccolti dei programmi che attuano un rimappamento della tastiera; alcuni di tali programmi sono detti Keyboard Enhancers (ampliatori della tastiera). Per tradurre un Input da tastiera in un evento di diverso tipo, si possono filtrare tutti gli Input attraverso la Console device o attraverso l'IDCMP e tradurre le sequenze relative ai tasti prima che raggiungano l'applicazione desiderata.

Se, comunque, si tenta di sviluppare e implementare qualcosa che possa funzionare con qualsiasi applicazione invece che solo con una certa applicazione all'interno del suo schermo personale, allora bisogna scendere più profondamente nel sistema. In particolare, si dovrebbe scrivere una routine di gestione degli eventi personalizzata da istallare nella catena degli Event Handler.

Vi è una priorità nel posizionamento all'interno di tale catena, per cui Intuition, se intercetta l'evento, potrebbe gestirlo per se e non passarlo mai alla routine di gestione del programmatore. Così sarebbe conveniente istallare tale routine, nella catena, davanti a Intuition in modo da poter esaminare gli eventi in entrata e magari sostituirvi una nuova sequenza di eventi per la propria routine da passare poi a Intuition.

Con questo metodo si può creare un programma che registra i movimenti del mouse effettuati dall'utente, le pressioni dei pulsanti di selezione e di menu, e gli eventi legati alla tastiera (in modo Record). Inoltre si può creare una routine che sia in grado di ripetere una serie di eventi del mouse e della tastiera. O, ancora, si potrebbe creare un ampliamento della tastiera che, per esempio, possa tradurre la pressione di F3 in "Copy myFile TO myFile.back".

La prima considerazione da fare è che ogni evento in Input che si registra occupa circa 24 Byte (le dimensioni della struttura dati InputEvent). Gli eventi della tastiera vengono trasmessi alle applicazioni un tasto alla volta (premuto o rilasciato). Una volta nell'applicazione, per esempio. si può riutilizzare la memoria occupata dall'InputEvent, prelevando solo uno o alcuni degli eventi che arrivano a ogni controllo dell'I/O, e salvando solo quei campi di dati dell'evento ai quali è interessata la applicazione in uso. Per una situazione in modo Record, sarebbe preferibile salvare ogni cosa riguardante ogni singolo evento in arrivo.

```
  /* joymouse.c */

/* può essere definito solo uno dei due ogni volta */

  #define MOUSE 1
/*
  #define JOYSTICK 1
*/
  #include <exec/types.h>
  #include <exec/devices.h>
  #include <graphics/gfx.h>
  #include <devices/gameport.h>
  #include <devices/inputevent.h>

  #define abs(x) (x < 0 ? -(x):(x))

  extern struct IOStdReq *CreateStdIO();
                          /* funzioni in uso */
  extern struct MsgPort  *CreatePort();

  main()
```

```
{
int error,errout,delta,timeouts;
struct GamePortTrigger trig;
struct IOStdReq *gameMessage; /* una richiesta di I/O */
struct MsgPort  *gameReplyPort;/* dove ritornare il msg */
struct InputEvent gameEvent;
                        /* dove immagazzinare l'evento */

struct InputEvent *ge;        /* puntatore a quel luogo */
UBYTE *g;
ge = &gameEvent;

gameReplyPort = CreatePort(0,0);
if(gameReplyPort == 0)
{
      exit(100);  /* impossibile creare la porta */
}
gameMessage = CreateStdIO(gameReplyPort);
if(gameMessage == 0)
{
      DeletePort(gameReplyPort);
      exit(101);  /* impossibile creare il messaggio */
}
error = OpenDevice("gameport.device",1,gameMessage,0);
/* l'unità 0 rappresenta la porta di sinistra */
/* l'unità 1 della device rappresenta quella di destra */
if(error)
{
      errout = 102;
      goto cleanup;
}
/* ora diciamo che dispositivo usiamo */

gameMessage->io_Command = GPD_SETCTYPE;
gameMessage->io_Length  = 1;
                  /* un Byte per settare il tipo */
gameMessage->io_Data    = (APTR)&gameEvent;
                  /* dove trovare i dati */

g     = (UBYTE *)&gameEvent;
#ifdef MOUSE
*g    = (char)GPCT_MOUSE;
delta = 5;
      /* riporta i movimenti del mouse se sono avvenuti almeno */
      /* cinque microspostamenti */

 #endif MOUSE

#ifdef JOYSTICK
*g    = (char)GPCT_ABSJOYSTICK;
delta = 1;
      /* riporta i movimenti del joystick */
#endif JOYSTICK

DoIO(gameMessage);
```

```
if(gameMessage->io_Error)
{
      errout = 103;
      goto cleanup; /* errore nel settare il tipo */
}
/* Setta le condizioni di informazione*/
/* quando e come deve rispondere la device? */

gameMessage->io_Command = GPD_SETTRIGGER;
gameMessage->io_Length  = sizeof(trig);
gameMessage->io_Data    = (APTR)&trig;

/* informa nel caso i pulsanti del mouse o il pulsante di */
/* Fire del joystick vengano premuti */

trig.gpt_Keys = GPTF_UPKEYS + GPTF_DOWNKEYS;

/* Dà informazioni con un intevallo di 10 secondi nel caso */
/* di movimenti, di pressione di tasti o di qualsiasi cosa. */
/* (Temporizzato in 60esimi di secondo mediante l'unità */
/* di VBLANK nei sistemi USA */

trig.gpt_Timeout = 10 * 60;

/* Da informazioni se il valore del delta è maggiore o */
/* uguale ai seguenti valori */

trig.gpt_XDelta = delta;
trig.gpt_YDelta = delta;

DoIO(gameMessage);
if(gameMessage->io_Error)
{
      errout = 104;
      goto cleanup;
}
timeouts = 0;

gameMessage->io_Command = GPD_READEVENT;
gameMessage->io_Data = (APTR)&gameEvent;

do {
      gameMessage->io_Length = sizeof(struct InputEvent);

      DoIO(gameMessage);

      switch(ge->ie_Code) {

            case (IECODE_LBUTTON):
                  printf("premuto pulsante sinistro\n");
                  break;

            case (IECODE_LBUTTON | IECODE_UP_PREFIX):
                  printf("rilasciato pulsante sinistro\n");
```

```
                    break;

                case (IECODE_RBUTTON):
                    printf("premuto pulsante destro\n");
                    break;

                case (IECODE_RBUTTON | IECODE_UP_PREFIX):
                    printf("rilasciato pulsante destro\n");
                    break;

                case (IECODE_NOBUTTON):
                    if(abs(gameEvent.ie_X) < delta &&
                       abs(gameEvent.ie_Y) < delta)
                    {
                        printf("timeout;\n");
                        timeouts++;
                    }
                    else
                    {
                        printf("avvenuto movimento;\n");
                    }

#ifdef MOUSE
                    printf("seguenti movimenti mouse:\n");
#endif MOUSE
#ifdef JOYSTICK
                    printf("seguenti movimenti joystick:\n");
#endif JOYSTICK
                    printf("x-delta = %ld\n",
                           gameEvent.ie_X);
                    printf("y-delta = %ld\n",
                           gameEvent.ie_Y);
                default:
                    break;
                }
            }
            while(timeouts < 10);
            errout = 0;
            /* per il joystick questo programma non può dire */
            /* la differenza tra timeout e la transizione da */
            /* Switch-on a Switch-off del joystick */

cleanup:
            DeleteStdIO(gameMessage);
            DeletePort (gameReplyPort);
            printf("Fatto!\n");
            exit(errout);
}           /* fine del main */
```

Listato 6.8

Vi è un'altra considerazione interessante che riguarda il filtraggio degli eventi della tastiera forniti dall'Input device. Cosa si deve fare infatti se un utente passa continuamente da una applicazione all'altra tra quelle attive contemporaneamente nello stesso schermo (o in schermi multipli)? Cosa fare se un utente digita due o tre caratteri in una applicazione, poi attiva un'altra applicazione e digita qui un'altro paio di caratteri?

Se si sta cercando di creare un espansore di abbreviazioni, tale programma di espansione potrebbe non essere in grado di dire quale applicazione ha ricevuto una parte del flusso di dati in Input. Per esempio, si supponga di avere due Wordprocessor in due differenti Window sullo stesso schermo e di avere un espansore di abbreviazioni che deve leggere la tastiera, e si immagini di dover espandere la parola "molt" per poter ricostruire la parola di senso compiuto "moltiplica". Se l'utente digita "mo" in una window, poi seleziona la seconda window e digita "lt", si potrebbe voler evitare di espandere la parola nella seconda window (poiché tale window non ha ricevuto l'abbreviazione completa "molt"). Potrebbe essere necessario dire all'utente che la ricostruzione espansa avviene solo per parole le cui abbreviazioni siano totalmente scritte senza l'intervento di movimenti del mouse.

L'utilizzazione completa della device Input non richiede solo l'uso del linguaggio C, ma anche del linguaggio macchina di Amiga. Ci si riferisca quindi all'Amiga ROM Kernel Manual per ulteriori informazioni riguardanti la costruzione e il collegamento di routine di gestione dell'Input device; tale manuale contiene un buon esempio che mostra come una tale routine possa essere creata e inserito nella corrente dei dati in Input.

Capitolo 7

L’animazione

Questo capitolo descrive gli strumenti per l'animazione disponibili su Amiga: il sistema a sprite semplici e il sistema Gel (Graphics element). Il sistema Gel è costituito dai Bobs (Blitter objects) e dagli sprite virtuali. Per animare un oggetto sullo schermo utilizzando il sistema Gel, si definisce l'oggetto e il suo movimento in relazione agli altri oggetti dello schermo. Il sistema Gel utilizza gli sprite hardware per definire gli sprite virtuali, e disegna i Bobs molto velocemente utilizzando il Blitter, un coprocessore che può muovere e combinare i dati grafici in modo molto veloce.

Gli oggetti vengono disegnati nella sequenza e nella posizione definite dal programmatore, con un opzione che permette di salvare l'area ricoperta dall'oggetto quando questo viene disegnato. Poi, quando l'oggetto viene spostato in una nuova locazione, l'area dello sfondo originale viene ripristinata, e l'oggetto viene disegnato nella nuova posizione.

Vi sono molte opzioni fornite dal sistema Gel, che permettono di mantenere uno stretto controllo del processo di animazione. Qui vengono descritte tali opzioni, e molte vengono usate in programmi di esempio, che comprendono vari strumenti che saranno estremamente utili per chi vorrà poi utilizzarli in programmi di gestione di animazioni.

Sprite semplici

Utilizzando gli sprite semplici, si possono creare e muovere sino a sette oggetti contemporaneamente, ognuno dei quali largo 16 Pixel a bassa risoluzione e composto di tre colori a scelta tra quelli disponibili. Questi oggetti possono essere creati per essere mossi con estrema velocità sullo schermo. Per esempio, il cursore del mouse è uno sprite semplice. Tali sprite utilizzano uno dei sistemi hardware residenti in Amiga - il sistema hardware degli sprite, appunto.

La visualizzazione dello sprite è totalmente indipendente dalle condizioni dello sfondo su cui si muove (chiamato Playfield). Solo nella scelta dei colori vi è un legame tra il Playfield e lo sprite semplice.

Vi sono occasioni nelle quali si desidera manipolare il pointer del mouse, per esempio nel caso dell'uso da parte dell'utente di un joystick o di un mouse collegato alla seconda porta. Oppure si potrebbe progettare un gioco nel quale si vo-

gliono visualizzare oggetti altamente dinamici nei movimenti. Per questi compiti, sono sicuramente adatti gli sprite semplici.

Amiga possiede otto sprite semplici disponibili, ognuno dei quali legato al sistema hardware residente degli sprite. La differenza tra gli sprite semplici e gli sprite hardware sta nel fatto che il Software di sistema per gli sprite semplici limita ogni sprite hardware ad un uso singolo all'interno di una singola scansione di visualizzazione del video. E' possibile, manipolando direttamente l'hardware riguardante gli sprite, riutilizzare gli sprite hardware più volte in una singola visualizzazione. Gli sprite semplici non usufruiscono di questo vantaggio ma invece forniscono un metodo facile per determinare il posizionamento di un oggetto quando viene visualizzato sullo schermo.

La struttura dati Simplesprite

Si deve costruire una struttura dati SimpleSprite per ogni sprite che si desidera usare nel proprio programma. Tale struttura dati è la seguente (da graphics/sprite.h):

```
struct SimpleSprite
{
   UWORD *postcldata;
   UWORD height;
   UWORD x,y;
   UWORD num;
};
```

Per stabile il contenuto della maggior parte dei campi dati, si utilizzano delle routine di sistema, come ChangeSprite e MoveSprite. Comunque, si può leggere il contenuto della struttura dati stessa per determinare la locazione corrente dell'oggetto o per determinare quale sprite hardware è stato assegnato dal sistema in modo da controllare direttamente i colori dello sprite.

Ogni sprite semplice, quando appare sullo schermo, è largo 16 Pixel in modo bassa risoluzione. Non importa se lo sprite appare in uno schermo a bassa (320 Pixel) o ad alta (640 Pixel) risoluzione. Lo sprite utilizza per la propria visualizzazione sempre il modo a bassa risoluzione per i suoi componenti elementari.

L'altezza dello sprite è determinata dal valore di Height immagazzinato nella sua struttura dati SimpleSprite. La posizione dello sprite è determinata dai valori x e y della struttura stessa. Come lo sprite appaia è un fatto che dipende dalla area di memoria puntata dal puntatore posctldata.

Ottenere uno sprite

Prima di poter costruire qualunque struttura dati per il proprio sprite, si deve ottenere uno sprite dal sistema in modo da poterlo assegnare all'uso desiderato. Per ottenere uno sprite, si utilizza la funzione di sistema GetSprite. Per richiamare GetSprite si scriva:

```
spritenum = GetSprite(ssp,num);
```

dove ssp è un puntatore a una struttura dati SimpleSprite, e num è il numero specifico dello sprite che si desidera utilizzare (un numero da 0 a 7).

Se lo sprite specificato non è già stato riservato per l'uso particolare di un altro Task, il sistema ritornerà in spritenum lo stesso valore richiesto in num. Se, invece, lo sprite è già stato riservato, i sistema ritornerà il valore -1, che implica che si deve ritentare.

Si può utilizzare -1 per richiedere qualsiasi sprite disponibile. Se tale richiesta ritorna un valore -1 di spritenum, significa che sono stati utilizzati tutti gli sprite del sistema. Ecco un frammento di programma che richiede uno sprite:

```
struct SimpleSprite mysprite;
BYTE spritenum;

spritenum = GetSprite(&mysprite,-1);

if(spritenum == -1) printf ("FINE DEGLI SPRITE !!!");
```

La funzione GetSprite riempie il campo num della struttura dati SimpleSprite. Si riceve il valore spritenum solo come controllo che tutto ha funzionato correttamente. Si può utilizzare più tardi questo valore per determinare i colori dello sprite.

Cambiare uno sprite

Altri elementi della struttura dati SimpleSrite vengono stabiliti inizialmente dalla routine ChangeSprite. Per richiamare tale routine si scriva:

```
ChangeSprite(pointer,addrsprite,addrdata);
```

dove pointer è un puntatore all'indirizzo di una struttura ViewPort o un valore nullo. Se contiene l'indirizzo di una ViewPort, allora lo sprite verrà posizionato relativamente all'angolo superiore sinistro della ViewPort stessa. Nel programma Simple Sprite, si vedrà che l'indirizzo della ViewPort viene estratto da una struttura dati Screen. Se si usa Intuition per stabilire la struttura Screen, e di conseguenza la struttura dati ViewPort, lo sprite si muoverà quando lo schermo si fermerà (come si potrà vedere nel far girare il programma d'esempio).

Se pointer ha un valore nullo, allora lo sprite sarà posizionato rispetto alla struttura dati View. Essa definisce tutta l'area di visualizzazione, piuttosto che gli schermi e le window individuali. Lo sprite, quindi sarà posizionato in relazione all'angolo superiore sinistro della zona reale di visualizzazione settata mediante l'uso delle Preferences. Quando pointer è zero, gli sprite possono muoversi attraverso i confini degli schermi. Questo crea un effetto piuttosto singolare: lo sprite assume i colori dello schermo sul quale appare.

Gli altri due parametri della funzione ChangeSprite sono addrsprite e addrdata. Il parametro addrsprite è un puntatore all'indirizzo di una struttura dati SimpleSprite. Questa struttura dati deve essere già stata inizializzata da GetSprite per contenere un numero di uno sprite valido. Il parametro addrdata è il puntatore alla prima Word di una struttura che descrive fisicamente la figura dello sprite, cioè consiste dei dati contenenti i bit del pattern usato dal sistema per definire la figura stessa.

I dati dello sprite

Ecco la struttura dati puntata dal parametro addrdata di ChangeSprite. Essa definisce la figura dello sprite. Non ha un nome specifico, ma i suoi elementi sono in relazione con la struttura dati SimpleSprite.

```
/* spritedata.c */

UWORD sprite_data{} =      {

   0,0,
               /* posizione di controllo */
   0x0fc3,0x0000,
               /* dati della linea 1 dell'immagine */
   0x3ff3,0x0000,
               /* dati della linea 2 dell'immagine */
   0x30c3,0x0000,
               /* dati della linea 3 dell'immagine */
   0x0000,0x3c033,
               /* dati della linea 4 dell'immagine */
   0x0000,0x3fc3,
               /* dati della linea 5 dell'immagine */
   0x0000,0x03c3,
               /* dati della linea 6 dell'immagine */
   0xc033,0xc033,
               /* dati della linea 7 dell'immagine */
   0xffc0,0xffc0,
               /* dati della linea 8 dell'immagine */
   0x3f03,0x3f03,
               /* dati della linea 9 dell'immagine */
   0,0
               /* fine della struttura */
   };
```

I dati per uno sprite semplice hanno sempre una forma del tipo sopra mostrato. Vi sono 2 word (32 bit) di zero all'inizio della struttura, seguite da tante coppie di word quante sono le linee (altezza) dello sprite, seguite alla fine da altre due word nulle.

Le prime due word sono usate per il controllo della posizione. Per uno sprite hardware, esse stabiliscono dove debba apparire lo sprite sullo schermo (e controllano altre caratteristiche che qui si omettono). Le ultime due Word servono anch'esse come controllo per la posizione per gli sprite hardware, ma sono usate per stabilire la fine dell'uso di uno sprite per una videata. Gli sprite semplici possono essere usati solo singolarmente per uno schermo. I dati del pattern per lo sprite qui mostrato formano il carattere:

```
S!
```

I dati con i quali opera uno sprite semplice devono trovarsi nella memoria accessibile ai Chip Custom. Cioè devono essere locati nei 512K inferiori della macchina. Il programma Simple sprite di questo capitolo, utilizza i dati sopra forniti, per tutti gli sprite. Esso alloca della memoria MEMF_CHIP e vi copia i dati. Poi,

una chiamata di ChangeSprite mostra al sistema dove trovare l'area dati che può servire per ogni singolo sprite. Deve essere fornita un'area dati separata per ogni sprite poiché il sistema modifica le voci riguardanti la posizione all'interno di ogni struttura dati SimpleSprite, e poi dice allo sprite hardware dove trovare i dati occorrenti.

I colori degli sprite

Per le linee che si trovano tra le due coppie di dati riguardanti il controllo della posizione, ogni coppia di word definisce il valore del colore che deve essere usato per la particolare posizione del bit nella linea dello sprite. Si prenda la linea di dati 1 dell'immagine come esempio:

```
0x0fc3,0x0000
```

Qui, 0x0fc3 traduce questo set di bit:

```
0001111111000011
```

e 0x0000 diventa:

```
0000000000000000
```

I bit che si trovano in posizioni corrispondenti si combinano per selezionare il colore di un particolare pixel di quella linea della figura. Ecco come si formano le combinazioni:

```
Valore della prima Word:              0 0 1 1

Valore della seconda Word:            0 1 0 1

Numero del colore selezionato:        t 1 2 3
```

La t rappresenta una combinazione di colori che risulta trasparente. In ogni zona dello sprite che risulti trasparente, si può vedere qualsiasi altra parte dello schermo che abbia una priorità di video inferiore.

Gli sprite semplici 0 e 1 usano gli stessi registri colore di sistema. Gli sprite 2 e 3 sono accoppiati, gli sprite 4 e 5 sono accoppiati, e gli sprite 6 e 7 sono accoppiati. Ogni coppia di sprite ha assegnati specificamente un gruppo di registri di sistema per il colore. Questi definiscono i colori che possono essere visualizzati. La tabella 7.1 mostra i raggruppamenti.

Così, nel sistema a sprite semplici, si ha di fatto la possibilità di scegliere solo tra quattro diversi gruppi di colori per gli sprite da usare, e si deve anche ricordare che i colori sono raggruppati come mostrato.

sprite	Numero del colore	Numero
0 e 1	1	17
	2	18
	3	19
2 e 3	1	21
	2	22
	3	23
4 e 5	1	25
	2	26
	3	27
6 e 7	1	29
	2	30
	3	31

Tabella 7.1

Intuition utilizza lo sprite semplice numero 0 come suo cursore. Pertanto, tale sprite non è solitamente disponibile al programmatore per l'assegnazione. Quando si usa LoadRGB4 per fissare i colori d'uso di uno schermo personalizzato, se si caricano i colori 17-19, si manometteranno i colori del cursore di sistema così come quelli degli altri sprite.

Il programma Simple sprite ha un effetto interessante tanto sui colori del cursore del mouse quanto su quelli degli sprite semplici. Il programma definisce metà degli sprite con un posizionamento relativo all ViewPort (cioè relativo allo

schermo nel quale la window è disegnata), e l'altra metà con un posizionamento relativo all'area totale di visualizzazione.

Se si usa il cursore per muoversi sulla barra superiore dello schermo, si preme il pulsante sinistro del mouse e si abbassa lo schermo per un terzo del video, si vedrà che la posizione relativa degli sprite cambierà. E alcuni sprite attraverseranno i limiti dello schermo (come può fare il cursore del mouse) e cambieranno il proprio colore.

Questo cambiamento di colore avviene perché l'Intuition assegna dinamicamente i colori ai registri di sistema dei colori in modo da permettere a ogni schermo di avere il proprio set di 32 colori. Così, il confine tra due schermi è un luogo di transizione dove cambiano i colori.

Liberare uno sprite

Se si usa GetSprite per ottenere l'uso di uno sprite, si deve anche liberare lo sprite una volta finitone l'uso. Il sistema non tiene traccia di quale Task abbia allocato uno sprite e se il Task sta ancora girando. Se non si libera uno sprite, allora nessun altra applicazione sarà in grado di usarlo sino al successivo Reset del sistema.

Per richiamare FreeSprite si scriva:

```
FreeSprite(num);
```

dove num è il numero dello sprite allocato mediante GetSprite.

Il programma Simple Sprite

Il listato 7.1 è il programma dimostrativo delle caratteristiche del sistema Simple Sprite. Viene aperto uno schermo personalizzato a 32 colori, con una window

nello schermo per fornire un Gadget Close per poter fermare l'esecuzione del programma, e un luogo nel quale ricevere i messaggi di temporizzazione (INTUITICKS) ogni decimo di secondo.

Si noti che gli sprite semplici possono muoversi molto più velocemente di quanto sia qui mostrato. Si potrebbe voler modificare il programma per creare degli stati di attesa per gli eventi di Intuition e del Timer (come mostrato nel sesto capitolo), proprio per ottenere un'esecuzione più veloce.

Ci sono un paio di cose da notare sull'esecuzione del programma. All'inizio, lo sprite più a sinistra coincide con lo sprite hardware numero 1. L'allocazione prosegue lungo lo schermo verso destra con gli sprite hardware numero 2, 3, 4, 5, 6, e 7. (Si potrebbero cambiare i dati dell'immagine dello sprite per far si che contenga lo sprite hardware che è stato allocato per l'immagine stessa).

Le priorità video degli sprite hardware sono tali per cui quelli aventi numero di assegnazione più basso hanno priorità maggiore. In altre parole, lo sprite 0, cioè il cursore del mouse, apparirà sempre al di sopra di ogni altro sprite. Lo sprite 1 apparirà al di sopra di tutti gli altri sprite eccetto lo 0, e cos via. Il sistema è inoltre settato in modo che gli sprite appaiano al di sopra di ogni altra cosa si trovi sull'area del Playfield (cioè dello schermo). Se si vogliono risettare le priorità di sistema in modo che alcuni sprite scompaiano sotto alcune parti dello sfondo, è possibile farlo. Si veda per questo l'Amiga ROM Kernel Manual.

Si provi a portare in giù lo schermo creato da programma e si veda cosa accade alla posizione e ai colori degli sprite. Questo esperimento aiuterà a capire come usare MoveSprite e ChangeSprite per posizionare gli sprite semplici.

Si noti che vi è una differenza nella struttura NewScreen del listato rispetto a ciò che si è visto finora in questo libro. Infatti, le variabili Modes vengono settate a un valore SPRITES. Questo informa l'Intuition e le sue routine di supporto che dovranno essere rappresentati degli sprite e che, quando lo schermo sarà costruito, si dovranno fornire delle istruzioni che indichino come deve avvenire il movimento di tali sprite.

```
/* simplesprite.c */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "graphics/sprite.h"
#include "exec/memory.h"

SHORT sprgot[7];        /* quale sprite viene preso ogni volta. */
UWORD *sprdata[7];          /* sette puntatori ai dati degli sprite */
SHORT xmove[7], ymove[7];   /* direzioni di movimento */
struct SimpleSprite sprite[7];      /* sette sprite semplici */
struct SimpleSprite *spr;   /* puntatore a uno sprite */
short maxgot;               /* # max di sprite prelevati */

struct Window *w; /* puntatore ad una Window  */
struct RastPort *rp;    /* puntatore ad una RastPort */
struct Screen *s; /* puntatore ad uno Screen */
struct ViewPort *vp;    /* puntatore ad una ViewPort */

struct Window *OpenWindow();
struct Screen *OpenScreen();
LONG GfxBase;
LONG IntuitionBase;

/* Questi sono dei dati "prototipo". Ogni sprite partirà con */
/* questi dati che però poi saranno cambiati dal sistema. Non è */
/* possibile far puntare più di uno sprite semplice a uno stesso */
/* set di dati. Inoltre i dati degli sprite devono trovarsi nella */
/* RAM accessibile ai Chip Custom, pertanto se il vostro Amiga è */
/* espanso i dati non saranno accessibili essendo parte del */
/* programma. Per questo vi è più avanti una allocazione */
/* specifica di tipo MEMF_CHIP. */

/* 22 Word di dati = 44 Byte di dati per lo sprite */

UWORD sprite_data[ ] = {
                0,0,        /* controllo della posizione */
                0x0fc3, 0x0000, /* dati della linea 1 dell'immagine */
                0x3ff3, 0x0000, /* dati della linea 2 dell'immagine */
                0x30c3, 0x0000, /* dati della linea 3 dell'immagine */
                0x0000, 0x3c03, /* dati della linea 4 dell'immagine */
                0x0000, 0x3fc3, /* dati della linea 5 dell'immagine */
                0x0000, 0x03c3, /* dati della linea 6 dell'immagine */
                0xc033, 0xc033, /* dati della linea 7 dell'immagine */
                0xffc0, 0xffc0, /* dati della linea 8 dell'immagine */
                0x3f03, 0x3f03, /* dati della linea 9 dell'immagine */
                0,0 /* fine della struttura */

                /* La fine della struttura contiene i dati per */
                /* il controllo della posizione per il */
                /* successivo sprite hardware. */
```

```
                    /* il sistema degli sprite semplici supporta */
                    /* l'uso di un solo sprite hardware per */
                    /* ogni scansione di video. */
                    /* le combinazioni della prima e della */
                    /* seconda Word di ogni linea dello sprite */
                    /* fornisce il colore per un Pixel della linea */
                    /* tutti i Pixel non nulli delle linee 1-3 */
                    /* hanno il colore "1" dello sprite, */
                    /* le linee 4-6 hanno il colore "2", e le */
                    /* linee 7-9 hanno il colore "3" */

                    };

/*
      CIO' CHE SEGUE E' SOLO PER INFORMAZIONE ... il sistema Simple
      sprite setta direttamente questi bit; l'utente non deve avere
      alcuna cura di essi. Si usi la funzione Changesprite e
      MoveSprite per ottenere degli effetti sugli Sprite.

      controllo della posizione:

                    prima UWORD:

Bit 15-8, valore iniziale verticale; gli otto
Bit bassi di tale valore.

Bit 7-0, valore iniziale orizzontale; gli
otto Bit alti di tale valore.

                    seconda UWORD:

Bit 15-8, valore finale verticale; gli otto
Bit bassi di tale valore.

Bit 7 = Attach-Bit (da allegare agli sprite
per ottenere un numero maggiore di colori
(15 invece di 3), non è disponibile sul sistema
Simple Sprite, lo è per gli sprite hardware).

Bit 6-4 (non usati)
Bit 2, valore iniziale verticale (il Bit 8 di tale valore ).
Bit 2, valore finale verticale (il Bit 8 di tale valore).
Bit 2, valore iniziale orizzontale (il Bit 0 di tale valore).

*/


movesprites()
{
      short i;
      short newx, newy;

      /* Il primo parametro di MoveSprite è nullo; lo sprite viene */
      /* posizionato in relazione alla VIEW invece che alla ViewPort */
      /* Questo significa che lo sprite non è soggetto ai */
```

```
        /* movimenti dello schermo. Se lo 0 è rimpiazzato da un */
        /* valore "vp", allora lo sprite è solidale con lo schermo */

        spr = &sprite[0];

        for (i=0; i<maxgot; i++)
        {
              newx = xmove[i]+spr->x;
              newy = ymove[i]+spr->y;

              /* Può sembrare strano, ma per mostrare entrambi i modi */
              /* di posizionamento metà degli sprite sono solidali */
              /* allo schermo e metà sono solidali alla VIEW. */
              /* Quando uno sprite passa da uno schermo ad un'altro */
              /* ne assume i colori */

              if( (i % 2) == 0 )
              {
                    /* Gli sprite pari si muovono con la VIEW */
                    MoveSprite(0,spr,newx,newy);
              }
              else
              {
                    /* Gli sprite dispari si muovono con lo Schermo */
                    MoveSprite(vp,spr,newx,newy);
              }

              if(spr->x >= 320 || spr->x <= 0) xmove[i]=-xmove[i];
              if(spr->y >= 190 || spr->y <= 0) ymove[i]=-ymove[i];

              spr++;
                    /* aggiorna il puntatore per il successivo sprite */
        }
}

#define AZCOLOR 1
#define WHITECOLOR 2

/* #include "mydefines.h"*/
/* Assegnazione flag di una Window */
/* mydefines.h */

#define WC WINDOWCLOSE
#define WS WINDOWSIZING
#define WDP WINDOWDEPTH
#define WDR WINDOWDRAG

#define NORMALFLAGS (WC|WS|WDP|WDR)

/* #include "myscreen1.h" */
/* myscreen1.h */

/* myfont1 specifica le caratteristiche del font di Default */
/* Qui viene selezionato il font a 80 colonne che viene però */
```

```
/* visualizzato a 40 colonne in bassa risoluzione */

struct TextAttr myfont1 = { "topaz.font", 8, 0, 0 };

struct NewScreen myscreen1 = {
      0, 0,             /* posizione schermo */
      320, 200,   /* dimensioni */
      5,          /* piani di Bit, sono disponibili 32 colori */
      1, 0,       /* DetailPen, BlockPen */
      SPRITES,    /* Modo di visualizzazione... 0 = bassa risoluzione */
      CUSTOMSCREEN,     /* tipo di schermo */
      &myfont1,   /* Font di Default */
      "32 Color Test", /* Nome dello schermo sulla barra */
      NULL,       /* Gadget per l'utente, qui nulli */
      NULL };
                  /* indirizzo della Bitmap per questo schermo */
                  /* qui non è usata */

/*    #include "window1.h" */
/* window1.h */

struct NewWindow myWindow = {
      0,          /* LeftEdge per la Window */
                  /* misurato in Pixel a partire dall'estremità */
                  /* sinistra dello schermo */
      15,         /* TopEdge per la Window*/
                  /* misurato in Pixel a partire dall'estremità */
                  /* superiore dello schermo */
      320, 150,   /* dimensioni della Window */
      0,          /* DetailPen - penna usata per i bordi */
                  /* della Window */
      1,          /* BlockPen - penna usata per i Gadget della Window */
                  /* 1 significa che si usano i valori di Default */
      CLOSEWINDOW | INTUITICKS,
                  /* Flag IDCMP */
      SIMPLE_REFRESH | NORMALFLAGS | GIMMEZEROZERO | ACTIVATE,
                        /* Flag della Window */
      NULL,       /* primo Gadget */
      NULL,       /* CheckMark  */
      "Click Close Gadget To Stop", /* nome della Window */
      NULL,       /* puntatore allo schermo se non è Workbench */
      NULL,             /* puntatore alla Bitmap se è */
                        /* una Window SUPERBITMAPPED */
      10, 10,           /* altezza max e min*/
      320, 200,         /* larghezza max e min */
      CUSTOMSCREEN
      };

#include "graphics/gfxmacros.h"

/* #include "event1.c"  */
/* carica la Routine di gestione degli eventi event1.c */

HandleEvent(code)
      LONG code;  /* fornito dal main */
```

```
{
      switch(code)
      {
      case CLOSEWINDOW:
            return(0);
            break;
      case INTUITICKS:
            movesprites();    /* 10 movimenti per secondo; test */
            default:
                  break;
      }
return(1);
}

UWORD mycolortable[] = {

      0x0000, 0x0e30, 0x0fff, 0x0b40, 0x0fb0, 0x0bf0,
      0x05d0, 0x0ed0, 0x07df, 0x069f, 0x0c0e,
      0x0f2e, 0x0feb, 0x0c98, 0x0bbb, 0x07df,

      0x0000, 0x0e30, 0x0fff, 0x0b40, 0x0fb0, 0x0bf0,
      0x05d0, 0x0ed0, 0x07df, 0x069f, 0x0c0e,
      0x0f2e, 0x0feb, 0x0c98, 0x0bbb, 0x07df
      };

      /* nero, rosso, rosso fuoco, arancione, giallo, verde limone */
      /* verde, verde acqua, blu scuro, porpora, violetto, bronzo, */
      /* marrone, grigio, blu cielo, (come sempre) */

main()
{
      struct IntuiMessage *msg;
      LONG result;
      short k, j;
      UWORD *src, *dest;      /* per copiare i dati degli sprite in RAM */

      GfxBase = OpenLibrary("graphics.library",0);

      IntuitionBase = OpenLibrary("intuition.library",0);
      /* ( si tralascia qui il controllo degli errori ) */

      s = OpenScreen(&myscreen1); /* tenta di aprire lo schermo */
      if(s == 0)
      {
            printf("Non posso aprire myscreen1\n");
            exit(10);
      }
      myWindow.Screen = s;    /* dice dove è collocato lo schermo */

      w = OpenWindow(&myWindow);
      if(w == 0)
      {
            printf("La Window non si apre!\n");
            CloseScreen(s);
            exit(20);
```

```
}
vp = &(s->ViewPort);
/* setta i colori per questa ViewPort */

LoadRGB4(vp, &mycolortable[0], 32);
rp = w->RPort;
      /* Ora attende il messaggio dall'Intuition */
      /* (il Task resta inattivo aspettando il messaggio) */

/* ************************************* */
/*    SEZIONE DEMO DEGLI SPRITE SEMPLICI    */

maxgot = 0;       /* quanti sprite abbiamo? */

spr = &sprite[0]; /* indirizzo del primo sprite */

for(k=0; k<7; k++)
{
      xmove[k]=1;
      ymove[k]=1;

      /* prende il prossimo sprite disponibile */
      sprgot[k] = GetSprite(spr,-1);

      if(sprgot[k] == -1) break;
      maxgot++;

      /* inizializza le informazioni sulle dimensioni */
      /* e sulla posizione */
      sprite[k].x = 0;
      sprite[k].y = 0;

      /* dice al sistema l'altezza dello sprite */

      sprite[k].height = 9;

      /* Alloca della RAM per mettervi i */
      /* dati dello sprite corrente */

      sprdata[k] = (UWORD *)AllocMem(44, MEMF_CHIP);

      if(sprdata[k] == NULL)
      {
            maxgot--;
            FreeSprite(sprgot[maxgot]);
            break;      /* Se non c'è abbastanza memoria blocca */
                        /* l'allocazione, ma cerca di proseguire */
                        /* lo stesso */
      }

      /* copia i dati prototipo nella RAM accessibile ai Chip */

      src = sprite_data; dest = sprdata[k]; /* source, destination */

      for( j=0; j<22; j++)
```

```
        {
                *dest++ = *src++;
        }
        /* dice al sistema di gestione degli sprite */
        /* dove trovare i dati */

        ChangeSprite(vp,spr,sprdata[k]);
        /* sceglie il punto di partenza in funzione */
        /* dello sprite in uso */

        MoveSprite(0,spr,10 + 20*sprgot[k],30);

        spr++;      /* opera sullo sprite successivo */
}

while(1)    /* "forever" */
{
        /* attende un messaggio */
        WaitPort( w->UserPort );

        /* riprende il messaggio dalla porta */
        msg = (struct IntuiMessage *)GetMsg(w->UserPort);
handleit:
        result = -1;
        if(msg != 0)
        {
                /* gestisce l'evento; controlla se è CLOSEWINDOW */
                result = HandleEvent(msg->Class);

                /* Lascia che l'Intuition riusi il messaggio */
                ReplyMsg(msg);
        }
        if(result == 0)
        {
                break; /* vi è un CLOSEWINDOW */
        }
        /* Svuota la porta prima di rimettersi in attesa */

        msg = (struct IntuiMessage *)GetMsg(w->UserPort);

        if(msg != 0)
        {
                goto handleit;
                /* 0 quando non ci sono più messaggi */
        }
}
/* Fatto! ora cancella */

/* libera gli sprite che sono stati usati */
for(k=0; k<maxgot; k++)
{
        FreeSprite(sprgot[k]);
                        /* libera gli sprite per altri usi */
        FreeMem(sprdata[k],44);
                        /* libera la memoria da essi usata */
```

```
        }

        CloseWindow(w);
        CloseScreen(s);
}
```

Listato 7.1

Sprite virtuali

Il sistema di animazione è strutturato a più livelli. Il sistema degli sprite semplici rappresenta il livello più vicino al sistema hardware, ed ha le limitazioni dell'hardware stesso. Senza l'uso di trucchi particolari, vi sono soltanto otto sprite hardware disponibili, pertanto se ne possono visualizzare soltanto otto.

La gestione degli sprite virtuali attraverso il sistema Gel risulta leggermente più complicata. Essa è tanto vicina al sistema hardware quanto lo è la gestione degli sprite semplici, per quanto riguarda il suo livello operazionale. Però, l'interfacciamento con l'hardware avviene in un modo molto diverso. Uno sprite virtuale è per definizione stessa un elemento software. Esso diventa "reale" solo quando il software di sistema lo assegna, per la visualizzazione, ad uno sprite hardware.

Invece di avere solo otto sprite sullo schermo contemporaneamente, si può arrivare ad averne 16, 24, 32 o persino 100, secondo la grandezza e la disposizione di ogni singolo sprite. E' chiaro che vi sono comunque delle limitazioni di vario genere al numero degli sprite e si dovrebbero fare delle prove per trovare tali limiti per le singole applicazioni.

Il sistema Gel sa che vi sono solo otto sprite hardware disponibili, e sa anche come debbono essere gestiti tali sprite per far si che possano essere riusati più volte in una singola schermata.

La struttura dati di uno sprite virtuale (VSprite) comprende la definizione delle posizioni dello sprite, la sua grandezza, il suo pattern in bit, e i colori che esso può utilizzare. Ogni volta che uno sprite è stato completamente visualizzato diventa libero di visualizzare una nuova immagine in una nuova posizione dello schermo che si trovi un paio di linee più in basso rispetto alla fine dell'immagine precedente e ovunque rispetto al riferimento orizzontale.

Il sistema non solo dice allo sprite hardware di ridisegnare se stesso in un'altra posizione con un diverso pattern, ma lo informa anche del nuovo set di colori da usare per l'immagine successiva, prelevando tale set dalla definizione di uno sprite virtuale.

Vantaggi dell'uso degli sprite virtuali

Il principale vantaggio dell'uso degli sprite virtuali sta nel fatto che non ci si deve preoccupare di come il sistema allochi esattamente gli sprite stessi. Questo avviene automaticamente. Si deve soltanto definire dove i propri sprite devono apparire e il sistema farà il resto. L'altro vantaggio, come già detto, consiste nel numero di sprite disponibili che, grazie alla riutilizzazione automatica del sistema, è superiore a otto.

Svantaggi nell'uso degli sprite virtuali

Se si richiede al sistema di visualizzare più di otto sprite virtuali sulla stessa linea orizzontale, alcuni degli sprite potrebbero scomparire. Questo è legato alle limitazioni hardware. Una volta che tutti gli otto sprite hardware sono occupati, non è possibile visualizzare attraverso di loro più di un set di 16 bit per linea orizzontale.Un'altro svantaggio sta nel fatto che, dopo tutto, non si hanno otto sprite hardware sui quali operare. In particolare, come è stato notato, lo sprite hardware 0 è riservato a Intuition che lo utilizza come pointer del mouse. Poiché lo sprite hardware 0 e lo sprite hardware 1 hanno lo stesso set di registri del colore, non è molto appropriato l'uso dello sprite 1 come sprite virtuale perché in questo modo si avrebbe, ogniqualvolta ad esso viene assegnata una nuova immagine con dei nuovi colori per la visualizzazione sullo schermo, un cambiamento simultaneo dei colori del pointer del mouse, cioè dello sprite 0, con un effetto fastidioso facilmente immaginabile.

Un terzo svantaggio deriva dall'area delle priorità video. Come si ricorderà, infatti, ogni sprite semplice è direttamente legato a uno sprite hardware, con la regola la priorità è inversamente proporzionale al numero che ogni sprite possiede.

Quando si usano gli sprite virtuali, il sistema assegna arbitrariamente gli sprite hardware per la visualizzazione degli sprite virtuali. Ciò implica che, se due sprite virtuali, nel loro movimento sullo schermo si trovano a condividere una posizione comune, è possibile che avvengano delle inversioni di priorità. Cioè può accadere che uno sprite che un attimo prima si trovava al di sotto di un altro sprite, improvvisamente passa in primo piano nascondendo ciò che prima era sopra di lui. Questo è un risultato della riassegnazione dinamica degli sprite operata dal sistema. Vi è un'altra cosa di cui si deve tenere conto quando si usano gli sprite virtuali: la scelta dei colori possibili per disegnare gli elementi dello sfondo. Si potrebbe voler limitare l'area del Playfield (cioè lo schermo) a un massimo di 16 colori. Oppure, si potrebbero specificare 32 colori, ma (dando per scontato che si usino gli sprite hardware 0 e 1 per evitare l'effetto sopra descritto di lampeggiamento del pointer del mouse) a causa delle restrizioni si potrebbero di fatto usare solo i colori 0-20, 24, e 28. Cioè, in altre parole, non si potrebbero usare i colori 21-23, 25-27, e 29-31.

Infatti, quando il sistema Gel assegna degli sprite virtuali agli sprite hardware, esso assegna anche un particolare colore dello sprite virtuale a un particolare registro di colore dello sprite hardware. Pertanto, ciò che è stato disegnato mediante quel colore verrà cambiato in funzione del colore correntemente assegnato a quello Sprite. Il programma makevsprite mostra con dei frammenti di testo come avvenga il riassegnamento dei registri di colore.

Inizializzare il sistema Gel

Per utilizzare gli sprite virtuali, o qualsiasi altra parte del sistema Gel, si deve per prima cosa inizializzare il sistema stesso. Una routine di nome ReadyGels può essere usata per questo scopo. Essa è mostrata nel listato 7.2. Si deve anche dichiarare una struttura dati di nome GelsInfo. Essa contiene i dati necessari al sistema Gel per tenere traccia sia degli sprite virtuali sia dei Bobs. Una volta aperto uno schermo, si può lanciare ReadyGels.

SpriteHead e SpriteTail. Nel listato 7.2 è possibile trovare i termini SpriteHead e SpriteTail. Il sistema costruisce una lista di tutti i Gels (Graphics element) presenti nel sistema stesso e della loro sequenza di visualizzazione. I Gel vengono ordinati dall'alto in basso e da destra a sinistra. SpriteHead e SpriteTail sono le due terminazioni della lista dei Gels.

sprite **riservati.** Si può dire al sistema quali sprite hardware deve usare per visualizzare gli sprite virtuali. Settando dei bit particolari a 1, si da al sistema un OK perché operi una riassegnazione dinamica dei registri di colore per uno sprite affinché sia uno sprite virtuale. I bit per i parametri di sprite riservato (sprRsrvd) sono numerati come gli sprite stessi. In altre parole, lo sprite 7 sarà rappresentato dal bit 7, lo sprite 6 dal bit 6, e così via.

La routine ReadyGels nel listato utilizza un pattern di bit settato a 0xfc, che ha un valore binario di 11111100. Ciò significa "non usare gli sprite 0 e 1". Ciò che si può voler fare è cambiare sprRsrvd per operare sul corrispondente oggetto nel sistema degli sprite semplici, localizzato in GfxBase->SpriteReserved. Infatti, ogniqualvolta il sistema degli sprite semplici alloca uno sprite per l'uso di un task, esso setta un bit di SpriteReserved in GfxBase. Se un bit si trova a uno in quella variabile, il corrispondente bit sarà impostato a zero.

Ecco una sequenza che fornisce il risultato desiderato:

```
struct GfxBase *GfxBase;
/* usato in alternativa alla dichiarazione */
/* LONG GfxBase */
/* già utilizzata in questo libro */

/* ...poi in ReadyGels, al posto di g->sprRsrvd = 0xFC */

g->sprRsrvd = 0xFC & (!(GfxBase->SpriteReserved));

/* setta a 0 ogni Bit per il quale vi sia un Bit a 1 */
/* nel sistema degli sprite semplici */
```

```
/* readygels.c */

struct VSprite *SpriteHead = NULL;
struct VSprite *SpriteTail = NULL;

void border_dummy()     /* una finta Routine di collisione */
```

```
{
   return;
}

ReadyGels(g, r)
struct RastPort *r;
struct GelsInfo *g;
{
   /* alloca l'inizio e la fine della lista */
   if ((SpriteHead = (struct VSprite *)AllocMem(sizeof
         (struct VSprite), MEMF_PUBLIC | MEMF_CLEAR)) == 0)
   {
      return(-1);
   }

   if ((SpriteTail = (struct VSprite *)AllocMem(sizeof
         (struct VSprite), MEMF_PUBLIC | MEMF_CLEAR)) == 0)
   {
      FreeMem(SpriteHead, sizeof(struct VSprite));
      return(-2);
   }
   g->sprRsrvd = 0xFC;  /* Non usare gli sprite 0 e 1. */

   if ((g->nextLine = (WORD *)AllocMem(sizeof(WORD) * 8,
         MEMF_PUBLIC | MEMF_CLEAR)) == NULL)
   {
      FreeMem(SpriteHead, sizeof(struct VSprite));
      FreeMem(SpriteTail, sizeof(struct VSprite));
      return(-3);
   }

  if ((g->lastColor = (WORD **)AllocMem(sizeof(LONG) * 8,
         MEMF_PUBLIC | MEMF_CLEAR)) == NULL)
   {
      FreeMem(g->nextLine,8 * sizeof(WORD));
      FreeMem(SpriteHead, sizeof(struct VSprite));
      FreeMem(SpriteTail, sizeof(struct VSprite));
      return(-4);
   }

  /* occorre poi preparare una tabella di puntatori per le Routine */
  /* che dovranno essere usate quando Docollision rileverà una */
  /* collisione. Questa dichiarazione potrebbe non essere */
  /* necessaria per un VSprite di base che non possiede delle */
  /* implementazioni di direzioni di collisione, ma viene fatto */
  /* per completare l'esempio */

   if ((g->collHandler = (struct collTable *)AllocMem(sizeof(struct
         collTable), MEMF_PUBLIC | MEMF_CLEAR)) == NULL)
   {
      FreeMem(g->lastColor, 8 * sizeof(LONG));
      FreeMem(g->nextLine,8 * sizeof(WORD));
      FreeMem(SpriteHead, sizeof(struct VSprite));
      FreeMem(SpriteTail, sizeof(struct VSprite));
      return(-5);
```

```
    }

    /* quando una qualsiasi parte dell'oggetto tocca o oltrepassa */
    /* questo confine, causerà il richiamo della Routine di */
    /* collisione col confine. Essa risiede in smash[0] nella */
    /* tabella di gestione delle collisioni e viene richiamata solo */
    /* se viene richiamata Docollisione */

    g->leftmost = 0;
    g->rightmost = r->BitMap->BytesPerRow * 8 - 1;
    g->topmost = 0;
    g->bottommost = r->BitMap->Rows - 1;

    r->GelsInfo = g;  /* Linka le due strutture */

    InitGels(SpriteHead, SpriteTail, g );

   /* Puntatori inizializzati per lo sprite finto */
   /* usati dal sistema per tenere traccia dell'animazione */

    SetCollision(0, border_dummy, g);
    WaitTOF();
    return(0);      /* se viene ritornato 0 è tutto OK,
                    /* ogni valore negativo avvertirà del fallimento */
                    /* Si noterà dalla struttura della Routine */
                    /* che i fallimenti possono avvenire per */
                    /* un Out of Memory */

}
```

Listato 7.2

Questa tecnica è necessaria per il fatto che il sistema degli sprite semplici fu sviluppato indipendentemente dal sistema Gel. Anche nella versione 1.2 del sistema operativo essi rimangono indipendenti l'uno dall'altro.

NextLines e LastColors. Il sistema necessita di alcune variabili perché possa essere abilitato a decidere quale sprite hardware deve usare per la visualizzazione del successivo sprite virtuale: questi valori sono dei parametri all'interno della struttura GelsInfo e includono un Array di "Next Lines" e di "Last Colors".

L'Array NextLines è utilizzato per dare informazioni al sistema circa il numero di linee presenti nello schermo nel quale ogni sprite hardware dovrà essere disponibile per la visualizzazione di uno sprite virtuale.

Nell'Array di puntatori LastColor, il sistema immagazzina un puntatore alla più recente definizione di colore usata. I colori dello sprite virtuale vengono scritti all'interno del set dei registri di colore dello sprite hardware al quale lo sprite virtuale è assegnato. Questo array contiene, per ogni sprite hardware, un puntatore all'ultimo set di tre colori (dal puntatore sprColor della struttura VSprite).

Quando il sistema sta operando una scansione per decidere quale sprite hard ware usare per rappresentare uno sprite virtuale, esso controlla il contenuto dell'Array lastcolor. Se uno sprite hardware è disponibile e gli è stato assegnato questo Set di colori, non è necessario nessun ulteriore assegnamento, e pertanto non verrà generata nessuna istruzione di cambiamento dei colori dalla Copper list - una lista di istruzioni del coprocessore. (Si veda l'Amiga hardware Manual per informazioni riguardanti il coprocessore Copper).

Se tutti gli sprite virtuali usano un diverso set di colori (cioè se i puntatori asprColors sono differenti per tutti gli Sprite), allora si è limitati all'uso di quattro sprite virtuali per ogni linea orizzontale. Se, invece, si definiscono otto sprite virtuali, con 0 e 1 aventi lo stesso colore, con 2 e 3 aventi lo stesso colore, lo stesso per 4, 5 e per 6 e 7, allora si potranno avere sino a otto sprite virtuali sulla stessa linea orizzontale.

Poiché il sistema hardware distribuisce i registri di colore tra coppie di sprite hardware, nel caso sopra menzionato esso potrà assegnare otto sprite virtuali agli sprite hardware perché vi sono solo quattro set di colori per otto sprite Virtuali, esattamente tanti quanti possono essere gestiti al massimo dalle risorse del sistema.(Si noti che lastcolor non viene usato per i Bobs, ma solo per gli Sprite).

La routine MakeVSprite

Il listato 7.3 contiene la routine MakeVSprite, la quale è di aiuto nella definizione di uno sprite virtuale. Nonostante essa crei uno sprite virtuale, essa non lo aggiunge alla lista di sistema in modo che possa essere visualizzato. Questa operazione deve essere fatta dal programma richiamante. Se MakeVSprite restituisce zero, significa che non c'era abbastanza memoria per creare lo sprite virtuale. Un valore diverso da zero rappresenterà l'indirizzo del nuovo sprite virtuale che può essere aggiunto al sistema mediante AddSprite.

La struttura VSprite

La routine MakeVSprite utilizza i parametri della struttura VSprite. Questa struttura viene usata per definire sia gli sprite virtuali sia i Bobs. Una struttura dati VSprite viene allocata dinamicamente, e le variabili fornite dal programmatore vengono copiate in essa. La struttura inoltre alloca alcune maschere e variabili di collisione, dando per scontato che sia presente una routine di collisione con i confini dell'area (chiamata border_dummy nel listato). Ciò che segue contiene la descrizione dei vari parametri della struttura VSprite.

Height. Il parametro di altezza specifica quanto sia alto in linee lo sprite considerato. Si può specificare un numero qualsiasi di linee. La definizione in bit dello sprite deve avere lo stesso numero di linee. (Anche per i Bobs Height può essere un numero qualsiasi. Più alto è il Bob più tempo impiegherà il sistema per disegnarlo).

```
/* MakeVSprite.c */

struct VSprite *MakeVSprite(lineheight, image, colorset, x, y,
                        wordwidth, imagedepth, flags)
SHORT lineheight; /*Quant'è alto questo VSprite? */
WORD *image;            /* La definizione dei dati dello Sprite, deve */
                        /* contenere il doppio del valore di Height in */
                        /* word */
WORD *colorset;    /* localizzazione delle tre word che */
                        /* descrivono i colori di questo sprite */
SHORT x, y;        /* posizione iniziale sullo schermo */
SHORT wordwidth, imagedepth, flags;
{
   struct VSprite *v;   /* crea un puntatore alla struttura */
                              /* VSprite allocata dinamicamente */
                              /* da questa Routine */

   if ((v = (struct VSprite *)AllocMem(sizeof(struct VSprite),
         MEMF_PUBLIC | MEMF_CLEAR)) == 0)
   {
      return(0);
   }

   v->Flags = flags;    /* VSprite o Bob? */

   v->Y = y;            /* Stabilisce la posizione iniziale rispetto */
   v->X = x;            /* alle coordinate di visualizzazione */

   v->Height = lineheight;    /* Viene richiesta l'altezza */
   v->Width = wordwidth;      /* Un VSprite è sempre largo una word */
```

```
 /* Vi sono due tipi di profondità... la profondità */
 /* dell'immagine stessa e la profondità dello sfondo */
 /* sul quale sarà disegnata. La profondità dell'immagine */
 /* informa circa lo spazio occupato dall'immagine stessa in */
 /* memoria se essa viene allocata dinamicamente. La profondità */
 /* del Playfield dice quanta memoria sarà necessaria per */
 /* salvare lo sfondo e ripristinarlo quando viene disegnato */
 /* un Bob. Un vSprite è sempre profondo 2 piani di Bit */
 /* a meno che non sia parte di un Bob */

v->Depth = imagedepth;

 /* Si da per scontato che tutti i programmi richiamanti abbiano */
 /* una Routine di collisione con i confini... il Bit 1 di questa */
 /* maschera è riservato alle collisioni con i confini rilevate */
 /* in DoCollision(). Le uniche collisioni qui rilevate saranno */
 /* quelle con i confini. Questa cosa può essere cambiata */

  v->MeMask = 1;
  v->HitMask = 1;

  v->ImageData = image;
           /* Il richiamante dice dove trovare i dati dell'immagine */

 /* Mostra al sistema dove trovare la maschera che è una versione */
 /* schiacciata del VSprite */

  if ((v->BorderLine = (WORD *)AllocMem((sizeof(WORD)*wordwidth),
        MEMF_PUBLIC | MEMF_CLEAR)) == 0)
  {
   FreeMem(v, sizeof(struct VSprite));
   return(0);
  }

 /* Mostra al sistema dove trovare la maschera che contiene */
 /* le posizioni dei Bit settati a uno */

  if ((v->CollMask = (WORD *)AllocMem(sizeof(WORD)*lineheight*wordwidth,
        MEMF_CHIP | MEMF_CLEAR)) == 0)
  {
     FreeMem(v, sizeof(struct VSprite));
     FreeMem(v->BorderLine, wordwidth * sizeof(WORD));
       return(0);
  }

 /* Usato solo per i VSprite, non per i Bob. E'il luogo dove il */
 /* richiamante dice dove si trovano i colori del VSprite */
 /* trovare
  v->SprColors = colorset;

  /* non usate per il VSprite, MakeBob le setterà per il Bob. */
  v->PlanePick  = 0x00;
  v->PlaneOnOff = 0x00;
```

```
    InitMasks(v);  /* crea la maschera di collisione e i confini */
    return(v);
}

/* fine di MakeVSprite.c */
```

Listato 7.3

L'immagine. Il parametro dell'immagine è un puntatore all'Array di word che definiscono l'aspetto reale dello Sprite. L'immagine dello sprite mostrata nel listato 7.1 è un esempio di tipica definizione di immagine. L'indirizzo di partenza dell'immagine è la locazione della linea 1 dei dati dell'immagine stessa.

A differenza degli sprite semplici, gli sprite virtuali non necessitano di un rivisitazione dei dati dell'immagine dello sprite per ogni riproduzione dell'immagine dello sprite stesso. Comunque, come per gli sprite semplici, i dati dell'immagine dello sprite devono trovarsi nella RAM accessibile ai Chip Custom.

I colori. Il parametro colorset è un puntatore a un set di tre word (48 byte) che definisce i colori che devono essere usati per visualizzare l'oggetto grafico. Queste word contengono i dati di colore per i colori numero 1, 2 dello sprite virtuale.

Il sistema tiene traccia, nell'Array lastcolors, del valore del puntatore usato per assegnare i colori a uno sprite particolare. Se si hanno parecchi sprite virtuali con colori uguali, il valore del puntatore sarà uguale per tutti. Quando il sistema sta cercando uno sprite hardware da assegnare a uno sprite virtuale, esso sa che vi sono coppie di sprite che condividono gli stessi registri di colore. Se si caricano gli stessi colori per ognuno degli sprite di una coppia, è molto più facile per il sistema trovare uno sprite disponibile da usare. Se tutti i valori del puntatore di colore sono differenti, si aumentano le possibilità che il sistema superi il numero di sprite hardware disponibili e che, pertanto, decida di non visualizzare alcuni degli sprite virtuali in quella particolare scansione video.

(Le decisioni su ciò che può essere visualizzato vengono prese ogni sessantesimo di secondo).

Posizione. I parametri x e y determinano la posizione dell'oggetto grafico sullo schermo. Diversamente dagli sprite semplici che possono essere posizionati relativamente all'area globale di visualizzazione, gli sprite virtuali hanno la restrizione di poter essere posizionati solo in relazione allo schermo nel quale vengono disegnati. Però, come gli sprite semplici, essi non interferiscono con il Playfield.

Larghezza. Il parametro widthword specifica la larghezza dell'oggetto grafico. Per uno sprite virtuale,questo valore è sempre 1. (Un Bob può avere qualunque larghezza; più largo è il Bob, più tempo sarà necessario per visualizzarlo).

Profondità. Il parametro imagedepth definisce il numero di piani di dati che sono contenuti nell'area dell'immagine. Per uno sprite virtuale, questo valore è sempre 2. (Per i Bob la profondità può assumere qualunque valore purché sia minore o uguale a quello dello schermo nel quale viene disegnato).

I Flag. I Flag identificano il tipo di oggetto grafico che viene rappresentato dalla struttura dati VSprite. Per uno sprite virtuale, si setti tale parametro a VSPRITE. (Lo si setti a 0 per un Bob).

Il programma Virtual Sprite

Il listato 7.4 è un programma che utilizza le routine che sono state create per duplicare alcune delle caratteristiche del programma Simple sprite precedentemente mostrato. La principale differenza tra i due programmi sta nel fatto che qui vengono creati molti sprite virtuali e, invece di rifarsi agli INTUITICK, questo programma attende l'inizio della videata successiva prima di generare un nuovo movimento dello sprite. Pertanto, gli sprite di questo esempio si muovono circa sei volte più velocemente di quelli dell'esempio precedente.

Il posizionamento iniziale degli sprite è casuale e, inoltre, sono stati creati abbastanza sprite da garantire che uno o due di essi scompaiano per una videata o due. Si guardi attentamente lo schermo per poter notare questa cosa.

Si notino anche le variazioni occasionali di priorità dei vari Sprite, e il cambiamento dei colori 21-23, 25-27 e 29-31 durante il movimento degli sprite sullo schermo.

```
#include "exec/types.h"
#include "intuition/intuition.h"
#include "graphics/sprite.h"
#include "exec/memory.h"
#include "graphics/gels.h"

/* dice al sistema di creare e gestire MAXSP VSprite */
#define MAXSP 28

/* definisce le possibili velocità dei VSprite in unità VBlank */

SHORT speed[] = { 1, 2, -1, -2 };

SHORT xmove[MAXSP], ymove[MAXSP];
                    /* direzioni di movimento del sprite */
struct VSprite *vsprite[MAXSP];     /* MAXSP sprite semplici */
struct VSprite *vspr;          /* puntatore a uno sprite */
short maxgot;                  /* # massimo di sprite creati */

struct GelsInfo mygelsinfo;    /* La RastPort della Window necessita */
                               /* di una di esse per creare lo sprite */
struct Window *w; /* puntatore a una Window */
struct RastPort *rp;    /* puntatore a una RastPort*/
struct Screen *s; /* un puntatore a uno schermo */
struct ViewPort *vp;    /* puntatore a una ViewPort */

struct Window *OpenWindow();
struct Screen *OpenScreen();
LONG GfxBase;
LONG IntuitionBase;

/* 18 word di dati = 9 linee di dati per lo sprite */

UWORD sprite_data[ ] = {
            0x0fc3, 0x0000, /* linea 1 dei dati dell'immagine */
            0x3ff3, 0x0000, /* linea 2 dei dati dell'immagine */
            0x30c3, 0x0000, /* linea 3 dei dati dell'immagine */
            0x0000, 0x3c03, /* linea 4 dei dati dell'immagine */
            0x0000, 0x3fc3, /* linea 5 dei dati dell'immagine */
            0x0000, 0x03c3, /* linea 6 dei dati dell'immagine */
            0xc033, 0xc033, /* linea 7 dei dati dell'immagine */
```

```
            0xffc0, 0xffc0, /* linea 8 dei dati dell'immagine */
            0x3f03, 0x3f03, /* linea 9 dei dati dell'immagine */
      };
UWORD *sprdata;

movesprites()
{
      short i;

      for (i=0; i<maxgot; i++)
      {
            vspr = vsprite[i];

            vspr->X = xmove[i]+vspr->X;
            vspr->Y = ymove[i]+vspr->Y;

            /* muove gli sprite...ora */

            if(vspr->X >= 300 || vspr->X <= 0) xmove[i]=-xmove[i];
            if(vspr->Y >= 190 || vspr->Y <= 0) ymove[i]=-ymove[i];

      }
      SortGList(rp);    /* prende la lista in ordine */
      DrawGList(rp, vp);      /* crea le istruzioni dello sprite */

      MakeScreen(s);
                  /* chiede all'Intuition di mettere tutto insieme */
      RethinkDisplay(); /* e di mostrarci cosa abbiamo */
}

#define AZCOLOR 1
#define WHITECOLOR 2

#define WC WINDOWCLOSE
#define WS WINDOWSIZING
#define WDP WINDOWDEPTH
#define WDR WINDOWDRAG

#define NORMALFLAGS (WC|WS|WDP)
/* Non usa WindowDrag perché non vuole che sia mosso lo schermo */

/* Permette il dimensionamento della Window in modo che l'utente */
/* possa rimpicciolirla e poi riingrandirla, per cancellare il */
/* testo di fondo. Si potrà facilmente notare che, quando vi sono */
/* molti sprite presenti, alcuni scompaiono perché il sistema */
/* non ha più sprite disponibili */

/* myfont1 specifica le caratteristiche del Font di Default; */
/* qui abbiamo un Font da 80 colonne visualizzato a 40 */
/* a causa del a bassa risoluzione */

struct TextAttr myfont1 = { "topaz.font", 8, 0, 0 };

struct NewScreen myscreen1 = {
      0, 0,       /* LeftEdge, TopEdge ... dove posizionare lo schermo */
```

```
      320, 200,   /* Width, Height ... dimensiono dello schermo */
      5,          /* 5 piani di Bit, 32 colori tra i quali scegliere */
      1, 0,       /* DetailPen, BlockPen */
      SPRITES,    /* modo di visualizzazione, 0 = bassa risoluzione */
      CUSTOMSCREEN,     /* tipo di schermo */
      &myfont1,   /* Font di Default per lo schermo */
      "32 Color Test", /* Nome dello schermo*/
      NULL, /* Gadget, tutti NULL, ignorati */
      NULL };
                  /* indirizzo della BitMap personalizzata */
                  /* non presente qui */

struct NewWindow myWindow = {
      0,          /* margine sinistro della Window, in pixel */
                  /* dell'angolo sinistro dello schermo */
      0,          /* estremità superiore della Window in pixel dalla */
                  /* estremità superiore dello schermo attuale */
      320, 185,   /* larghezza e altezza della Window */
      0,          /* DetailPen - colore usato per i bordi della */
                        /* Window */
      1,          /* BlockPen - colore usato per i gadget della Window */
                  /* per entrambe le penne il valore 1 significa */
                  /* che vengono usate quelle di Default */
      CLOSEWINDOW,      /* il programma Simple sprite usava anche */
                        /* INTUITICKS */
                        /* Flag IDCMP */
      NORMALFLAGS | GIMMEZEROZERO | ACTIVATE,
                  /* Flag della Window */
      NULL,       /* primo Gadget */
      NULL,       /* CheckMark  */
      "Click Close Gadget To Stop", /* nome della Window */
      NULL,       /* puntatore a uno schermo se non si usa */
                        /* il Workbench */
      NULL,       /* puntatore a una BitMap se si tratta di una */
                        /* Window SUPERBITMAP */
      320, 10,    /* larghezza e altezza minime */
      320, 200,   /* larghezza e altezza massime */
      CUSTOMSCREEN
      };

#include "graphics/gfxmacros.h"

/* #include "event1.c"  */
/* carica il gestore di eventi */
/* event1.c */

HandleEvent(code)
      LONG code;  /* fornito dal main */
{
      switch(code)
      {
      case CLOSEWINDOW:
            return(0);
            break;
      case INTUITICKS:
```

```
            movesprites();    /* 10 movimenti per secondo; test */
            default:
                  break;
      }
return(1);
}

UWORD mycolortable[] = {

      0x0000, 0x0e30, 0x0fff, 0x0b40, 0x0fb0, 0x0bf0,
      0x05d0, 0x0ed0, 0x07df, 0x069f, 0x0c0e,
      0x0f2e, 0x0feb, 0x0c98, 0x0bbb, 0x07df,

      0x0000, 0x0e30, 0x0fff, 0x0b40,
      0x0fb0, 0x0bf0, 0x05d0, 0x0ed0,
      0x07df, 0x069f, 0x0c0e, 0x0f2e,
      0x0feb, 0x0c98, 0x0bbb, 0x07df
      };

      /* nero, rosso, bianco, rosso fuoco, arancione, giallo */
      /* verde limone, verde, verde acqua, blu scuro, porpora */
      /* violetto, bronzo, marrone grigio, blu cielo (come sempre) */

UWORD colorset0[ ] = { 0x0e30, 0xffff, 0x0b40 };
                                    /* come per i colori 17-19 */
UWORD colorset1[ ] = { 0x0bf0, 0x05d0, 0x0ed0 }; /* 21-23 */
UWORD colorset2[ ] = { 0x069f, 0x0c0e, 0x0f2e }; /* 25-27 */
UWORD colorset3[ ] = { 0x0c98, 0x0bbb, 0x07df }; /* 29-31 */
UWORD *colorset[ ] = {
                  colorset0, colorset1,
                  colorset2, colorset3 };
int choice;
char *numbers[] = { "17","18","19",
                "20","21","22","23",
                "24","25","26","27",
                "28","29","30","31" };

#include "purgegels.c"
#include "readygels.c"
#include "makevsprite.c"

main()
{
      struct IntuiMessage *msg;
      LONG result;
      SHORT k, j, x, y, error;
      UWORD *src, *dest;
            /* per copiare i dati degli sprite nella RAM */

      GfxBase = OpenLibrary("graphics.library",0);

      IntuitionBase = OpenLibrary("intuition.library",0);
      /* tralasciato il controllo degli errori per brevità */

      s = OpenScreen(&myscreen1);  /* tenta di aprirlo */
```

```
if(s == 0)
{
      printf("Non posso aprire myscreen1\n");
      exit(10);
}
myWindow.Screen = s;    /* localizzazione dello schermo */

ShowTitle(s, FALSE);
      /* impedisce l'abbassamento dello schermo...*/

w = OpenWindow(&myWindow);
if(w == 0)
{
      printf("La Window non si è aperta!\n");
      CloseScreen(s);
      exit(20);
}
vp = &(s->ViewPort);
/* setta i colori per la ViewPort */

LoadRGB4(vp, &mycolortable[0], 32);
rp = w->RPort;
      /* Ora il Task attende un messaggio dall'Intuition e */
      /* diviene inattivo durante l'attesa */

/* Scrive un testo con i colori degli sprite in modo */
/* da mostrare cosa succeda ai colori stessi */
/* quando si abbassa lo schermo */
/* si dovrebbe evitare l'uso dei registri di colore */
/* utilizzati dagli sprite */

/* Si noti anche che i colori 17-19 non vengono toccati */
/* questo a causa del valore sprRsrvd=0xFC in ReadyGels */
/* esso impedisce l'uso degli sprite 0 e 1 */
/* come sprite virtuali */

for(j=8; j<180; j+=50)
{
      for(k=0; k<15; k++)
      {
            Move(rp,k*20,j);
            SetAPen(rp,k+17); /* mostra 17-31 */
            /* 16, 20, 24, 28 sono ignorati dai VSPRITES
             * perché non sono utilizzati dagli sprite hardware */

            Text(rp,numbers[k],2);
      }
}
/* ************************************ */
/*              VSPRITE DEMO            */
/* ************************************ */

/* Alloca della memoria accessibile ai Chip Custom */
/* per i dati dello sprite attuale */
/* è necessario in caso di macchine con espansione */
```

```
sprdata = (UWORD *)AllocMem(36, MEMF_CHIP);

if(sprdata == NULL)
{
      /* non c'è abbastanza memoria per lo sprite */
}
/* ora copia i dati dello sprite nella Ram prima allocata */

src = sprite_data;
dest = sprdata;   /* sorgente, destinazione */

for( j=0; j<18; j++)
{
      *dest++ = *src++;
}

choice = 0;
maxgot = 0;

/* Prepara il sistema GELS a lavorare con un VSprite o un Bob */

error = ReadyGels(&mygelsinfo, rp);

for(k=0; k<MAXSP; k++)  /* qualunque numero di VSprite */
{
      xmove[k]=speed[RangeRand(4)];
      ymove[k]=speed[RangeRand(4)];

      /* fissa una posizione per lo sprite */
      x = 10 + RangeRand(280);
      y = 10 + RangeRand(170);

      /* crea un VSprite */
      vsprite[k] = MakeVSprite( 9, sprdata, colorset[choice],
                        x, y, 1, 2, VSPRITE);
      /* alto 9 linee, usa un'immagine di dati MEMF_CHIP */
      /* per il VSprite, con un particolare set di colori, */
      /* a una locazione X,Y. Largo 1 word, e profondo 2 piani */
      /* (come tutti i VSprite). Specifica che è un VSPRITE */

      if(vsprite[k] == 0)
      {
            break;      /* out of memory! */
      }
      AddVSprite(vsprite[k], rp);

      maxgot++;

      choice++; /* sceglie un nuovo set di colori */
      if(choice >= 4)
      {
            choice = 0;  /* ricomincia da capo con i set */
      }
```

```
    }
    while(1)    /* per sempre */
    {
        WaitTOF();
        movesprites();

        result = -1;
            /* ora guarda se un CLOSEWINDOW è in attesa*/
        msg = (struct IntuiMessage *)GetMsg(w->UserPort);
        if(msg != 0)
        {
            result = HandleEvent(msg->Class);

            /* lascia che l'Intuition riutilizzi il messaggio */
            ReplyMsg(msg);
        }
        if(result == 0)
        {
            break; /* è arrivato un CLOSEWINDOW */
        }
    }

    /* Fatto! Ora cancella */

    /* Libera tutti gli sprite virtuali che ha creato */

    for(k=0; k<maxgot; k++)
    {
        DeleteGel(vsprite[k]);
    }
    /* cancella ciò che ReadyGel ha creato */
    PurgeGels(&mygelsinfo);

    FreeMem(sprdata, 36);
                    /*libera tutto quello che ha allocato */

    CloseWindow(w);
    CloseScreen(s);
}
```

Listato 7.4

Queste figure vengono disegnate nell'area di disegno con i colori sopra menzionati, per mostrare come sta operando il sistema Gel con i registri di tali colori durante lo spostamento degli sprite.

Nota: il programma del listato 7.4 funziona correttamente solo con la versione 1.2 del sistema operativo. Vi era un problema nel sistema Gel che è stato risolto con la versione 1.2. Mentre per le funzioni dei Bob non vi sono problemi seri, gli sprite virtuali hanno dei grossi problemi con la versione 1.1.

I Blitter Object (BOB)

L'argomento seguente da trattare, per quanto concerne l'animazione, sono i Blitter Object, anche chiamati Bob. La struttura dati Bob definisce tali oggetti. Come gli sprite virtuali, i Bob fanno parte del sistema Gel. Diversamente dagli sprite virtuali, che sono indipendenti dallo sfondo sul quale si muovono, i Bob sono disegnati come una parte dell'area di sfondo. Si può usare un Bob come pennello per un programma di Paint, disegnarlo ovunque sull'area di disegno con il numero di colori desiderato (Il numero di colori del Bob è lo stesso dell'area sul quale lo si disegna). Si può muovere un Bob forzandolo a salvare l'area ricoperta dal suo rettangolo di contenimento prima che esso sia effettivamente disegnato, e ripristinando tale area una volta che il Bob si è spostato.

La struttura dati Bob contiene dei campi che descrivono l'aspetto del Bob e il suo comportamento. La struttura contiene anche un puntatore a una struttura VSprite, che contiene il resto della definizione dei dati. Il fatto di avere una struttura VSprite come parte della definizione globale di un Bob, permise ai progettisti del sistema di utilizzare una sola routine per tenere traccia delle posizioni di tutti gli elementi grafici come se facessero parte di una sola animazione.

La routine MakeBob

Il listato 7.5 contiene la routine MakeBob. La maggior parte dei parametri di MakeBob sono uguali a quelli di MakeVSprite. Tra quelli che invece differiscono vi sono il puntatore all'immagine e i parametri PlanePick e PlaneOnOff.

Il puntatore all'immagine. Si ricorda che ogni linea di uno sprite è formata da una coppia di word che si trovano in locazioni di memoria consecutive, per esempio:

```
0x0fc3,0x0000,
   /* linea 1 dei dati dell'immagine di uno sprite */
```

Per un Bob, i dati sono differenti. Tutti i dati che riguardano un particolare piano di bit sono raggruppati insieme. Quando si specificano i dati per un Bob, è più piacevole esteticamente raggruppare i dati in modo da ricostruire la figura reale

del Bob in modo che i Pattern di bit siano qualcosa di riconoscibile. La differenza tra la definizione dei colori di un Bob e di uno Sprite, sta nel fatto che i bit che si combinano per specificare i colori del Bob si trovano in posizioni corrispondenti all'interno delle linee di definizione del Pattern in ognuno dei piani di bit del Pattern stesso. Ecco un esempio con un Bob largo 32 bit:

```
/* piano 0 del Pattern */
0x0000,0x1111,
0xcccc,0xeeee,
/* piano 1 del Pattern */
0xffff,0x0000,
0xaaaa,0x7777
```

```
/* MakeBob.c */

struct Bob *MakeBob(bitwidth,lineheight,imagedepth,image,
      planePick,planeOnOff, x,y, flags)
SHORT bitwidth,lineheight,imagedepth,planePick,planeOnOff,x,y,flags;
WORD *image;
{
   struct Bob *b;
   struct VSprite *v;
   SHORT wordwidth;

   wordwidth = (bitwidth+15)/16;

   /* crea un VSprite per questo Bob, bisognerà deallocarlo */
   /* (liberarlo) quando si cancellerà questo Bob */
   /* Si noti che non vi sono set di colori per un Bob */

   if ((v = MakeVSprite(lineheight, image, NULL, x, y, wordwidth,
         imagedepth, flags)) == 0)
   {
      return(0);  /* non c'è abbastanza memoria per il VSprite */
   }

   /* Il richiamante seleziona i piani di Bit nei quali */
   /* è disegnata  l'immagine */
   v->PlanePick = planePick;

   /* Cosa accade ai piani all'interno dei quali l'immagine non è */
   /* disegnata */
   v->PlaneOnOff = planeOnOff;

   if ((b = (struct Bob *)AllocMem(sizeof(struct Bob),
         MEMF_PUBLIC | MEMF_CLEAR)) == 0)
   {
      return(0);  /* non c'è abbastanza memoria per il Bob */
```

```
    }

    v->VSBob = b; /* Linka il Bob e le sue strutture VSprite */

    b->Flags = 0; /* Non è parte di una animazione (BOBISCOMP) */
    /* e non è  necessario mantenere l'immagine dopo lo */
    /* spostamento del Bob */

   /* Dice dove salvare lo sfondo. Deve esserci uno spazio */
   /* proporzionato alla profondità di ciò che è disegnato */

    if ((b->SaveBuffer = (WORD *)AllocMem(sizeof(SHORT) * wordwidth
          * lineheight * imagedepth, MEMF_CHIP | MEMF_CLEAR)) == 0)
    {
       FreeMem(b, sizeof(struct Bob));
       return(0);
    }

    b->ImageShadow = v->CollMask;

   /* Le priorità sono organizzate in modo inversamente */
   /* proporzionale all'ordine di definizione */

    b->Before = NULL;
    /* Lascia che sia il richiamante a preoccuparsi delle priorità */
    b->After = NULL;

    b->BobVSprite = v;

   /* InitMask non setta ImageShadow. Si può creare la propria */
   /* versione di ombreggiatura o usare collMask */


    b->BobComp = NULL;   /* non è parte di una animazione */
    b->DBuffer = NULL;   /* non è doppiamente bufferizzato */

   /* ritorna un puntatore al Bob appena creato per chiamate */
   /* successive o per AddBob(b) */
    return(b);
}
/* fine di makebob.c */
```

Listato 7.5

Il colore del pixel nell'angolo superiore sinistro di questo Bob è determinato dalla combinazione dei bit prelevati dai bit più significativi delle word 0000 e 1111. I bit del piano avente numero maggiore hanno maggior significato, in altre parole, ecco le possibili selezioni di colore ottenibili con i bit dei piani 1 e 0:

```
Piano di Bit 1:                    0 0 1 1

Piano di Bit 0:                    0 1 0 1

Numero del colore selezionato:     0 1 2 3
```

PlanePick e PlaneOnOff. I valori binari dei bit che selezionano i numeri dei colori sono soggetti a modificazioni mediante i valori di PlanePick e PlaneOnOff. Questi parametri permettono al programmatore a di definire un oggetto avente solo quattro colori. Per esempio, nonostante l'oggetto qui descritto abbia solo quattro colori (infatti l'oggetto è definito mediante due piani), si può scegliere quali siano i quattro colori da usare - tra i 32 disponibili in questo schermo - ogni volta che l'oggetto viene disegnato.

PlanePick è un bit che, una volta settato, dice al sistema : "Questi piani faranno parte del Pattern, a partire dal piano inferiore di bit. Disegna i bit prelevandoli dal piano di bit avente numero minimo e ponendoli nel BitPlane del Pattern avente numero minimo, purchè in tale BitPlane sia settato a 1 il bit PlanePick. Disegna i bit prelevandoli dal piano successivo e ponendoli nel BitPlane successivo per il quale sia settato a 1 il Bit PlanePick, e così via".

Per esempio, se PlanePick ha un valore binario 0101, allora significa:

```
Piano coinvolto:                          3 2 1 0

Valore di PlanePick:                      0 1 0 1

Usa i dati sorgente dai piani di immagine:  X 1 X 0
```

Pertanto i dati dell'immagine contenuti nel piano 0 vengono scritti nel piano 0 dell'area di destinazione, e i dati dell'immagine contenuti nel piano 1 vengono scritti nel piano 2 dell'area di destinazione.

In questo schema vi sono due piani dell'area di destinazione nei quali non viene scritto alcun dato: i piani 1 e 3. Cosa bisogna fare con i dati che in essi già si trovano?.

Ecco che interviene PlaneOnOff. Anche PlaneOnOff è un valore binario. Esso definisce, per i piani non considerati con PickPlane, cosa si debba fare dei dati in essi contenuti. Se un dato valore di PlaneOnOff è settato a 0, e il corrispondente piano non è stato considerato in PlanePick tra quelli da riempire, allora saranno

settati a 0 tutti i bit in corrispondenza dei quali esistono pixel con colori non trasparenti per l'oggetto. Se il suddetto valore di PlaneOnOff fosse invece settato a 1, allora verrebbero settati a 1 tutti i bit di quel piano in corrispondenza dei quali nell'oggetto finale vi sono pixel con colori non trasparenti.

Il sistema Gel genera e usa una maschera per i Bob che contiene dei bit settati a 1 dovunque vi sia un corrispondente bit settato a 1 nell'oggetto. Utilizzando questa maschera, il sistema può controllare il modo in cui modifica i piani di bit dell'area di visualizzazione che non vengono considerati in PlanePick.
Se PlaneOnOff ha un valore binario di 0000, allora i colori risultanti nell'oggetto saranno basati sulle posizioni dei bit settati a 1 all'interno del Pattern dell'oggetto:

```
0 X 0 X
```

da PlaneOnOff per i piani non considerati, e

```
X 1 X 1
```

da PlanePick dove i Pattern vengono disegnati. I bit segnati con una X indicano dei valori che non interagiscono con il processo di selezione dei colori.Ecco i possibili valori dei colori per questa combinazione:

```
0 0 0 0 = colore 0 (trasparente)

0 0 0 1 = colore 1

0 1 0 0 = colore 4

0 1 0 1 = colore 5
```

Pertanto PlanePick e PlaneOnOff hanno tradotto le possibili combinazioni dei colori 0, 1, 2, e 3 di un oggetto a due piani nella selezione dei colori 0, 1, 4, e 5. Se, invece, PlaneOnOff avesse avuto un valore pari a 1010, la selezione dei colori sarebbe stata:

```
1 0 1 0 = colore 10

1 0 1 1 = colore 11

1 1 1 0 = colore 14

1 1 1 1 = colore 15
```

In altre parole, in questo caso, non sarebbe stato disponibile un colore trasparente.

PlanePick e PlaneOnOff sono di fatto valori binari lunghi otto bit (UBYTE), dei quali solo sei sono usati nella corrente versione dell'Hardware di Amiga. Si può tentare mediante questi valori di alterare i colori a proprio piacimento. Si noti che PlanePick e PlaneOnOff possono essere usati per trasformare la profondità di qualsiasi oggetto definito in una profondità qualsiasi che appartenga a un Playfield sul quale si vuole disegnare.

La routine PurgeGels

Si avrà bisogno di due routine per restituire la memoria al sistema. Una di esse è chiamata PurgeGels; essa opera in maniera opposta a ReadyGels. L'altra è chiamata DeleteGels. DeleteGel dà per scontato che tutti gli elementi grafici siano annessi alla lista del sistema Gel. Il listato 7.6 è il sorgente di entrambe le routine.

Vantaggi nell'uso dei Bob

I Bob permettono di creare oggetti le cui limitazioni sono dettate solo dalla memoria disponibile per il loro uso. In questo sono totalmente diversi dagli sprite che possono essere larghi solo 16 bit.

I Bob permettono di poter scegliere tra una varietà di colori sconosciuta agli sprite. Invece di tre colori più uno trasparente (o di 15 colori più uno trasparente ottenibili in un modo particolare che qui non abbiamo trattato), si può avere un numero di colori nel corpo del Bob pari a quello dell'area del Playfield (schermo) sul quale si opera. Tale numero in modo Hold and Modify (HAM) può essere uguale persino a 4096 (la totalità dei colori disponibili su Amiga).

Per i Bob, si possono definire specificatamente le priorità di disegno manipolando i puntatori Before e After. Si può dire al sistema di disegnare questo Bob

prima di quello e dopo quell'altro, mantenendo pertanto la priorità tra i Bob al livello settato in partenza.

Svantaggi nell'uso dei Bob

Poiché i Bob vengono disegnati come parti dell'area del Playfield, usando i Bob si ottengono velocità di tracciamento inferiori a quelle che si riscontrano nell'uso dei VSprite. Inoltre, facendo parte dello sfondo stesso, bisogna operare molto spesso, in fase di disegno, una doppia bufferizzazione (come viene fatto nel programma del listato 7.7) per non perdere delle parti dell'area visualizzata.

```
/* purgegels.c */

/* Lo si usi per liberarsi degli oggetti Gel quando non se ne ha */
/* più bisogno. Si deve aver allocato le info riguardanti */
/* gli elementi Gel (mediante ReadyGel)*/

PurgeGels(g)
struct GelsInfo *g;
{
   if (g->collHandler != NULL)
      FreeMem(g->collHandler, sizeof(struct collTable));
   if (g->lastColor != NULL)
      FreeMem(g->lastColor, sizeof(LONG) * 8);
   if (g->nextLine != NULL)
      FreeMem(g->nextLine, sizeof(WORD) * 8);
   if (g->gelHead != NULL)
      FreeMem(g->gelHead, sizeof(struct VSprite));
   if (g->gelTail != NULL)
      FreeMem(g->gelTail, sizeof(struct VSprite));
}

/* Dealloca la memoria allocata dalla Routine Makexxx. */
/* Dà per scontato che le immagini e le ombre delle */
/* immagini stesse vengano deallocate da qualcun'altro */

DeleteGel(v)
struct VSprite *v;
{
   if (v != NULL) {
      if (v->VSBob != NULL) {
         if (v->VSBob->SaveBuffer != NULL) {
            FreeMem(v->VSBob->SaveBuffer, sizeof(SHORT) * v->Width
                  * v->Height * v->Depth);
```

```
            }
            if (v->VSBob->DBuffer != NULL) {
               if (v->VSBob->DBuffer->BufBuffer != 0) {
                  FreeMem(v->VSBob->DBuffer->BufBuffer,
                        sizeof(SHORT) * v->Width * v->Height * v->Depth);
               }
               FreeMem(v->VSBob->DBuffer, sizeof(struct DBufPacket));
            }
            FreeMem( v->VSBob, sizeof(struct Bob));
         }
         if (v->CollMask != NULL) {
            FreeMem(v->CollMask, sizeof(WORD) * v->Height * v->Width);
         }
         if (v->BorderLine != NULL) {
            FreeMem(v->BorderLine, sizeof(WORD) * v->Width);
         }
         FreeMem(v, sizeof(struct VSprite));
      }
   }

   /* fine di purgegels.c */
```

Listato 7.6

Questo significa visualizzare in uno schermo mentre si disegna in un altro schermo, operare uno scambio degli schermi e poi ripetere l'operazione stessa.

I Bob utilizzano una quantità di memoria superiore a quella necessaria per gli sprite virtuali poiché nel loro uso si opera quasi sempre un SAVEBACK - salvataggio dell'area dello sfondo sulla quale viene disegnato il Bob - in modo da poter ripristinare lo fondo una volta che il Bob si è spostato. Di solito, per ogni Bob presente sullo schermo vi è una quantità di memoria extra utilizzata per salvare ciò che esso ricopre.

Il programma Bob

Una volta lanciato il programma del listato 7.7, si noterà che avviene un'operazione di disegno direttamente sull'area di visualizzazione dello schermo invece che in una Window.

Le capacità di animazione grafica di Amiga sono veramente vaste. Questo capitolo ha fornito dei Tools che possono essere incorporati all'interno di programmi di grafica dinamica.

```
/* bobdemo.c */

/* Questo esempio sui Bob segue il più da vicino possibile */
/* quello sui VSprite per mostrare come il sistema GEL possa */
/* operare sugli uni e sugli altri. Confrontando i due programmi */
/* si potranno notare le differenze e le affinità esistenti */
/* fra VSprite e Bob. Viene usata la stessa immagine dati */
/* Si vedrà che i Bob non intaccano lo fondo poiché operano */
/* un salvataggio di ciò che ricoprono. Si vedrà anche che i Bob */
/* non interagiscono con i registri di colore degli sprite virtuali */

#include "exec/types.h"
#include "exec/libraries.h"
#include "exec/devices.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "graphics/layers.h"
#include "intuition/intuition.h"
#include "graphics/sprite.h"
#include "exec/memory.h"
#include "graphics/gels.h"
#include "functions.h"

/* chiede al sistema di generare e gestire MAXSP Bob */
#define MAXSP 26

/* definisce le possibili velocità dei Bob in unità VBlank */

SHORT speed[] = { 1, 2, -1, -2 };

SHORT xmove[MAXSP], ymove[MAXSP];
                        /* direzioni di movimento dello sprite */

/* cambiati per i BOBS */
struct Bob *bob[MAXSP];       /* MAXSP Bob */
struct VSprite *vspr;         /* puntatore ad uno sprite */

/* aggiunto per i Bob */
/* Tutte le combinazioni dei bit; */
/* seleziona dove assegnare i piani dell'immagine del Bob */
/* per creare le scelte dei colori  */

BYTE pick[] = { 0x03, 0x05, 0x09, 0x11, 0x06,
            0x0c, 0x18, 0x0A, 0x12, 0x14 };

short maxgot;                 /* # max di Bob creati */
```

```
struct GelsInfo mygelsinfo;   /* la RastPort della Window necessita */
                              /* di una di queste per creare i VSprite */
struct Window *w; /*  puntatore ad una Window  */
struct RastPort *rp;    /* puntatore alla RastPort di una Window */
struct Screen *s; /* puntatore a uno schermo */
struct ViewPort *vp;    /* puntatore a una ViewPort */

struct RastPort *srp;   /* puntatore a una RastPort di uno schermo */

struct Window *OpenWindow();
struct Screen *OpenScreen();
struct GfxBase *GfxBase;
struct IntuitionBase *IntuitionBase;

/* 18 word di dati del Bob = 9 linee in ognuno dei due piani */

UWORD bob_data[ ] = {
                  /* immagine del primo piano... uguale alle word di */
                  /* sinistra nei dati dell'immagine del VSprite *

                  0x0fc3,
                  0x3ff3,
                  0x30c3,
                  0x0000,
                  0x0000,
                  0x0000,
                  0xc033,
                  0xffc0,
                  0x3f03,

            /* immagine del secondo piano... uguale alle word di */
            /* destra nei dati dell'immagine del VSprite *

                  0x0000,
                  0x0000,
                  0x0000,
                  0x3c03,
                  0x3fc3,
                  0x03c3,
                  0xc033,
                  0xffc0,
                  0x3f03,
            /* piano 3 dell'immagine (di fatto i piani di dati */
            /* reali sono soltanto due, ma InitMask, chiamata da */
            /* makeVSprite e MakeBob, richiede questi piani */
            /* addizionali con Bit tutti nulli per creare la */
            /* maschera correttamente */
                  0,0,0,0,0,0,0,0,0,
            /* piano 4 */
                  0,0,0,0,0,0,0,0,0,
            /* piano 5 */
                  0,0,0,0,0,0,0,0,0 };

UWORD *bobdata;
UWORD *sprdata;
```

```
/* per muovere un Bob si muovono gli sprite componenti, pertanto */

/* il nome della Routine è appropriato */
movesprites()
{
      short i;

      for (i=0; i<maxgot; i++)
      {
            vspr = bob[i]->BobVSprite;/* cambiato per i BOB */

            vspr->X = xmove[i]+vspr->X;
            vspr->Y = ymove[i]+vspr->Y;

            /* muove gli sprite... qui */

            if(vspr->X >= 300 || vspr->X <= 0) xmove[i]=-xmove[i];
            if(vspr->Y >= 190 || vspr->Y <= 0) ymove[i]=-ymove[i];

      }

      SortGList(srp);   /* prende la lista in ordine */
      DrawGList(srp, vp);     /* crea le istruzioni per gli sprite */

      MakeScreen(s);
      RethinkDisplay();

      return(0);
}

#define AZCOLOR 1
#define WHITECOLOR 2

#define WC WINDOWCLOSE
#define WS WINDOWSIZING
#define WDP WINDOWDEPTH
#define WDR WINDOWDRAG

#define NORMALFLAGS (WC|WS|WDP)
/* Non usa WindowDrag perché non vuole che sia mosso lo schermo */

/* Permette il dimensionamento della Window in modo che l'utente */
/* possa rimpicciolirla e poi ingrandirla, per cancellare il */
/* testo di fondo. Si potrà facilmente notare che, quando vi sono */
/* molti sprite presenti, alcuni scompaiono perché il sistema */
/* non ha più sprite disponibili */

/* myfont1 specifica le caratteristiche del Font di Default; */
/* qui abbiamo un Font da 80 colonne visualizzato a 40 */
/* a causa del a bassa risoluzione */

struct TextAttr myfont1 = { (STRPTR)"topaz.font", 8, 0, 0 };

struct NewScreen myscreen1 = {
```

```
      0, 0,       /* LeftEdge, TopEdge ... dove posizionare lo schermo */
      320, 200,    /* Width, Height ... dimensiono dello schermo */
      5,          /* 5 piani di Bit, 32 colori tra i quali scegliere */
      1, 0,       /* DetailPen, BlockPen */
      SPRITES,          /* modo di visualizzazione, 0 = bassa risoluzione */
      CUSTOMSCREEN,     /* tipo di schermo */
      &myfont1,   /* Font di Default per lo schermo */
      (UBYTE *)"32 Color Test", /* Nome dello schermo */
      NULL,       /* Gadget, tutti NULL, ignorati */
      NULL };           /* indirizzo della BitMap personalizzata */
                        /* non presente qui */

struct NewWindow myWindow = {
      0,          /* margine sinistro della Window, in pixel */
                  /* dall'angolo sinistro dello schermo */
      0,          /* estremità superiore della Window in pixel dalla */
                  /* estremità superiore dello schermo attuale */
      320, 185,   /* larghezza e altezza della Window */
      0,          /* DetailPen - colore usato per i bordi della */
                        /* Window */
      1,          /* BlockPen - colore usato per i Gadget della Window */
                  /* per entrambe le penne il valore 1 significa */
                  /* che vengono usate quelle di default */
      CLOSEWINDOW, /* il programma Simple sprite usava anche */
                        /* INTUITICKS */
      NORMALFLAGS | GIMMEZEROZERO | ACTIVATE | BACKDROP,
                  /* Flag della Window */
      NULL,       /* primo gadget */
      NULL,       /* CheckMark  */
      (UBYTE *)"Click Close Gadget To Stop",/* nome della Window */
      NULL,       /* puntatore a uno schermo se non si usa il */
                        /* Workbench */
      NULL,       /* puntatore a una BitMap se si tratta di una */
                        /* Window SUPERBIMAP */
      320, 10,    /* larghezza e altezza minime */
      320, 200,   /* larghezza e altezza massime */
      CUSTOMSCREEN
      };

#include "graphics/gfxmacros.h"

/* #include "event1.c"  */
/* carica il gestore di eventi */
/* event1.c */

HandleEvent(code)
      LONG code;  /* fornito dal main */
{
      switch(code)
      {
      case CLOSEWINDOW:
            return(0);
            break;
```

```
      case INTUITICKS:
            movesprites();    /* 10 movimenti al secondo; test */
            default:
                  break;
      }
return(1);
}

UWORD mycolortable[] = {

      0x0000, 0x0e30, 0x0fff, 0x0b40, 0x0fb0, 0x0bf0,
      0x05d0, 0x0ed0, 0x07df, 0x069f, 0x0c0e,
      0x0f2e, 0x0feb, 0x0c98, 0x0bbb, 0x07df,

      0x0000, 0x0e30, 0x0fff, 0x0b40,
      0x0fb0, 0x0bf0, 0x05d0, 0x0ed0,
      0x07df, 0x069f, 0x0c0e, 0x0f2e,
      0x0feb, 0x0c98, 0x0bbb, 0x07df
      };
      /* nero, rosso, bianco, rosso fuoco, arancione, giallo */
      /* verde limone, verde, verde acqua, blu scuro, porpora */
      /* violetto, bronzo, marrone grigio, blu cielo (come sempre) */

UWORD colorset0[ ] = { 0x0e30, 0xffff, 0x0b40 };
                                    /* come per i colori 17-19 */
UWORD colorset1[ ] = { 0x0bf0, 0x05d0, 0x0ed0 }; /* 21-23 */
UWORD colorset2[ ] = { 0x069f, 0x0c0e, 0x0f2e }; /* 25-27 */
UWORD colorset3[ ] = { 0x0c98, 0x0bbb, 0x07df }; /* 29-31 */
UWORD *colorset[ ] = {
                  colorset0, colorset1,
                  colorset2, colorset3 };
int choice;
char *numbers[] = { "17","18","19",
                "20","21","22","23",
                "24","25","26","27",
                "28","29","30","31" };

#include "purgegels.c"
#include "readygels.c"
#include "makevsprite.c"
/* aggiunto per i BOBS */
#include "makebob.c"
main()
{
      struct IntuiMessage *msg;
      LONG result;
      SHORT k, j, x, y, m, error;
      UWORD *src, *dest;
            /* per copiare i dati degli sprite nella RAM */

      GfxBase = (struct GfxBase *)
               OpenLibrary("graphics.library",0);

      IntuitionBase = (struct IntuitionBase *)
                  OpenLibrary("intuition.library",0);
```

```
/* tralasciato il controllo degli errori per brevità */

s = OpenScreen(&myscreen1);  /* tenta di aprirlo*/
if(s == 0)
{
      printf("Non posso aprire myscreen1\n");
      exit(10);
}
myWindow.Screen = s;    /* localizzazione dello schermo */

ShowTitle(s, FALSE);
      /* impedisce l'abbassamento dello schermo...*/

w = OpenWindow(&myWindow);
if(w == 0)
{
      printf("LA Window non si è aperta!\n");
      CloseScreen(s);
      exit(20);
}
vp = &(s->ViewPort);
/* setta i colori per la ViewPort */

srp = &(s->RastPort);
      /* disegna i Bob direttamente sullo schermo */
LoadRGB4(vp, &mycolortable[0], 32);
rp = w->RPort;
      /* Ora il Task attende un messaggio dall'Intuition e */
      /* diviene inattivo durante l'attesa */

/* Scrive un testo con i colori degli sprite in modo */
/* da mostrare cosa succeda ai colori stessi */
/* quando si abbassa lo schermo */
/* si dovrebbe evitare l'uso dei registri di colore */
/* utilizzati dagli sprite */
/* il Bob non si cura dei registri colore */

for(j=8; j<180; j+=50)
{
      for(k=0; k<15; k++)
      {
            Move(rp,k*20,j);
            SetAPen(rp,k+17); /* usa 17-31 */
            /* 16, 20, 24, 28 non vengono coinvolti perché non */
            /* sono utilizzati dagli sprite Hardware */

            Text(rp,numbers[k],2);
      }
}

/* Alloca della memoria nella RAM accessibile ai Chip Custom */
/* dove porre i dati del Bob corrente */
/* sempre necessario su macchine aventi espansioni di memoria */

bobdata = (UWORD *)AllocMem(90, MEMF_CHIP);
```

```
if(bobdata == NULL)
{
      /* non c'è memoria per il Bob */
      printf("Non c'è memoria per i dati del Bob!\n");
      goto finish;
}
/* ora copia i dati del Bob nella Ram prima allocata */

src = bob_data;
dest = bobdata;   /* sorgente, destinazione */

for( j=0; j<45; j++)
{
      *dest++ = *src++;
}
maxgot = 0;

/* Prepara il sistema GELS a lavorare con VSprite o Bob */

error = ReadyGels(&mygelsinfo, srp);
for(k=0; k<MAXSP; k++)  /* qualunque numero di Bob */
{
      xmove[k]=speed[RangeRand(4)];
      ymove[k]=speed[RangeRand(4)];

      /* fissa una posizione per il Bob */
      x = 10 + RangeRand(280);
      y = 10 + RangeRand(170);

      /* Dimostra che con un Bob si può usare una varietà */
      /* maggiore di colori, mediante PlanePick e PlaneOnOff, */
      /* di quanti siano disponibili con il semplice uso dei */
      /* VSprite */

      /* Crea un BOB */
      bob[k] = MakeBob( 16, 9, 5, bobdata,
             pick[RangeRand(10)], RangeRand(31),
                   x,y,SAVEBACK | OVERLAY );

      /* largo 16 Bit, alto nove linee, profondo 5 piani di Bit */
      /* (vi sono infatti 3 piani extra quindi, anche se */
      /* l'immagine è su 2 soli piani, per ogni Bob che noi */
      /* disegniamo devono essere salvati 5 piani di dati!!!) */
      /* Si riempiono in modo Random di 0 e 1 i piani non */
      /* considerati in PlanePick, in modo da avere i colori */
      /* appropriati nella maschera di Bit del Bob */
      /* viene posto a X, Y; si salva e poi si ripristina */
      /* lo sfondo che viene ricoperto */

      if(bob[k] == 0)
      {
            printf("Out of Memory eseguendo MakeBob\n");
            goto finish;
      }
```

```
            if(k > 0)
            {
                  /* fissa un ordine di disegno */
                  m = k-1;
                  /* Disegna il Bob corrente DOPO quello precedente */

                  bob[k]->After = bob[m];
                  /* e di conseguenza disegna il Bob precedente PRIMA */
                  /* di quello appena creato */
                  bob[m]->Before = bob[k];
            }

            AddBob(bob[k], srp);
            maxgot++;
            /* I Bob non interagiscono con colorset */
      }

      while(1)    /* per sempre */
      {
            WaitTOF();
            movesprites();

            result = -1;      /* ora controlla se vi è in attesa un */
                              /* CLOSEWINDOW */
            msg = (struct IntuiMessage *)GetMsg(w->UserPort);
            if(msg != 0)
            {
                  result = HandleEvent(msg->Class);

                  /* Lascia che l'Intuition riutilizzi il messaggio */
                  ReplyMsg(msg);
            }
            if(result == 0)
            {
                  break; /* è stato richiesto un CLOSEWINDOW */
            }
      }

      /* FATTO, ora cancella */

      /* Cancella tutti i Bob che sono stati creati */
finish:
      for(k=0; k<maxgot; k++)
      {
            if(bob[k])
            {
                  DeleteGel(bob[k]->BobVSprite);
            }
      }
      /* cancella ciò che è stato creato da ReadyGels */

      PurgeGels(&mygelsinfo);
      if(bobdata)
      {
            FreeMem(bobdata, 90);   /* libera le risorse allocate */
```

```
        }
        if(w)
        {
                CloseWindow(w);
        }
        if(s)
        {
                CloseScreen(s);
        }
}
```

Listato 7.7

Capitolo 8

Il Suono

Questo capitolo presenta il sistema sonoro di Amiga. Mostra come si possano riservare i canali audio per un proprio uso e come immagazzinare i dati direttamente nei registri audio. Per comprendere il compito assegnato ad ogni singola funzione fornita dal software sonoro di Amiga, è utile comprendere prima alcune cose riguardanti l'hardware del sistema audio.

L'hardware dell'audio

Per molte delle funzioni svolte da Amiga, come la grafica, vi sono parecchi stadi del sistema operativo tra il programma applicativo e l'hardware della macchina. Il sistema software che gestisce l'audio è leggermente diverso, infatti esso fornisce poche routine macroscopiche che svolgano i compiti richiesti, anzi, deve essere il programmatore stesso che spesso deve manipolare direttamente l'hard ware per ottenere l'effetto desiderato.

L'hardware del suono su Amiga è costituito da quattro canali audio indipendenti, dei quali due (i canali 0 e 3) sono connessi all'uscita jack di sinistra che si trova sul retro della macchina, e due (i canali 1 e 2) sono connessi all'uscita jack di destra. Ogni canale possiede un registro di regolazione del volume che permette di avere 64 diversi livelli di volume per il canale stesso.

Ogni canale possiede un registro di periodo e un registro dati. Il registro di periodo viene usato per stabilire il valore di campionamento del canale. Maggiore è il valore a cui viene settato il registro di periodo e minore è la frequenza (quindi più basso il tono) in uscita. Il registro dati punta al luogo nel quale si trovano i dati che quel canale deve utilizzare, tale luogo deve trovarsi nella memoria accessibile ai Chip Custom (MEMF_CHIP). In più, un registro di lunghezza definisce la lunghezza dello spazio di memoria nel quale i dati del canale audio risiedono.

Ognuno dei quattro canali audio può recuperare i dati di cui necessita attraverso un Direct Memory Access (DMA, accesso diretto alla memoria) oppure direttamente dal microprocessore. Il DMA è il metodo più comunemente usato. Usare il DMA, infatti, significa dare la possibilità al canale di recuperare direttamente i dati senza coinvolgere il microprocessore che può quindi fare qualcosa d'altro. Con un totale di 26 canali DMA presenti su Amiga, inclusi quattro canali DMA dedicati al suono, un programma può fare altre cose, come ad esempio

calcolare la nuova forma d'onda da utilizzare o prelevare dati dall'utente, mentre il suono viene emesso dai canali audio senza intervento alcuno del 68000.

Per produrre dei dati che siano interpretabili da Amiga come dei suoni da mandare in output, si deve fornire un'area di memoria nella quale costruire una rappresentazione numerica dei dati dell'output audio. Se si opera esclusivamente via hardware, dopo aver riempito quest'area dati, si setta il volume al livello desiderato, si fissa il registro di periodo per stabilire il valore del tono con cui saranno emessi i dati dai circuiti di generazione sonora, si fa puntare il registro dati all'area di memoria nella quale sono immagazzinati i dati stessi, si fissa la lunghezza di tali dati mediante il registro di lunghezza, e infine si attiva il DMA audio.

Comunicare con l'Audio device

Come gli altri device, l'Audio device utilizza un blocco IORequest per comunicare con il task del programmatore. Il blocco di richiesta per l'audio viene chiamato IOAudio. Per avere accesso all'Audio device, si usa la funzione OpenDevice, passandole una struttura IOAudio che contiene dati tutti nulli. OpenDevice inizializzerà l'indirizzo dell'Audio device (io_Audio), che servirà in seguito per chiamare ciascuna delle funzioni audio disponibili.

Il listato 8.1 è un piccolo sottoprogramma che mostra come si possa iniziare l'accesso all'Audio device. Esso costituisce una subroutine che può essere usata per allocare dinamicamente una struttura dati IOAudio, inizializzata a zero. Si può, quindi, usare questa struttura dati per creare un blocco di richiesta da usare con le routine audio. La routine CreateAudioIO ritorna un puntatore a un blocco di richiesta IOAudio. Si completano poi le inizializzazioni degli altri campi nel modo desiderato. Una routine corrispondente, anch'essa mostrata, libera la memoria utilizzata da questo blocco IOAudio. Un programma dovrebbe tenere traccia del numero di blocchi che esso stesso crea, e dovrebbe liberare tutto ciò che essi occupano una volta terminato il suo lavoro.

```
/* newaudioblock.c */

struct IOAudio *
CreateAudioIO()
{
```

```
        struct IOAudio *iob;

        iob = (struct IOAudio *)
              AllocMem(sizeof(struct IOAudio),
                    MEMF_CHIP | MEMF_CLEAR);

        return(iob);      /* ritorna 0 se vi è un Out of Memory */
}

void
FreeAudioIO(iob)
struct IOAudio *iob;
{
        FreeMem(iob, sizeof(struct IOAudio));
}
```

Listato 8.1

Per aprire il device, si usa un puntatore a un blocco di richiesta IOAudio allocato precedentemente. Il listato 8.2 mostra una semplice chiamata della funzione che apre l'Audio device.

La variabile globale auDevice viene settata al valore ritornato nel campo io_Device dalla chiamata della funzione OpenDevice in modo da poter copiare il valore in altri blocchi di richiesta non inizializzati per delle chiamate ulteriori di funzioni audio.

Una volta completata la allocazione, un valore chiave di allocazione viene generato e immagazzinato con i canali audio. Se la chiamata successiva non utilizza la chiave di allocazione (io_AllocKey), il comando verrà rifiutato. Così, la chiave di allocazione ritornata dalla funzione OpenDevice viene salvata per usi futuri.

Il software audio

Il software audio fornisce dei metodi per adattare l'hardware del suono di Amiga con il sistema Multitasking. Il software prevede routine per fare ciò che segue:

- Allocare uno o più canali per un uso esclusivo a favore di un solo task.

- Stabilire una priorità di utilizzazione per permettere a un suono più importante di essere sentito, anche se un altro task ha allocato quel canale.
- Permettere a un task a priorità inferiore di terminare l'uso di un canale prima che questo sia ceduto per essere utilizzato da un altro task.
- Abilitare il programmatore a far partire o fermare un canale audio così come ad accodare più suoni in modo che vengano emessi in sequenza.
- Specificare quanto spesso una particolare definizione di forma d'onda debba essere usata o informare il programmatore circa quali forme d'onda siano correntemente usate o lo siano state.
- Abilitare il programmatore a passare direttamente all'hardware se si desidera operare a quel livello.

Allocare un canale

Nel sistema multitasking di Amiga, si può desiderare di ottenere l'uso di uno o più canali audio ma si possono anche usare dei canali su prenotazione (cioè permettendo a un altro task di rubare il canale, o semplicemente di chiederne l'uso).

Il sistema audio permette al programmatore di richiedere che i canali vengano riservati per il proprio task in due modi diversi: durante la chiamata di OpenDevice e, come chiamata di funzione separata, mediante l'uso di ADCMD_ALLOCATE.

```
int error;
struct Device *auDevice;
struct IOAudio *audioIOB;
WORD allocationKey;

audioIOB = createAudioIO();

/* se audioIOB ritorna u valore nullo, vi è stato un Out of Memory */
/* e si deve uscire dando un messaggio di errore */
error = OpenDevice("audio.device,0,audioIOB,0);

/* se error è diverso da 0, ancora una volta, si deve uscire */

/* fissa l'indirizzo della device per usi futuri */
```

```
auDevice = audioIOB->ioa_Request.io_Device;

/* fissa la chiava di allocazione per usi futuri */

allocationey = audioIOB->ioa_AllocKey;

/* ora il resto del blocco di richiesta audio può essere */
/* inizializato e essere inviato alla device usando SendIO o */
/* DoIO (o BeginIO, che in effetti è una versione più veloce */
/* di SendIO) */
```

Listato 8.2

Nell'operare una allocazione in uno o nell'altro dei due modi, la priorità nell'uso dei canali richiesta dal programmatore per il proprio task viene stabilita dal campo di priorità del blocco di richiesta IOAudio di nome ioa_Request.ioMessage.mn_Node.ln_Pri. Se si usa la routine CreateAudio, il valore di priorità per la richiesta dell'uso dei canali audio sarà impostata a zero. Ecco un esempio che setta la priorità a 127, dove iob è un puntatore a un blocco di richiesta IOAudio:

```
struct AudioIO *iob;

/* setta l apriorità di questo blocco di richiesta a 127 */

iob->ioa_Request.ioMessage.mn_Node.ln_Pri = 127;
```

I valori di priorità possono andare da -128 (minimo) a +127 (massimo). Se si setta al massimo la priorità per un certo task, allora il sistema farà in modo di non impedire mai a tale task l'uso dei canali audio. Se il programmatore setta un valore di priorità inferiore e avviene una richiesta di allocazione da parte di un task avente una valore maggiore di priorità, allora o viene abortita la richiesta del task del programmatore avente priorità inferiore, o si informerà il task del programmatore che un altro task vuole utilizzare i suoi canali audio su una base di priorità.

La priorità del canale è stabilita quando viene utilizzata la funzione OpenDevice. Se si desidera cambiare il valore di priorità in un secondo tempo, si deve usare il comando ADCMD_SETPREC che verrà presentato più avanti nel capitolo.

A prescindere dal fatto che la richiesta si riservi uno o più canali (con OpenDevice, o con una chiamata separata di ADCMD_ALLOCATE), essa assume una certa forma: punta il puntatore io_Data al primo byte a un array di byte di allocazione, e setta, nel blocco di richiesta, il valore di ioa_Length in modo da specificare il numero di byte presenti nell'array suddetto.

Ogni byte di allocazione contiene quattro bit (i quattro bit meno significativi del byte) che dicono quale, o quali, dei quattro canali audio il programmatore vuole utilizzare. Le posizioni dei bit corrispondenti direttamente ai quattro canali audio, a partire dal canale 0, sono mostrate nella tabella 8.1.

Canali Audio	Posizione del Bit nel Byte dei dati	Valore Binario	Valore Decimale
0	0	0001	1
1	1	0010	2
2	2	0100	4
3	3	1000	8

Tabella 8.1

Per riservare due canali per un task, si deve creare un valore chiamato maschera di allocazione, la quale è formata da valori che rappresentano il canale destro e il canale sinistro.

Con una maschera di allocazione si può essere molto precisi, si può specificare: "Se non posso ottenere l'uso del canale 0 e 1, non voglio nessun altro canale". Oppure, si può chiedere al sistema di cercare alcune combinazioni in una sola chiamata di allocazione, come ad esempio "Dammi i canali 0 e 1, o 0 e 2, o 3 e 1, o 3 e 2". Queste combinazioni sono quelle che possono creare un effetto stereo. Per operare questa richiesta, si fornisce un array di quattro byte, nel quale ognuno dei byte rappresenta una delle combinazioni a quattro bit che formano la coppia stereo, come mostrato nella tabella 8.2.

Una volta che il sistema ha esaudito la richiesta di allocazione del programmatore, i canali che esso possiede per un uso esclusivo del proprio task sono indicati dal campo io_Unit della richiesta. Il listato 8.3 utilizza il comando ADCMD_ALLOCATE (una volta aperta la device) per cercare di allocare un ca-

nale singolo o una coppia stereo. Ogni routine del listato ritorna un valore che indica quale canale è stato allocato per l'uso del task.

Si noti che una volta che un canale è stato allocato dal sistema, esso dovrà poi essere liberato. Se il canale non viene liberato, nessun altro task potrà allocarlo per un proprio uso finché il sistema non verrà resettato.

Canali	Valore del Byte (binario)	Valore del Byte (decimale)
0 e 1	0011	3
0 e 2	0101	5
3 e 1	1010	10
3 e 2	1100	12

Tabella 8.2

Quando il sistema alloca uno o più canali in una singola richiesta, esso fornisce un valore che si usa come chiave di accesso al canale. E' possibile accodare una o più richieste di output per ognuno dei quattro canali audio. Quando una richiesta raggiunge l'inizio della coda, il valore di io_AllocKey della richiesta viene messo a confronto con il valore contenuto internamente dall'Audio device per ogni canale. E' proprio mediante questa chiave che un canale sa quale è il task al quale è correntemente affidato.

Ogni volta che un canale viene liberato, e poi riallocato, viene generato un nuovo valore chiave. Le richieste che possiedono la corretta chiave di accesso vengono esaudite; le richieste aventi invece una chiave non corretta vengono restituite a ciò che le ha inviate, con un messaggio di errore. Pertanto, le routine del listato 8.3 richiedono che vi sia un blocco di richiesta IOAudio inizializzato (sono necessari solo i campi io_Device e mn_ReplyPort), e l'indirizzo al quale si trova una variabile globale, all'interno della quale si possa salvare il valore della chiave d'allocazione. Tutte le funzioni di controllo che si possono esercitare sui canali audio richiedono un valore corretto di questa chiave di allocazione all'interno del blocco di richiesta IOAudio, altrimenti non potrebbero funzionare e la richiesta verrebbe respinta.

Se lo si desidera, si può anche utilizzare OpenDevice per allocare automaticamente i canali quando si accede all'Audio device. Questo metodo richiede le

stesse condizioni di quello precedente, ma non richiede che venga richiamata separatamente ADCMD_ALLOCATE. Il listato 8.4 contiene una routine che alloca mediante OpenDevice e ritorna un valore che rappresenta il numero del canale allocato.

Si noti che una volta che il device è stato aperto, si può recuperare un puntatore ad essa dal campo io_Device del blocco di richiesta IOAudio e una chiave di allocazione dal campo ioa_AllocKey del blocco stesso. Si noti anche che il valore contenuto nel campo io_Unit del blocco di richiesta contiene il valore della maschera che definisce quali canali siano stati allocati.

Non è necessario aprire il device più volte. Per ottenere un altro canale, una volta aperta il device, si deve usare il comando ADCMD_ALLOCATE.

```
/* getaudio.c */

WORD mykey;       /* un valore global; chiave d'accesso */

UBYTE
GetAnyStereoPair(iob)
struct IOAudio *iob;
{
      /* da per scontato che iob sia già inizializzato */
      /* nel campo device */

      /* da anche per scontato che il campo ReplyPort */
      /* sia già inizializzato */

      UBYTE stereostuff[4];
      UBYTE mychan;
      int error;

      stereostuff[0] = 3;
      stereostuff[1] = 5;
      stereostuff[2] = 10;
      stereostuff[3] = 12;

      /* Setta al precedenza della richiesta del canale */

      iob->ioa_Request.io_Message.mn_Node.ln_Pri = 20;

      /* Tipo di comando per la allocazione */

      iob->ioa_Request.io_Command = ADCMD_ALLOCATE;

      /* punta alla mappa di allocazione */
```

```
        iob->ioa_Data = (UBYTE *)stereostuff;

        /* 4 valori */

        iob->ioa_Length = 4;

        /* Non aspetta l'allocazione, i canali devono essere */
        /* disponibili! Se non si setta ADIOF_NOWAIT, il task */
        /* aspetterà inutilmente per avere una possibilità */
        /* di allocazione del canale, controllando ogni volta */
        /* che un task alloca o libera un canale */

        iob->ioa_Request.io_Flags = ADIOF_NOWAIT | IOF_QUICK;

        BeginIO(iob);

        error = WaitIO(iob);
              /* ritorna un valore non nullo in caso di errore */

        if(!(iob->ioa_Request.io_Flags & IOF_QUICK))
        {
              /* se il flag non è settato, il messaggio */
              /* è nella ReplyPort */
              GetMsg(iob->ioa_Request.io_Message.mn_ReplyPort);
        }
        if(error)
        {
              return(0);
        }
        mychan = ((ULONG)(iob->ioa_Request.io_Unit)) & 0xFF;
        return(mychan);
}

UBYTE
GetAnyChannel(iob)
struct IOAudio *iob;
{
        UBYTE anychan[4];
        UBYTE mychan;
        int error;

        anychan[0] = 1;
        anychan[1] = 2;
        anychan[2] = 4;
        anychan[3] = 8;

        iob->ioa_Request.io_Message.mn_Node.ln_Pri = 20;
        iob->ioa_Request.io_Command = ADCMD_ALLOCATE;
        iob->ioa_Data = (UBYTE *)anychan;
        iob->ioa_Length = 4;
        iob->ioa_Request.io_Flags = ADIOF_NOWAIT | IOF_QUICK;
        BeginIO(iob);
        error = WaitIO(iob);
              /* ritorna un valore non nullo se vi è un errore */
```

```
        if(!(iob->ioa_Request.io_Flags & IOF_QUICK))
        {
              GetMsg(iob->ioa_Request.io_Message.mn_ReplyPort);
        }
        if(error)
        {
              return(0);
        }
        mychan = ((ULONG)(iob->ioa_Request.io_Unit)) & 0xFF;
        return(mychan);
}
```

Listato 8.3

Attendere l'allocazione. In tutte le forme di allocazione, il task viene posto in stato di inattività finché non è stata completamente esaudita la sua richiesta di allocazione. Ogni volta che uno o più canali sono stati liberati, l'Audio device esaminerà la richiesta di allocazione e cerca di esaudire quella avente una priorità maggiore. Se non ci sono in ogni caso abbastanza canali liberi per esaudire tale richiesta, il task continuerà a rimanere inattivo attendendo la risposta dall'Audio device.

L'attesa legata all'allocazione (usando DoIO) è mostrata nelle routine Get AnyChannel e GetAnyStereoPair. Nonostante non sia visibile nella routine Open AnyAudio, il task non ritorna dalla chiamata di OpenDevice finché l'allocazione non viene completata.

Se non si desidera che task attenda l'allocazione, allora si può settare un flag addizionale, ioa_Flag, a un valore ADIO_NOWAIT. Se questo flag contiene valori diversi da zero e la allocazione non può essere eseguita immediatamente, il blocco di richiesta sarà ritornato subito. Il valore di errore (in ioa_Request.io_Error) sarà IOERR_ALLOCFAILED. Se la allocazione fallisce, il valore ritornato da qualunque routine sarà zero, e il valore dell'errore indicherà il codice di fallimento IOERR_ALLOCFAILED.

```
/* openanyaudio.c */

BYTE

OpenAnyAudio(iob)
struct IOAudio *iob;
```

```
{
      int error;
      BYTE anychan[4];
      BYTE mychannel;

      anychan[1] = 1;
      anychan[2] = 2;
      anychan[3] = 4;
      anychan[4] = 8;

      iob->ioa_Data = (UBYTE *)anychan;
      iob->ioa_Length = 4;

      error = OpenDevice("audio.device",0,iob,0);

      /* se error non è zero, deve ritornare una chiave nulla */
      /* e un valore nullo per l'unità corrente */

      if (error != 0 )
      {
            return((Byte)0);
      }
      else
      {
        .   return(((ULONG)(iob->ioa_Request.io_Unit)) & 0xFF);
      }
}
```

Listato 8.4

Ecco un esempio che mostra come si possa settare il flag di ritorno immediato:

```
iob->ioa_Request.io_Flags = ADIOF_NOWAIT;
```

Utilizzare i canali allocati. Quando si apre l'Audio Device, riempie il campo io_Request.io_Device del blocco di richiesta IOAudio fornito dal programmatore alla funzione OpenDevice.

Quando si allocano uno o più canali, o durante l'esecuzione di OpenDevice sull'Audio device, o più tardi con l'uso di ADCMD_ALLOCATE, il sistema inserisce nel campo ioa_Request.io-Unit della richiesta un valore che rappresenta il canale o i canali allocati. Il sistema inoltre assegna una chiave esclusiva di allocazione in ioa_AllocKey rappresentante la richiesta.

Se si usa CreateAudioIO per creare più blocchi di richiesta in modo da accodare più comandi audio, allora tutti e tre i valori - quello riguardante il device, quello riguardante l'unità, e quello rappresentante la chiave di allocazione - devono essere copiati da un blocco di richiesta IOAudio già inizializzato per poter essere usati per inviare richieste al canale.

La sola eccezione si ha quando sono stati allocati con successo due canali stereo per un task. In questo caso, si cambia il valore dell'unità per poter mandare comandi separatamente a ognuno dei canali stereo allocati. Per esempio, se sono stati allocati i canali 1 e 2, il valore binario dell'unità che sarà ricevuto è 0110. Si manderanno dei comandi ai canali in modo separato utilizzando per l'unità i valori 0100 (per il canale 2) e 0010 (per il canale 1). La richiesta di allocazione, ritornando il valore originale 0110, indica semplicemente che è stato ottenuto l'uso dei canali 1 e 2. Sarà compito del programmatore poi inviare i comandi separatamente a ognuno di essi dividendo il numero dell'unità che è stato ricevuto. Se si desidera che il task vada avanti a fare qualcos'altro nel caso dovesse fallire al primo tentativo una richiesta di allocazione, si setti la variabile flag di comando a ADIOF_NOWAIT.

Bloccare un canale

Il valore di priorità che viene settato dal programmatore nella richiesta audio dirige l'Audio device in modo che essa rifiuti le richieste aventi priorità pari o inferiore a quella della richiesta del programmatore stesso. Se si desidera mantenere l'uso di un canale per sempre, si deve settare il valore della priorità a 127. Una volta che ci si è garantiti l'uso del canale, nessuna altra richiesta di nessun altro task sarà presa in considerazione finché non si libera il canale. Se si desidera però ripartire le risorse del sistema in modo dinamico si dovrebbe abbassare la priorità e bloccare il canale operando un Lock.

Infatti se un task effettua una richiesta avente una priorità superiore a quella del programmatore, vi sono due possibilità: o il task del programmatore lascia che il task a priorità superiore gli rubi il canale fermando improvvisamente l'output sonoro in corso, oppure stabilisce un Lock sul canale. Se si opera un Lock, il task riceverà solo un messaggio che lo informa che un task a priorità superiore sta richiedendo l'uso del canale audio e che quindi deve finire ciò che sta facendo e liberare il canale il più presto possibile. Se si stanno utilizzando semplice-

mente i comandi standard di I/O dell'Audio device, allora non sarà necessario bloccare il canale.

Comunque, se, dopo aver allocato un canale, il proprio task stesse scrivendo direttamente nei registri audio, si dovrebbe usare la funzione di Lock per cancellare i registri in modo appropriato, e per esplicitare la fine dell'uso dei registri stessi. Questo fa in modo di liberare il canale e permette al task a priorità superiore di accedervi. Se non si utilizzasse la funzione di Lock prima di scrivere direttamente nei registri, entrambi i task tenterebbero di servirsi degli stessi registri di output creando pertanto effetti imprevedibili.

Generalmente, si desidererà bloccare un canale subito dopo la sua allocazione. Se non si potesse bloccare il canale, significherebbe che un task a priorità superiore ha già rubato il canale per un suo uso. Se l'operazione di Lock ha successo, allora si possiede il canale in modo esclusivo finché non lo si libera, a prescindere dalla priorità che si possiede.

Per utilizzare il listato 8.5, il blocco di richiesta IOAudio deve contenere la chiave di allocazione e il valore auDevice. Per operare effettivamente l'I/O, si deve anche avere una porta di risposta alla quale possa essere mandata la richiesta di I/O una volta compiuta l'operazione di I/O. Pertanto, ioa_Request.ReplyPort deve contenere l'indirizzo valido di una porta di risposta.

Si noti che il blocco di richiesta IOAudio che viene usato in questo esempio viene semplicemente inviato all'Audio device e viene trattenuto (non restituito al task) finché questo task non libera il canale o finché un altro task a priorità maggiore non ruba il canale stesso. Pertanto, si può vedere ancora un motivo per usare al funzione CreateAudioIO: essa fornisce blocchi di richiesta multipli da usare per comunicare con l'Audio device.

Settare un nuovo valore di priorità

Si potrebbe essere in una condizione in cui si devono generare parecchi suoni, dei quali alcuni sono importanti e alcuni sono di sottofondo. Per i suoni importanti, non si desidera certo che un altro task rubi il canale in uso.

```
/* lockachannel.c */

LockAChannel(lockiob,channel)
struct IOAudio *lockiob;
UBYTE channel;
{
      /* dice quale canale bloccare */

      lockiob->ioa_Request.io_Unit = (struct Unit *)channel;

      lockiob->ioa_Request.io_Command = ADCMD_LOCK;
      lockiob->ioa_Request.io_Flags = 0;

      /* Lancia questo comando. Non ritorna alla porta di */
      /* risposta a meno che un task a priorità maggiore */
      /* gli rubi il canale o questo task non liberi */
      /* il canale stesso. E'utile avere due porte di */
      /* di risposta, una per l'I/O standard e una */
      /* per le richieste di allocazione del canale */
      /* da parte di altri task */

      BeginIO(lockiob);
}

FreeAChannel(iob)
struct IOAudio *iob;
{
      /* i valori della chiave di allocazione e del numero */
      /* dell'unità devono già essere validi */
      iob->ioa_Request.io_Command = ADCMD_FREE;
      iob->ioa_Request.io_Flags = IOF_QUICK;
      BeginIO(iob);
      WaitIO(iob);
}
```

Listato 8.5

Pertanto, si desidererà una priorità alta per il proprio canale audio nel periodo nel quale vengono emessi i suoni importanti e una priorità inferiore quando vengono emessi i suoni meno importanti.

Il comando ADCMD_SETPREC permette al programmatore di stabilire dinamicamente la precedenza (un valore di priorità) del canale che si possiede. Il listato 8.6 contiene una routine per settare un nuovo valore di precedenza. Si noti

che il valore corrente della precedenza sarà stato settato durante l'esecuzione della funzione OpenDevice.

Controllo dell'output audio

Ora che si sa come allocare e liberare i canali, si deve sapere come fornire i dati a tutti i canali audio per creare l'effettivo output sonoro. Vi è un insieme di comandi il cui scopo è proprio di controllare l'output audio. In più, vi sono due comandi che permettono di sincronizzarsi con l'ouput dei canali audio. I comandi di controllo sono i seguenti:

CMD_WRITE	Accoda ad altre la definizione di una forma d'onda perché possa essere usata per l'output sonoro. Questo comando può includere anche un setting per fissare i valori del periodo e del volume di una forma d'onda o può fare in modo che siano usati i valori di periodo e di volume del CMD_WRITE precedente.
ADCMD_PERVOL	Cambia il periodo o il volume (o entrambi) di una certa forma d'onda che è in fase di esecuzione su di un canale. Il cambiamento può essere immediato o può avvenire dopo che è stato completamente eseguito il ciclo della forma d'onda corrente.

```
/* setprec.c */

int
SetChanPrecedence(iob,channel,prec)
struct IOAudio *iob;
BYTE channel;
BYTE prec;
{
            iob->ioa_Request.io_Command =ADCMD_SETPREC;
            iob->ioa_Request.io_Unit    = channel;
            iob->ioa_Request.io_Message.mn_Node.ln_Pri = prec;
            BeginIO(iob);
```

```
    WaitIO(iob);            /* aspetta che accada */
    return(iob->ioa_Request.io_Error);
                    /* 0 se non ci sono errori */
}
```

Listato 8.6

ADCMD_FINISH — Blocca l'esecuzione della corrente forma d'onda CMD_WRITE, o immediatamente, o dopo il completamento del ciclo attuale.Il valore del flag ADIOF_SYNCYCLE viene usato sia da ADCMD_PERVOL sia da ADCMD_FIMISH, per specificare se lo stop dell'esecuzione deve avvenire subito (di Default) o dopo il completamento del ciclo corrente (se il flag viene settato).

CMD_STOP — Blocca temporaneamente l'output del canale.

CMD_START — Fa ripartire l'output del canale bloccato.

CMD_FLUSH — Rimuove dalla coda di attesa del canale tutto ciò che vi si trova scritto.

CMD_RESET — Ripristina tutti i registri audio allo stato iniziale. Esegue un CMD_FLUSH, esegue un CMD_START, e ripristina i vettori di Interrupt dell'audio al Default di sistema. Questo è un comando multicanale; esso ha effetto su tutto l'hardware dell'audio ma non ha effetto sul bloccaggio dei canali. Se un task ha operato un Lock su di un canale, esso resta ancora bloccato, e sarà il task stesso che dovrà liberare il canale.

I comandi di sincronizzazione sono i seguenti:

CMD_READ — Ritorna un puntatore al blocco di richiesta IOAudio, informando il programmatore che un certo canale audio sta emettendo un suono. (Può trovarsi in qualsiasi punto del ciclo).

ADCMD_WAITCYCLE — Fa in modo che il comando non ritorni finché non è stato completato l'ultimo CMD_WRITE. (Si usi DoIO per inviare questo comando alla device).

Usando una combinazione di CMD_READ e ADCMD_WAITCYCLE, il proprio task può sincronizzarsi con l'output audio accodato. Questa caratteristica può rivelarsi molto utile per sincronizzare azioni grafico-sonore. Per esempio, si po-

trebbe creare un certo suono quando un oggetto sbatte e rimbalza contro un muro.

I dati audio

Per comprendere come si possano creare dei dati compatibili con la funzione svolta dei canali audio, si devono sapere alcune cose riguardo a come l'hardware interpreta i dati inviatigli. Per definire un output audio è necessario specificare tre cose:

- La forma stessa dell'onda
- Il periodo di campionamento
- Il valore del volume

Definizione della forma d'onda

La definizione della forma d'onda che si deve fornire in una tabella di valori digitali corrisponde alla forma dell'onda sonora che si sta cercando di riprodurre. Si supponga che la forma appaia come in figura 8.1.

Per riprodurre questa forma d'onda, la tabella dati deve contenere un determinato numero di byte che rappresentino la forma dell'onda campionata a intervalli discreti. in poche parole, si ottiene una miglior qualità di riproduzione se utilizza l'intera gamma dinamica dell'hardware audio. Questo significa che, come mostrato nella tabella 8.3, si dovrebbe progettare un'onda il cui massimo sia il più vicino possibile al valore +127 e il cui minimo sia il più vicino possibile a -127. In questo modo, si fornisce la massima risoluzione di valori per misurare la forma dell'onda stessa. Il processo di conversione di una forma d'onda in una serie di valori numerici discreti viene detto conversione analogico-digitale.

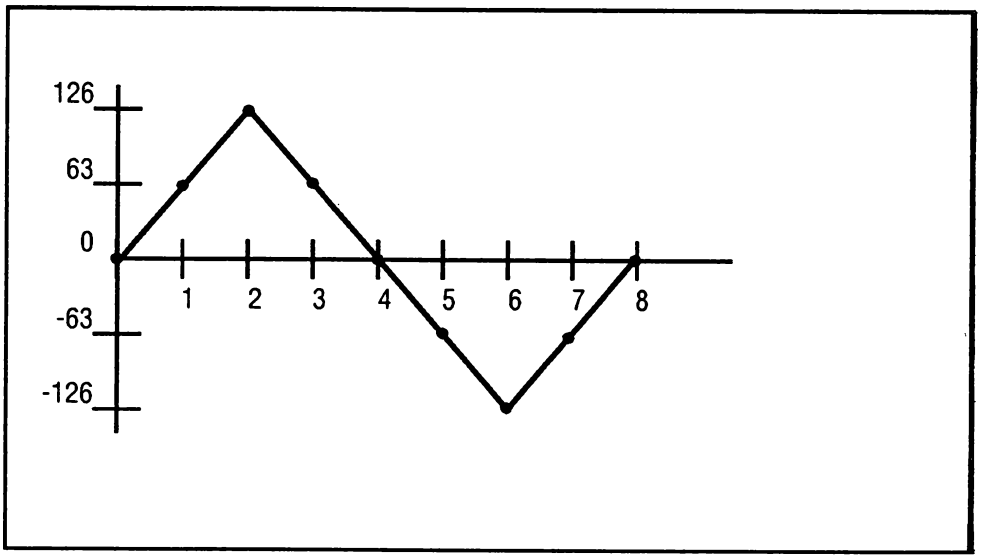


Figura 8.1

Periodo di campionamento. Per costruire la forma d'onda a partire dai dati digitali, l'hardware dell'audio invia i dati a un convertitore analogico-digitale, un dato per ogni periodo di campionamento. I circuiti audio, a turno, riempiono i segmenti temporali esistenti tra un dato e l'altro, ricostruendo così un modello della forma d'onda.

Più veloce è il campionamento, più spesso viene letta e campionata la tabella della forma d'onda, quindi, maggiore è il valore di ripetizione nella lettura dei dati. Questo significa una elevata frequenza in uscita, cioè una nota alta. Più lento è il campionamento, più basso è il valore della frequenza e pertanto più bassa la nota riprodotta.

Su Amiga il periodo di campionamento per una forma d'onda viene specificato da un valore divisore (chiamato period) piuttosto che da un valore che esprima direttamente l'intervallo di campionamento. Il valore divisore viene applicato a un Clock di campionamento interno che, diviso per questo valore, determina il valore attuale di campionamento. Questo implica che il valore che specifica la frequenza di uscita della forma d'onda corrente è inversamente proporzionale al valore period che controlla il periodo di campionamento. In altre parole, se si desidera una nota molto alta, si usa un valore molto basso di period, se si desidera una nota molto bassa, si specificherà un valore alto di period.

Il valore minimo di period è 126, questa è una limitazione legata all'hardware DMA di Amiga. Si veda l'Amiga Hardware Manual se si desiderano maggiori informazioni circa i limiti dell'hardware audio.

Valore dell'intervallo di tempo	Ampiezza della forma d'onda
0	0
1	63
2	126
3	63
4	0
5	63
6	126
7	63

Tabella 8.3

Cancellare l'output audio. Come parte del comando CMD_WRITE, si può dire al sistema quante volte una certa forma d'onda deve essere emessa. Pertanto, la stessa tabella di dati digitali può essere letta più e più volte prima che un nuovo comando CMD_WRITE venga eseguito. Per evitare che vi siano disturbi nell'emissione sonora, si dovrebbe definire una forma d'onda in modo tale che il passaggio tra forme d'onda con ripetizione di una stessa avvenga in modo sfumato. Naturalmente questo vale anche per il passaggio da una forma d'onda a un'altra diversa da questa.

La transizione sfumata tra due tipi di forme d'onda avviene nel punto 0,0. Cioè, se una certa forma d'onda parte e finisce con un valore nullo, sarà facile legarvi più forme d'onda, anche differenti, se anche queste hanno un valore nullo (toccano l'asse X) all'attacco e alla fine.

Se due forme iniziano e finiscono, per esempio, al valore 37, non avverrà necessariamente una distorsione se la pendenza di entrambe è uguale nel punto in cui avviene il passaggio (per esempio se tutte e due hanno pendenza negativa nel punto in cui si uniscono). Però, unire due forme d'onda aventi entrambi valore nullo è meglio, perché in tali condizioni il valore del voltaggio sui canali di output è zero e pertanto è disponibile il minor quantitativo di energia sonora, e quindi si evitano facilmente i disturbi che potrebbero derivare dal passaggio da un'onda a un'altra.

Posizionare i dati audio. I dati audio devono trovarsi all'interno della memoria accessibile ai Chip Custom. Questo significa che si deve allocare della memoria di tipo MEMF_CHIP e vi si devono copiare o generare i dati audio. La memoria che si trovi al di fuori di quella accessibile ai Chip Custom non può essere letta dall'hardware audio.

I dati devono essere nella forma di coppie di byte, allineate per word, lette una alla volta dal DMA audio. Se si usa AllocMem l'allineamento è automatico. Ecco le istruzioni per allocare la memoria per la forma d'onda mostrata nella figura 8.1:

```
BYTE *audata;

audata = (Byte *)AllocMem(8,MEMF_CHIP);
         /* sono richiesti 8 Byte */
```

Se audata è diverso da zero, allora i dati audio possono essere copiati all'interno di quell'area e possono essere usati dai Chip Custom per generare un suono.

Controllo del volume. Settando la tabella per l'output audio, è stato definito l'aspetto della forma d'onda. Si deve, però, specificare anche quanto deve essere intenso il suono. Ogni canale ha un controllo di volume ad esso associato; 64 rappresenta il massimo valore per il volume; 0 rappresenta il volume minimo. I valori compresi tra questi estremi controllano linearmente il valore del volume dei suoni in output sul canale. In alternativa, naturalmente, si può settare al massimo il volume su Amiga (come avviene nel programma Audio che segue) e regolare il reale volume del suono operando sull'amplificatore esterno.

Il programma Audio

Il listato 8.7 contiene un programma audio completo che utilizza tutti e quattro i canali. Questi canali emetteranno la semplice forma d'onda descritta a quattro diverse frequenze simultaneamente.

```
/* audio.c */

#include "exec/types.h"
#include "exec/memory.h"
#include "devices/audio.h"

extern struct IOAudio *CreateAudio();

BYTE trianglewave[8] = { 0, 63, 126, 63, 0, -63, -126, -63 };

UWORD period[4] = { 508, 428, 339, 254 };

struct Device  *auDevice=0;
BYTE *chipaudio;
struct IOAudio *audioIOB[4];
struct IOAudio *aulockIOB[4];
struct IOAudio *aufreeIOB[4];

extern struct  MsgPort *CreatePort();
struct MsgPort *auReplyPort, *auLockPort;
extern struct  IOAudio *CreateAudioIO();
extern UBYTE   GetAnyChannel();
extern void    FreeAudioIO();

main()
{
int error,i,chan;
BYTE *s, *d;
struct Unit *auunit;

for(i=0; i<4; i++)
{
      audioIOB[i]  = CreateAudioIO();
      aulockIOB[i] = CreateAudioIO();
      aufreeIOB[i] = CreateAudioIO();

      if(audioIOB[i] == 0 || aufreeIOB[i] == 0 | aulockIOB[i] == 0)
      {
            finishup("out of memory!");
      }

}

chipaudio = (BYTE *)AllocMem(8, MEMF_CHIP);
if(chipaudio == 0)
{
      finishup ("out of memory!");
}
d = chipaudio;  s = trianglewave;

for(i=0; i<8; i++)
```

```
    {
        *d++ = *s++;
/* copia all'interno della memoria accessibile ai Chip Custom */

    }

    error = OpenDevice("audio.device",0,audioIOB[0],0);

    if(error)
    {
        finishup ("L'Audio Device non si apre!");
    }
    /* prende l'indirizzo del device per usi futuri */

    auDevice = audioIOB[0]->ioa_Request.io_Device;

    /* crea delle porte per le risposte della device */

    auReplyPort = CreatePort(0,0);
    auLockPort  = CreatePort(0,0);

    if(auReplyPort == 0 || auLockPort == 0)
    {
        finishup("Non posso creare la porta!");
    }

    for(i=0; i<4; i++)
    {
        /* Inizializza gli indirizzi della porte e i campi device */
        /* per tutti i blocchi di richiesta audio */

        audioIOB[i]->ioa_Request.io_Device = auDevice;
        aulockIOB[i]->ioa_Request.io_Device = auDevice;

        audioIOB[i]->ioa_Request.io_Message.mn_ReplyPort
                    = auReplyPort;
        aulockIOB[i]->ioa_Request.io_Message.mn_ReplyPort
                    = auLockPort;
    }
    for(i=0; i<4; i++)
    {
        chan = GetAnyChannel(audioIOB[i]);

        printf("Ho il canale %ld\n",chan);

        /* Opera con la chiave di allocazione*/

        aulockIOB[i]->ioa_AllocKey = audioIOB[i]->ioa_AllocKey;

        /* Opera con il numero di unità */
        auunit = audioIOB[i]->ioa_Request.io_Unit;

        aulockIOB[i]->ioa_Request.io_Unit = auunit;

        LockAChannel(aulockIOB[i],chan);
```

```
        /* Se checkio ritorna True, allora la richiesta è stata */
        /* restituita. Vuol dire che il canale ci è */
        /* stato rubato */

        if(CheckIO(aulockIOB[i]))
        {
              finishup("Mi hanno rubato un canale!");
        }

}
for(i=0; i<4; i++)
{
        /* dando per scontato che nulla ci è stato rubato */
        /* una richiesta di output su un canale */

        audioIOB[i]->ioa_Data   = (UBYTE *)chipaudio;
        audioIOB[i]->ioa_Length = 8/2; /* 4 WORDS nella tabella */
        audioIOB[i]->ioa_Period = period[i]; /* dalla tabella */
        audioIOB[i]->ioa_Volume = 64;/* massimo */
        audioIOB[i]->ioa_Cycles = 10000;/* 10000 volte */

        audioIOB[i]->ioa_Request.io_Command = CMD_WRITE;
        audioIOB[i]->ioa_Request.io_Flags = ADIOF_PERVOL;

        /* copia il blocco audio per liberare più tardi i canali */

        *aufreeIOB[i] = *audioIOB[i];

        printf("Sto inviando la richiesta\n");
        BeginIO(audioIOB[i]);
}
for(i=0; i<4; i++)
{
        WaitIO(audioIOB[i]);
}
for(i=0; i<4; i++)
{
        FreeAChannel(aufreeIOB[i]);
        printf("Sto liberando un canale\n");
}
finishup("Fatto!\n");
}


finishup(string)
char *string;
{
        int i;

        if(auDevice) CloseDevice(audioIOB[0]);
        if(chipaudio) FreeMem(chipaudio,8);
        for(i=0; i<4; i++)
        {
              if(audioIOB[i]) FreeAudioIO(audioIOB[i]);
```

```
            if(aulockIOB[i]) FreeAudioIO(aulockIOB[i]);
        }
        if(auReplyPort) DeletePort(auReplyPort);
        if(auLockPort)  DeletePort(auLockPort);

    printf("%ls\n",string);
    exit(0);
    }

    #include "newaudioblock.c"
    #include "lockachannel.c"
    #include "getaudio.c"
```

Listato 8.7

Se si volesse sapere di più sulla sintesi sonora digitale, io consiglio di leggere Musical Application of Microprocessor di Hal Chamberland (Hayden, 1986). Inoltre, un buon trattato sulle capacità di sintesi sonora e vocale di Amiga può essere trovato in Inside the Amiga di John Thomas Berry (Indianapolis: Sams, 1986).

Capitolo 9

Il multitasking

Come è stato detto nel terzo capitolo, l'Exec ha la capacità di gestire molti task contemporaneamente. Questo capitolo fornisce alcuni chiarimenti su come faccia l'Exec a gestire il multitasking e fornisce anche degli esempi che mostrano come creare un nuovo task e come far girare più task allo stesso tempo.

I task

L'Exec ripartisce le capacità del microprocessore permettendo così la creazione dei task. Ogni task del sistema ha l'uso esclusivo dei registri macchina nel momento in cui sta girando e eseguendo le proprie istruzioni. Se si crea e si fa partire un task, allora tale task potrà spartire le risorse del microprocessore con gli altri task esistenti nel sistema.

Un task viene definito da un blocco di controllo task. In questo blocco vi sono i parametri che definiscono lo stato operativo del task stesso, e vi sono anche quelle aree in cui vengono salvati i contenuti dei registri macchina nel periodo nel quale il task non è operativo perché ne è stata sospesa temporaneamente l'attività.

Ogniqualvolta avviene un Interrupt di sistema, l'Exec determina quale sia il task correntemente attivo e la lista dei task che sono pronti ad essere attivati. L'Exec quindi determina quale task debba essere fatto girare confrontando la priorità del task correntemente attivo con la priorità dei task in stato di attesa. Se un task in attesa ha un valore maggiore o uguale a quello del task correntemente attivo, allora lo stato completo dei registri del 68000 viene salvato nel blocco di controllo task del task corrente e lo stato dei registri salvato del nuovo task viene caricato dal suo blocco di controllo ed esso viene lanciato, diventando così il task correntemente attivo. task che abbiano priorità uguale continueranno ad alternarsi nell'uso del microprocessore, dando l'apparenza di girare contemporaneamente.

Vi sono delle limitazioni alle funzioni che ogni singolo task può svolgere. Per esempio, nessun task può fare qualcosa che richieda l'uso di una qualsiasi delle funzioni dell'AmigaDOS. La struttura dell'AmigaDOS è fatta in modo da utilizzare delle variabili addizionali legate alla struttura dati task. L'AmigaDOS usa queste variabili per ricevere messaggi, o per tenere traccia del luogo dal quale vengono caricati i codici delle istruzioni del programma o dei dati, o per tenere traccia dei Lock operati sulla Directory corrente, e così via.

Una cosa da tenere a mente è che soltanto il task con la priorità maggiore ha la possibilità di girare. Se si crea un task ad alta priorità che non opera I/O e non esegue funzioni di attesa temporizzata o di Wait, è possibile che tale task si appropri totalmente della macchina, e i task a priorità inferiore potrebbero non avere mai la possibilità di girare.

La priorità di un task può assumere un valore qualsiasi compreso tra -128 e +127. Il più delle volte i task vengono lanciati con una priorità nulla (cosa che fa l'AmigaDOS lanciando i programmi dalla CLI), pertanto ogni task ha un uguale possibilità di girare e pertanto sembrerà che tutti girino contemporaneamente. Tutti i task a priorità uguale operano un'alternanza; cioè, non appena un task viene disattivato in favore del task successivo della lista dei task pronti a girare, viene posto in fondo alla lista stessa. Quando tutti i task che lo precedono saranno stati prelevati dalla testa della lista esso stesso raggiungerà la cima e potrà essere lanciato nuovamente.

Il process

L'AmigaDOS può gestire il multitasking costruendo una propria struttura, chiamata process, in cima alla struttura dati task dell'Exec. La maggior parte della struttura dati process è legata alle parti interne dell'AmigaDOS e, pertanto, gli elementi del blocco di controllo del process non sono qui descritti. La struttura dati process è completamente listata nell'Amiga ROM Kernel Manual nel file include di nome libraries/dosextens.h.

Le strutture process svolgono molti compiti ad alto livello. L'AmigaDOS, infatti, svolge più compiti con i process che con i task. Comunque, oltre a dire che i task non possono eseguire le funzioni dell'AmigaDOS, questo libro non andrà avanti nella distinzione tra i task e i process. Si troveranno parecchi esempi in cui si lanciano task e process attraverso i quali si capirà come si possono creare delle applicazioni personali.

Il modo facile per lanciare qualcosa di nuovo

Prima di parlare del metodo più complicato per fare le cose (usare i task e i process) ecco qui un modo semplice : si possono usare le capacità dell'Amiga-

DOS per lanciare un process separato e continuare automaticamente a svolgere le funzioni del proprio programma mentre il nuovo process gira parallelamente.

Usando la CLI, si è visto certamente un modo per lanciare più programmi e farli girare contemporaneamente. Per esempio, si può scrivere:

```
RUN clock
```

e

```
RUN notepad
```

Questo fa sì che si aprano due finestre, ognuna delle quali contiene il programma corrispondente in fase operativa, cioè entrambi stanno girando. Si ha ancora, inoltre, a disposizione la CLI originale nella quale si possono digitare altri comandi. Oppure si può scrivere:

```
NEWCLI
```

e ottenere una nuova Window CLI dalla quale si possono eseguire altri comandi ancora.

Per far partire facilmente altri programmi dall'interno di un programma, si può richiamare Execute con una stringa di comando esattamente uguale a quella usata direttamente nella CLI per far partire indipendentemente i programmi. Proprio come se si scrivesse:

```
RUN QUALCOSA
```

sulla CLI, si può aggiungere al proprio programma una chiamata della funzione Execute di questo tipo:

```
success = Execute("RUN QUALCOSA",0,0);
```

dove per QUALCOSA si intende il nome di un programma che si desidera lanciare.

Come si potrà notare quando si digita il comando RUN sulla CLI, l'AmigaDOS restituisce il controllo immediatamente alla CLI in modo da poter continuare a digitare altri comandi. La stessa cosa vale per la funzione Execute. Il comando RUN carica e fa partire QUALCOSA, restituendo il controllo al programma richiamante Execute, generando cos un'esecuzione parallela di programmi. In questo modo si possono lanciare molti programmi e farli girare contemporaneamente.

Il valore ritornato in success è sempre TRUE. Cioè la funzione Execute ha sempre successo con il comando RUN.Se si desidera, si possono includere nella stringa di comando dei simboli di redirezione in modo che l'Input-Output di questo task avvenga da o verso un certo file su disco. Ecco un esempio:

```
success = Execute("RUN<inputfile>outputfile QUALCOSA",0,0);
```

dove inputfile rappresenta il nome del file dal quale si caricheranno dei dati e outputfile rappresenta il nome del file sul quale si opererà l'output generato da QUALCOSA durante la sua esecuzione. Le frecce nella stringa di comando sono i simboli di redirezione dell'AmigaDOS. Essi sono spiegati nel secondo capitolo.

Una volta che il comando RUN è eseguito, l'AmigaDOS fa partire un process separato per questo nuovo programma. Entrambi i processi, quello richiamante e quello richiamato, hanno la possibilità di girare, spartendosi le risorse della macchina.

Questo semplice modo di lanciare più programmi che girino contemporaneamente di fatto apre un nuovo process dell'AmigaDOS per ogni nuovo programma che lancia. Se non si specifica nessun tipo di redirezionamento nella stringa di comando, allora tutte le operazioni di Output vengono effettuate sulla CLI dalla quale è stato lanciato il programma originale. Non c'è bisogno di preoccuparsi del tipo di funzioni eseguite dal proprio programma poiché c'è un blocco di controllo del process impiantato dall'AmigaDOS.

Pertanto si è in grado di richiamare le funzioni dell'AmigaDOS all'interno del programma aperto. Si ricorda ancora però che, se si lanciano solo task, invece che process, bisogna stare attenti al tipo di funzioni utilizzate all'interno dei task. In altre parole, si dovrebbe evitare di utilizzare tutto ciò che direttamente o indirettamente utilizza l'AmigaDOS. Richiamare direttamente l'AmigaDOS implica un uso diretto delle funzioni dell'AmigaDOS stesso descritte nel secondo capitolo

(come Open, Read, Write, Close e Delay). Richiamare indirettamente l'AmigaDOS implica l'uso di alcune delle funzioni di I/O residenti nelle librerie C di Amiga.

Le funzioni di I/O sono definite come operazioni che in un modo o nell'altro fanno avvenire uno scambio di dati tra un programma e delle periferiche. Tra queste funzioni troviamo getchar, putchar, fopen, fclose, printf, e fprintf. Il concetto guida che permette di riconoscere una funzione di I/O è : "E' l'uso di questa funzione un modo per muovere dei dati attraverso una periferica, come la tastiera, lo schermo o il disco?". Se la risposta è affermativa, potrebbe accadere che un task non sia in grado di eseguirla. Invece un process può fare qualunque cosa, ed è per questo che è meglio aprire un process piuttosto che un task. Perché, allora, usare un task se un process va bene ugualmente? Orbene, a volte si desidera minimizzare lo spazio occupato da un programma, e, se un task può fare ciò di cui si ha bisogno, l'uso di un task risulta la scelta più logica.

Perché, allora, impantanarsi nella difficile gestione dei task e dei process quando risulta facile lanciare un nuovo programma dalla CLI o da un altro programma nel modo precedentemente visto? A volte si desidera personalizzare il modo in cui il sistema gira e magari si vuole che un task comunichi o cooperi con un altro task. L'esempio che segue mostra come si possa ampliare il blocco di controllo task per fornire una intercomunicazione tra task diversi.

Un esempio di Tasking

L'esempio di questo paragrafo contiene un Tool per inizializzare un blocco di controllo task e per far partire un task separato. Per motivi di convenienza sia il task primario, sia quello lanciato da esso sono contenuti nello stesso listato. Questo è un programma molto semplice e, di fatto, né il programma primario né quello secondario fanno qualcosa di realmente significativo. Esso esiste solo per avere un valore esplicativo. Il programma principale opera semplicemente un'inizializzazione del task secondario, e poi invia ad esso un messaggio di StartUp, e si pone in stato di inattività attendendo la risposta.

Il programma principale avrebbe potuto essere progettato per fare qualcos'altro invece di limitarsi ad aspettare che il programma secondario gli rispondesse. Per esempio, esso poteva leggere dei dati immessi dall'utente, e il programma secondario poteva fare dei calcoli sui dati introdotti precedentemente, mentre il

primario, appunto, prelevava nuovi dati. (Precisamente, si immagini che il task minore esegua un algoritmo per il calcolo di una nuova mossa per un programma di scacchi mentre il task principale aggiorna il cronometro di gioco e attende la nuova mossa dell'utente).

Una volta che il task minore ha risposto al messaggio di Startup, entra in un Loop senza fine (mediante un Wait). Normalmente, quando si scrive un programma, del tipo:

```
main()
{
   printf("Ciao a tutti\n");
}
```

è perfettamente ammissibile "dimenticarsi" la fine. In altre parole, si può tranquillamente omettere qualunque istruzione return o exit, poiché viene data per scontata dal compilatore C. Una volta che il programma è terminato, ritorna al codice di StartUp (Astartup o LStartup) con il quale è stato linkato all'inizio. Tale codice fa in modo di inizializzare correttamente il programma e si preoccupa anche di cancellarlo una volta terminato.

Se si lancia un task, non è possibile "dimenticarsi" la fine, perché il sistema non possiede un codice di StartUp associato a quel task, e quindi non saprà dove tornare e cosa fare. Si possono fare due cose diverse per far terminare il task. Si può forzare il programma in un Wait senza fine. In tal caso, l'AmigaDos blocca il programma principale quando finisce e, di conseguenza, il programma principale libera la memoria del task secondario in Loop e quindi cancella il task dalla lista di sistema dei task. Oppure si può fare in modo che il task si autodistrugga una volta finito. Questo metodo viene mostrato in questo capitolo.

Il File di Link per l'esempio di Tasking

Il File del listato 9.1, controlla il linkaggio per l'esempio di Tasking. Per compilare questo programma, si proceda seguendo i punti espressi nella pagina seguente.

```
; tasklink.lnk
;
; execute make ram:tasking
; execute make ram:inittask
; assign lib: df1:lib
; df1:c/alink with ram:tasklink.lnk
;
; (df1: è il disco del C.)
; NOTA.... lo si compili con l'opzione -v in LC@ per disabilitare;
la parte di controllo dello Stack.... altrimenti il Linker troverà le
; voci indefinite cxovf e _base.
; (Infatti l'esempio non usa Lstartup.obj e lc.lib.)
;
;SE si da un RUN dalla CLI:

FROM lib:Astartup.obj ram:tasking.o ram:inittask.o
TO ram:tasking
LIBRARY lib:amiga.lib
```

Listato 9.1

1. Dal proprio disco si, scriva:

```
COPY tasking.c ram:
COPY inittask.c ram:
COPY tasklink.lnk ram:
```

2. Con il disco dell'Amiga C nel Drive 1, si scriva:

```
CD df1: examples
EXECUTE make ram: tasking.c
EXECUTE make ram: inittask.c
df1:c/alink with ram:tasklink.lnk
```

Il risultato sarà un programma di nome Tasking.

3. Dalla CLI, si dia il comando:

```
RUN ram:tasking
```

e si osservi il risultato.

Il listato con il programma principale e con quello secondario

Il listato 9.2 è completamente commentato e mostra entrambi i programmi descritti: quello primario e quello secondario lanciato dal primario stesso.

La funzione InitTask

Il listato 9. contiene la funzione InitTask usata nel listato 9.2. Essa differisce dalla funzione CreateTask fornita nelle funzioni di Library di Amiga per il fatto che il task creato non viene lanciato automaticamente. L'esempio di Tasking attende infatti un messaggio di StartUp invece di partire immediatamente. In più, il programma principale alloca e disalloca la memoria necessaria al task secondario, mentre in CreateTask la maggior parte delle allocazioni di memoria è eseguita dalla funzione stessa invece che dal process che lancia il secondo task.

Un esempio di Processing

Se il programma che si scrive deve eseguire un I/O o deve comunque utilizzare delle funzioni dell'AmigaDOS, allora dovrà essere lanciato come process invece che come task. Anche questo fa parte delle capacità multitasking di Amiga, ma viene eseguito ad alto livello, cioè dall'AmigaDOS stesso.

Vi sono due programmi in questo paragrafo. Il programma proctest carica e lancia littleproc,e poi scarica le sue istruzioni e i suoi dati una volta che questo è terminato.

```
/* tasking.c */

/* Semplice programma che lancia un task proprio  -

      * Creazione del task generatore e del task generato.

      * Il programma principale alloca della memoria per il blocco
      * di controllo e per una porta di messaggio per il task
```

```
      * generato.

      * Il task principale inizializza la porta e il TBC e li
      * fornisce al task generato.

      * Il task generato si pone in stato di inattività
      * attendendo che un messaggio arrivi alla sua porta.

      * Inviando quel messaggio il task generatore si pone esso
      * stesso in stato di inattività attendendo la risposta del
      * task generato.

      * Quando il messaggio del task generatore arriva alla porta
      * di quello generato, questo si sveglia e risponde.
      * Poi entra in un Loop senza fine.

      * Il task generatore si risveglia ricevendo il messaggio sulla
      * propria porta di risposta, disalloca la memoria del task
      * precedentemente generato e termina esso stesso */

/* Software di sistema V1.1  o successive */

/* Informazioni riguardanti il programma:

      si faccia il Link con Astartup.obj, InitTask.o, amiga.lib
      per creare tasking */


#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/tasks.h"
#include "exec/execbase.h"

#include "exec/io.h"

#define PRIORITY 0
#define STACKSIZE 500

extern struct Message   *GetMsg();
extern struct MsgPort   *CreatePort();
extern struct MsgPort   *FindPort();
extern APTR             AllocMem();
extern struct task      *FindTask();
extern int        InitTask();

struct MyExtendedTask {
        struct task          met_Task;    /* blocco di controllo task */
        struct MsgPort       met_MsgPort;/* a una porta di messaggio */
        int                  met_Status; /* un valore di status per info */
```

```
};

littletask()
{
      int signalbit;
      /* Bit di segnalazione per il controllo degli errori */

      /* puntatore alla porta di messaggio del task generato */

      struct MsgPort *mp;

      /* puntatore al messaggio del task generato*/

      struct Message *msg;

      /* puntatore ad un task, in questo caso, noi stessi */

      struct MyExtendedTask *met;

#ifdef AZTEC_C
      geta4();          /* operazione di Set Up */
#endif

      met = (struct MyExtendedTask *)FindTask(0);

      /* usato per dire che tutto è OK finora */

      met->met_Status = 0;

      /* Punta alla porta di messaggio */

      mp = &(met->met_MsgPort);

      /* Ora bisogna settare la porta di messaggio e disattivarsi
       * attendendo che il messaggio arrivi alla porta stessa
       * Dice alla porta a quale task deve segnalare l'arrivo di
       * un messaggio a questa porta.  Non succede nulla finché
       * mp_Flags non viene settato a PA_IGNORE dal task primario. */

      mp->mp_SigTask = (struct task *)met;

      /* Ora riceve il numero di un Bit di segnalazione...questo */
       * non fallirà perché il task è nuovo è ha una moltitudine
       * di Bit di segnalazione a disposizione. Comunque il
       * controllo dell'errore viene fatto in ogni caso, sta al
       * programmatore scegliere cosa fare in caso di fallimento */

      signalbit = AllocSignal(-1);
      /* alloca ogni Bit di segnalazione */
      if(signalbit != -1)
      {
            /* E' stato allocato un Bit di segnalazione valido! */
            /* ORA, dice alla porta di segnalarci l'arrivo di */
            /* di un messaggio. */
```

```
        mp->mp_Flags = PA_SIGNAL;
}
else
{
        /* non era disponibile alcun Bit di segnalazione */
        met->met_Status = -1;
        goto finish;
}
/* Se è avvenuto un errore, allora non avverrà mai settaggio
 * del Flag a PA_IGNORE. Il risultato di ciò è che, questo
 * povero task minore potrebbe restare inattivo per sempre
 * in attesa di un messaggio che non arriverà mai. Il task
 * principale dovrebbe inviare il messaggio, e nel caso non
 * riceva alcuna risposta dal task secondario entro un certo
 * periodo di tempo, dovrebbe controllare lo status del task
 * per vedere se è realmente partito. Se cos non è allora
 * nella variabile di status potrebbe trovarsi un messaggio
 * di errore. Invece di forzare questa azione, abbiamo scelto
 * di terminare il task. (goto finish) */

WaitPort(mp);       /* Attende il settaggio dei Bit */
                        /* di segnalazione */

/* Se vi si trova già un messaggio, allora non resteremo */
/* inattivi un solo istante */

msg = GetMsg(mp);

/* è stato risposto al messaggio  con successo */

met->met_Status = 1;

/*    QUI SI FANNO ALTRE COSE... QUALUNQUE COSA SI VOGLIA
      CHE IL TASK FACCIA */

finish:

Forbid();         /*  Disabilita l'alternanza dei task */
ReplyMsg(msg);    /* rimanda il messaggio al task di partenza */

 * Ci si aspetta che vengano liberati i Bit di segnalazione
 * mediante FreSignal. Però, poiché il task sta per essere
 * rimosso, non ci preoccupiamo di fare ciò */

RemTask(0); /*
/*
 * Si autorimuove dalla lista dei task; il task successivo */
 * può iniziare a girare. Si può fare questo
 * grazie alla MemList presente in InitTask... ogni memoria
 * presente su tale lista viene restituita al sistema
 * automaticamente quando viene eseguita RemTask */

/* Si noti che il task minore DEVE finire con un Wait() o
 * qualche tipo di Loop senza fine, o con qualsiasi cosa
 * faccia si che esso venga rimosso nel modo qui mostrato.
```

```
        * Se esso finisse da solo non restituirebbe il controllo a
        * nessuno. Il task minore deve esistere mentre il task
        * principale lo cancella.
        */
}           /* fine del task secondario */

/* ************************************************************************/


/* ************************** NOTA ***************************


Questo esempio NON tenta di far stampare qualcosa a questo task
secondario mediante un printf perché esso non è un process
ma solo task. Quando si lancia qualcosa con RUN (o semplicemente
dal CLI), tale programma viene allegato a un process, che è
qualcosa di un livello superiore rispetto a un task. Le
funzioni descritte nel ROM Kernel Manual possono essere eseguite
dai task. Le funzioni descritte nell'AmigaDos Developers'
Manual o dal Lattice C Manual devono essere richiamate dal main()
o da.qualcuna delle sue SubRoutine. Tutto ciò che viene lanciato
come task e, non come process, deve limitarsi a usare le funzioni
a cui sono abilitati i task.  (Delay(xx) è una funzione DOS
pertanto il task secondario non può utilizzarla).

(printf implica un Output controllato da una Window CON: o RAW:
dell'AmigaDOS, e quindi richiede un uso interattivo dell'AmigaDOS
che può avvenire solo attraverso un process e non un task)

Un utente di task deve essere particolarmente attento a quelle
cose di tipica competenza dei task, soprattutto ciò che avviene
attraverso istruzioni totalmente contenute nel codice del task
stesso. Ogni cosa che debba utilizzare il DOS dovrebbe essere
evitata. Questo comprende l'apertura di Library, di Font o
Device, poiché ognuna di queste cose causa un accesso al disco
(per il caricamento di parti di codice non residenti). Si possono
trovare delle scappatoie, ma la regola è che, se si deve usare
l'AmigaDOS, è molto meglio lanciare un PROCESS piuttosto che un task.

Vi è un'altro esempio nel capitolo che mostra come lanciare un
process invece di un task.

Questo esempio di Tasking funziona con un codice macchina che
risulti residente, cioè quando sia il main() che il task
secondario sono caricati contemporaneamente.
L'esempio del process carica il suo secondo programma e lo
scarica una volta terminato.

*************************************************************************/

main()
{
      struct Message mymessage;
            /* una struttura dati Message */
```

```
struct MsgPort *mainmp;
/* puntatore alla porta di risposta del main() */
struct MyExtendedTask *met;
/* puntatore al blocco di controllo di un task esteso */

struct MsgPort *mp;
/* puntatore alla porta di messaggio del task secondario */

int result;             /* diverso da zero se InitTask è OK */
printf("\n Lanciato il main()\n");

mainmp = CreatePort(0,0);

/* Usata sola per la risposta, quindi non necessita di un */
/* nome, l'indirizzo della risposta è contenuto nel */
/* messaggio stesso */


if(mainmp == 0) exit(20);     /* errore creando la port */

/* Setta la struttura dati del messaggio per poter usare */
/* PutMsg per trasmetterla al task secondario */

mymessage.mn_Node.ln_Type = NT_MESSAGE;
mymessage.mn_Length       = sizeof(struct Message);
mymessage.mn_ReplyPort    = mainmp;

/* ora alloca la memoria per il secondo task ne inizializza */
/* il blocco di controllo */

met = (struct MyExtendedTask *)AllocMem(sizeof
            (struct MyExtendedTask), MEMF_PUBLIC | MEMF_CLEAR );

if(met == 0)
{
      /* Errore in AllocMem */
      DeletePort(mainmp);
      exit(40);
}

/* Ora inizializza la parte del blocco di controllo del task */
/* generato */

result = InitTask( (struct task *)met, "littletask",
                  PRIORITY, STACKSIZE );

if (result == 0)
{
      /* Errore in InitTask... non c'è memoria per lo Stack */
      DeletePort(mainmp);
      exit(45);
}
/* Legge l'indirizzo della porta di messaggio */

mp = &(met->met_MsgPort);
```

```
      /* inizializza la porta di messaggio per il task secondario */

      mp->mp_Node.ln_Type = NT_MSGPORT;/* porta di messaggio */
      mp->mp_Flags = PA_IGNORE;     /* quando il messaggio
                                    /* arriva non segnalarlo */
      NewList(&(mp->mp_MsgList));   /* inizializza la lista */
                                    /* di messaggio */

      /* Ora che è stata settata la porta di messaggio, possiamo */
      /* inviarvi un messaggio, possiamo lanciare il messaggio prima */
      /* o dopo aver aggiunto il secondo task */

      PutMsg(mp,&mymessage);

      AddTask( met, littletask, 0 );

      printf("\n Secondo task creato e lanciato");

      WaitPort(mainmp); /* attende la sua risposta */


/* Questo esempio dà per scontato che tutto sia OK. Se però ci
 * fosse un errore, si creerebbe un Loop realmente infinito
 * (perché il secondo task non invierebbe mai la risposta)
 * l'alternativa è la creazione di un Timer,e di richiamare
 *
 *
 * wakeupmask = Wait( TIMER_SIGNAL_BIT | REPLYPORT_SIGNAL_BIT );
 *
 * pertanto se non ci fossero risposte dopo un certo tempo, si
 * esamini met_Status per vedere cosa è andato storto al
 * povero task minore, e quindi fare qualcosa di conseguenza
 */

      GetMsg(mainmp);   /* rimuove il messaggio */
      printf("\n main: Il task secondario ha ricevuto il mio
                      messaggio\n");

cleanup:
      if(met) {
        /* rimuove il task secondario prima di terminare */
        FreeMem(met, sizeof(struct MyExtendedTask));
      }

      printf("\n main: Liberata la memoria usata dal task secondario");

      if(mainmp) {
        /* è la nostra porta di messaggio */
        DeletePort(mainmp);
      }
```

```
/* Delay(250);
 * Attende 5 secondi prima di uscire in modo che l'utente possa
 * vedere il nostro messaggio sulla Window del Workbench
 */

}     /*fine del main */
```

Listato 9.2

```
/* ********************************************************** */
/* InitTask.c -

1. Si usi della memoria allocata da qualcun'altro, e si faccia
   in modo che la liberi più tardi

2. Inizializza quella parte del blocco di controllo task
   necessaria (La stessa quantità inizializzata da CreateTask)

********************************************************************** */

/* Da un programma originale di Carl Sassenrath e Neil Katin;
* modificato per permettere al main di realizzare più
* inizializzazioni prima di scaricare il Task
*/

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/tasks.h"
#include "exec/execbase.h"


/*
 *  Inizializza un task dandogli un nome, una priorità e una
    certa dimensione di Stack. */

/* Sostegno per le MemEntries. Purtroppo questo è difficile da fare
 * in C : le MemEntris hanno unioni particolari, non possono
 * essere inizializzate staticamente...
 *
 * Per semplicità ho riprodotto qui le strutture MemEntry con
 * dimensioni appropriate. Copieremo questo poi in una variabile
 * locale e setteremo la dimensione dello Stack secondo ciò che viene
 * specificato dall'utente, poi tentiamo di allocare di fatto la
```

```
 * memoria
 */

#define ME_STACK        0
#define NUMENTRIES      1

struct FakeMemEntry {
    ULONG fme_Reqs;
    ULONG fme_Length;
};

struct FakeMemList {
    struct Node fml_Node;
    UWORD   fml_NumEntries;
    struct FakeMemEntry fml_ME[NUMENTRIES];
} TaskMemTemplate = {
    { 0 },                          /* Node */
    NUMENTRIES,                     /* numero di Entries */
    {                               /* attuali Entries: */
      { MEMF_CLEAR,     0 }         /* Stack */
    }
};

int
InitTask( task, name, pri, stackSize )
    struct task *task;
    char *name;
    UBYTE pri;
    ULONG stackSize;
{
    struct task *newTask;
    struct FakeMemList fakememlist;
    struct MemList *ml;

    /* arrotonda lo Stack a Word long... */
    stackSize = (stackSize +3) & ~3;

    /*
     * Questo allocherà un segmento di memoria
     * uno Stack di PRIVATE
     */
    fakememlist = TaskMemTemplate;

    fakememlist.fml_ME[ME_STACK].fme_Length = stackSize;

    ml = (struct MemList *) AllocEntry( &fakememlist );

    if( ! ml ) {
      return( 0 );
    }

    /* Setta il calcolo dello Stack */
    newTask = task;

    newTask->tc_SPLower = ml->ml_ME[ME_STACK].me_Addr;
```

```
    newTask->tc_SPUpper = (APTR)((ULONG)(newTask->tc_SPLower) + stackSi-
ze);
    newTask->tc_SPReg = newTask->tc_SPUpper;

    /* Strutture dati varie del task */
    newTask->tc_Node.ln_Type = NT_TASK;
    newTask->tc_Node.ln_Pri = pri;
    newTask->tc_Node.ln_Name = name;

/*  aggiungilo alla lista della memoria del task */
    NewList( &newTask->tc_MemEntry );
    AddHead( &newTask->tc_MemEntry, ml );

    return( 1 );
}
```

Listato 9.3

Pertanto, il task secondario viene lanciato da Proctest. Le istruzioni costituenti il codice di StartUp con le quali viene linkato, aspettano automaticamente un messaggio di StartUp proveniente dal Workbench prima di lanciare il programma. Utilizzando la stessa porta di messaggio usata all'inizio del process, esso torna nuovamente in stato di inattività in attesa di un messaggio che contiene delle informazioni specifiche -in questo, caso i parametri usati dal programma primario, di fatto le routine di gestione file Stdout e Stderr. Così questo process lanciato può effettuare il suo Output nella stessa Window dalla quale il process originale era stato aperto.

Un process è qualcosa di superiore a un task, e le varie routine dell'AmigaDOS, per poter girare, necessitano della presenza di un process con un blocco di controllo già inizializzato e delle informazioni ad esso associate. Questo esempio è stato fatto in modo da mostrare a un programmatore, che abbia bisogno di usare un process invece di un task, come un process possa essere costruito.

I File di Link per l'esempio di process

Per provare questo esempio, si dovranno compilare entrambi i programmi separatamente. I file di Link (da usare con il programma alink) vendono qui forniti. Ecco i vari passi da eseguire :

1. Dal proprio disco sorgente si scriva :

```
COPY process
COPY littleproc.c ram:
```

2. Con il disco dell'Amiga C nel Drive 1, si scriva :

```
CD df1: examples
```

3. Si usi un Text Editor qualsiasi per modificare il file di nome make nella directory degli esempi. Si cambi la linea che comincia con "lc2" in modo che cominci con "lc2-v". Questo fa si che sia disabilitato per questo esempio il controllo dello Stack.

4. Si scrivano le seguenti frasi:

```
EXECUTE make ram: process
EXECUTE make ram: littleproc

df1:c/alink with ram:process.with
df1:c/alink with ram:littleproc.with
```

Ecco è il contenuto del file di nome process.with, che è il primo file di Link :

```
FROM lib:Astartup.obj process.o
TO process
LIBRARY lib:amiga.lib
```

Ecco il contenuto del file di nome littleproc.with, che è il secondo file di Link:

```
FROM lib:Astartup.obj littleproc.o
TO littleproc
LIBRARY lib:amiga.lib
```

Il programma process

Per far girare i programmi dei listati 9.4 e 9.5, ci si assicuri che si trovino nella stessa directory perché proctest cercherà littleproc nella stessa directory da cui è partito. Esso caricherà e lancerà littleproc, poi lo scaricherà dal sistema, e terminerà.

```
/* proctest.c */

/* Versione del Software di sistema : V1.1 */

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/libraries.h"
#include "exec/ports.h"
#include "exec/interrupts.h"
#include "exec/io.h"
#include "exec/memory.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"

#include "workbench/startup.h"

#define PRIORITY 0
#define STACKSIZE 5000

extern struct Message *GetMsg();
extern int LoadSeg();
extern struct MsgPort *CreateProc();
extern struct MsgPort *CreatePort();

struct MyMess {
   struct Message mm_Message;
   int            mm_OutPointer;
   int            mm_ErrPointer;
};

#ifdef AZTEC_C
int stdout, stderr;
#else
extern int stdout;
extern int stderr;
#endif

main()
{
      struct Message *reply;
      struct process *myprocess;

      /* Messaggio inviato al process per risvegliarlo */

      struct WBStartup *msg;

      /* Messaggio contenente i miei parametri che dovranno essere
       * passati al process secondario (littleproc). Per dimostrare
       * che stiamo creando un process gli passiamo qualcosa che non sia
       * nullo : come stdout e stderr... infatti, dandogli i NOSTRI
```

```
       * valori possiamo condividere con esso la stessa Window
       */

      struct MyMess *parms;

       /* poiché il main() viene esso stesso lanciato come process,
       * ha una porta di messaggio allocata automaticamente. Essa è
       * locata a ((struct process *)FindTask(0))->pr_MsgPort
       */

      int littleSeg;

      /* Di fatto littleSeg è un BPTR, ma la dichiarazione int
       * soddisfa il compilatore e, poiché non usiamo tale valore,
       * ci limitiamo a passarlo così com'è
       */

      char *startname, *parmname;

      struct MsgPort *mainmp;
      /* puntatore alla porta di messaggio del primario */
      struct MsgPort *littleProc;
      /* puntatore alla porta di messaggio del secondario */

/* Fornisce i nomi ai messaggi che stiamo passando in modo da
 * controllare i messaggi restituiti alla porta di messaggio...
 * se lo desideriamo. */
      startname = "startermessage";
      parmname  = "parameterpass";

/* CARICA IL PROGRAMMA SECONDARIO DA LANCIARE ************** */

      littleSeg = LoadSeg("littleproc");
      if(littleSeg == 0)
      {
            printf("\n non trovo littleproc");
            exit(999);
      }

/* CREA UN PROCESS PER QUESTO SECONDO PROGRAMMA ************** */

      littleProc = CreateProc("littleguy",PRIORITY, littleSeg, STACKSIZE);
      if( littleProc == 0 )
      {
            printf("\nNon posso creare il process");
            UnLoadSeg(littleSeg);
            exit( 1000 );
      }

/* ************************************************************ */
/* Localizza la porta di messaggio allocata come parte del process */
/* che ha lanciato il main() all'inizio */

      myprocess = (struct process *)FindTask(0);
```

```
      mainmp = CreatePort(0,0);

/* ************************************************************** */
/* IL BLOCCO DI ISTRUZIONI CHE SEGUE LANCIA IL PROCESS
   COME SE FOSSE STATO RICHIAMATO DAL WORKBENCH */

/*
Infatti, poiché abbiamo creato il process nel modo mostrato, se si usa il
codice standard di StartUp, il process deve essere lanciato come se ri-
chiamato dalla Workbench. Esso è ora in attesa di un messaggio di StartUp.

Vi è infatti, un altro modo per richiamare un programma
caricato da disco, ma non comporta la creazione di un process. Esso uti-
lizza una chiamata diretta (come se fosse una subroutine) del programma.
Questo programma gira, in questo caso, sullo Stack di chi lo richiama,
pertanto il programma richiamante deve, di fatto, avere uno Stack suffi-
centemente grande per gestire due programmi. Inoltre gira sotto il pro-
cess del richiamante, pertanto il richiamante non ha più il controllo
finché questo secondo programma non termina. Tale programma opera poi un
return() o un exit() a favore del richiamante mediante un codice di ritor-
no appropriato

************************************************************** */

      /* Questo blocco di messaggio è una chiamata che risveglia */
      /* il programma caricato */
      msg = (struct WBStartup *)AllocMem(sizeof(struct WBStartup),
            MEMF_CLEAR);
      if(msg)
      {
      /* Setta gli argomenti necessari al passaggio del messaggio */

            msg->sm_Message.mn_ReplyPort = mainmp;
            msg->sm_Message.mn_Length = sizeof(struct WBStartup);
            msg->sm_Message.mn_Node.ln_Name = startname;

            /* Non si passano gli argomenti del Workbench a questo
             * process; noi non siamo WBench. Naturalmente, se
             * volessimo passare degli argomenti tipo WBench,
             * lo potremmo fare
             */

            msg->sm_ArgList = NULL;

            /* Se il process si aprisse senza una ToolWindow ( la
             * Workbench la setta) come  un Parent, esso continuerebbe
             * a far andare il suo main()... come mostrato in
             * Astartup.asm
             */
            msg->sm_ToolWindow = NULL;

            /* Invia il messaggio di StartUp */

            PutMsg(littleProc,msg);
      }
```

```
    else
    {
        printf("\n Non posso allocare lo spazio per WBStartup!\n");
        goto aarrgghh;    /* Oh no, un "goto"! */
    }
    /* ************************************************************** */
    /* Un semplice messaggio, stiamo ancora usando lo stesso
     * Message e la stessa porta di risposta.
     * Littleproc è un process cooperante... esso SA che deve
     * aspettare un messaggio in arrivo alla sua porta, il quale
     * contiene i parametri che esso deve usare per l'Output.
     * Il messaggio di StartUp viene gestito dal codice standard
     * di StartUp.
     * Questo messaggio-parametro è gestito dal programma stesso.
     * Il messaggio di StartUp viene restituito alla porta di
     * risposta da codice di StartUp, dopo che il programma
     * è terminato o ha operato un return
     */

    parms = (struct MyMess *)AllocMem(sizeof(struct
    MyMess),MEMF_CLEAR);
    if(parms)
    {
        parms->mm_Message.mn_ReplyPort = mainmp;
        parms->mm_Message.mn_Length = sizeof(struct MyMess);
        parms->mm_Message.mn_Node.ln_Name = parmname;

        /* SI NOTI CHE QUESTI SONO Astartup.asm stdout e stderr ;
         * l'esempio funziona solo se entrambi i programmi
         * vengono compilati e linkati con lo stesso codice di
         * StartUp
         */

#ifdef AZTEC_C
        stdout = Open("*", MODE_NEWFILE);
        stderr = Open("*", MODE_NEWFILE);
#endif
        parms->mm_OutPointer = (int)stdout;
        parms->mm_ErrPointer = (int)stderr;
        /* gli inviamo i nostri parametri */

        PutMsg(littleProc,parms);

        /* aspetta la risposta al passaggio dei parametri */

        WaitPort(mainmp);

        reply = GetMsg(mainmp);

        /* Il nome del nodo di messaggio dovrebbe contenere
         * l'indirizzo della stringa "parms" se vi è un
         * controllo degli errori.
         *
         * Si dovrebbero allocare porte separate per il
         * passaggio dei parametri, che non siano la porta
```

```
             * allocata automaticamente dal sistema all'inizio del
             * process.
             *
             *      ORA IL MAIN PUO' PROSEGUIRE E FARE QUALCOSA
             *      DI UTILE, E POI PUO' TORNARE A VEDERE SE IL
             *      PROCESS SECONDARIO HA FINITO  E DEVE ESSERE
             *      ESTROMESSO DAL SISTEMA
             *
             *
             *  Si aspetta il ritorno del messaggio wbstartup prima
             *  di permettere al primario stesso di terminare
             */

            WaitPort(mainmp);

            reply = GetMsg(mainmp);

            /* Il nome del nodo di messaggio */
            /* dovrebbe essere l'indirizzo di "startermessagge" */

/* NOTA BENE : bisognerebbe controllare se il messaggio
* ricevuto a questa porta era la stringa, o la chiamata
* di risveglio. Questo semplice programma da per scontato
* che la stringa sia ricevuta e restituita per prima, poi
* viene restituita la chiamata di risveglio quando
* littletask sta finendo
*/

            UnLoadSeg(littleSeg);
            printf("\nIl secondario termina; il Master scarica
            le sue istruzioni e i suoi dati\n);
      }
      else
      {
            printf("\nNon posso allocare memoria per il messaggio-parametro\n");
      }
    aarrgghh:
            /* arriva qui in ogni caso */

      if(mainmp){
                  DeletePort(mainmp);
               }
      if(parms) { FreeMem( parms, sizeof(struct MyMess));    }
      if(msg)   { FreeMem( msg,   sizeof(struct WBStartup)); }
#ifdef AZTEC_C
      Close(stdout);
      Close(stderr);
#endif

}     /* fine del main */
```

Listato 9.4

Si noti che questi programmi sono stati scritti per essere usati con il codice di StartUp di Amiga (AStartup.obj) piuttosto che con il codice di StartUp della Lattice (Lstartup.obj). Non è stato fatto alcun tentativo di per tentare di adattarli al codice della Lattice. L'unico punto di incompatibilità dovrebbe essere nell'uso di Stdout e Stderr. Essi vengono definiti diversamente dalla Lattice. Essa, infatti, li definisce come l'indirizzo di un blocco di I/O. Se il process principale e quello secondario viene compilato con il compilatore della Lattice allora i valori passati a Stdout e Stderr saranno compatibili.

Lo Stdout e lo Stderr dell'AmigaDOS sono puntatori del linguaggio BCPL a una struttura dati dell'AmigaDOS. Quando si usa amiga.lib per fornire la funzione printf al momento del linkaggio, la versione di printf dell'amiga.lib usa al proprio interno la versione AmigaDOS di Stdout.

```
/* littleproc.c ************************************* */

/* Semplice servoprogramma per creare un nuovo process */

/* Versione del Software di sistema : V1.1             */

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/libraries.h"
#include "exec/ports.h"
#include "exec/interrupts.h"
#include "exec/io.h"

#include "libraries/dos.h"
#include "libraries/dosextens.h"

#include "workbench/startup.h"

/* Stanno per essere forniti al servoprogramma dal richiamante */
/* essi sono di fatto definiti in Astartup.asm */

#ifdef AZTEC_C
int stdout;
int stderr;
#define printf ourprintf
#else
extern int stdout;
extern int stderr;
#endif

struct MyMess {
   struct Message mm_Message;
```

```
    int mm_OutPointer;
    int mm_ErrPointer;
};

extern struct Message *GetMsg();
extern struct task *FindTask();
extern struct FileHandle *Open();

main()
{
      struct MyMess *msg;
      struct MsgPort *myport;
      struct process *myprocess;

      struct FileHandle *myOwnOutput;

      myprocess = (struct process *)FindTask(0);

      myport = &myprocess->pr_MsgPort;

      /* Aspetta che il richiamante invii un messaggio. Il semplice
       * messaggio ha i propri stdout e stderr, pertanto si può
       * utilizzare la stessa Window CLI da cui siamo partiti
       */

      WaitPort(myport);

      msg = (struct MyMess *)GetMsg(myport);

      stdout = msg->mm_OutPointer;

      /* Si prova printf per provare che si tratta realmente
       * di un process... un task non potrebbe farlo senza
       * impiantarsi
       */
      printf("\nEccomi, sono il servoprogramma che hai lanciato!!!");
      printf("\nOra sto per aprire una Window tutta MIA\n");
      Delay(100);

      /* ORA POTREBBE FARE QUALCOSA DI UTILE PER IL PROGRAMMATORE...
       * QUALSIASI COSA SI DESIDERI
       */

      myOwnOutput = Open("CON:140/50/300/80/SlaveProcess",MODE_NEWFILE);
      if(myOwnOutput == 0)
      {
            ReplyMsg(msg);    /* dice al primario che ha finito */
            exit(0);    /* non può ritornare alcun codice di errore */
      }
      else
      {
            stdout = (int)myOwnOutput;
            /* resetta la gestione File */
      /* fa in modo che il servoprogramma scriva sulla sua Window */
            printf("Guarda, posso usare l'AmigADOS!");
```

```
            Delay(250); /* 250/50 = 5 secondi */
            stdout = msg->mm_OutPointer;
            Close(myOwnOutput);
            ReplyMsg(msg);
      }

      /* Ora semplicemente finisce e ritorna al codice di StartUp */
      /* uscendone presumibilmente pulito */

}

#ifdef AZTEC_C
/* un piccolo ricollocamento printf per operare un echo alla */
/* gestione di Output() */
ourprintf(s) char *s; {
      return(Write(stdout, s, strlen(s)));
}
#endif
```

Listato 9.5

Se si linka con lc.lib specificata come prima Library, allora quel file link usa la sua interpretazione di stdout, che non sarà compatibile con i valori che riceve dalla funzione Open (Open è una funzione dell'AmigaDOS, open è della Lattice).

Si noti anche che l'esempio utilizza la porta di messaggio riservata dall'AmigaDOS per il process per potervi ricevere dei messaggi DOS di I/O. Se si trova questo esempio interessante e utile, si dovrà probabilmente inizializzare una porta di messaggio separata per i i messaggi che si ritengono necessari invece di spartire la stessa porta di messaggio con l'AmigaDOS. Rubare la porta di messaggio dell'AmigaDOS è un modo semplice e conveniente per far funzionare le cose in questo caso. Se si necessita di un I/O esteso, è sicuramente consigliabile creare una porta di messaggio aggiuntivo per i propri scopi.

Come nota finale circa questo esempio di processing diciamo che si poteva facilmente creare un process aggiuntivo mediante il comando Execute dell'AmigaDOS:

```
success = Execute("qualcheprogramma",0,0);
```

Ma questo esempio era chiaramente rivolto a chiarificare il settaggio dell'intercomunicazione tra process e il passaggio di messaggi.

Comunicazione tra task

Per la comunicazione e l'interelaborazione tra task, il programma mostrato usava delle porte di messaggio esplicitamente create allo scopo. Se si hanno però due task lanciati in modo completamente indipendente, allora vi sono parecchi modi per far si che un task riesca a trovare e contattare un altro task preesistente e riesca quindi a comunicare in modo interattivo con questo task che sta girando indipendentemente da esso.

Trovare un task

Il sistema mantiene parecchie liste differenti, compresa una lista dei task e una delle porte. Quando si lancia un task mediante la funzione CreateTask, la funzione stessa pone il nome del task all'interno del nodo del blocco di controllo task. Poi, si potrà utilizzare la funzione FindTask per localizzarlo.

Nel listato 9.3, alla funzione InitTask veniva fornito un nome attraverso il quale essa poteva trovare il blocco di controllo. Il nome era littleguy. Invece di tenere in memoria l'indirizzo del blocco di controllo creato dal programma primario, il primario avrebbe potuto aggiungere il task al sistema, e poi trovarlo (e quindi trovare la porta di messaggio) nel seguente modo:

```
struct *taskblock;
   /* un puntatore a un blocco di controllo task */

taskblock = FindTask("littleguy");
```

Così, qualunque cosa si voglia associare direttamente al blocco di controllo task può essere trovato dopo che il task stesso è stato aggiunto alla lista di sistema dei task. Comunque, nel listato 9.3, poiché è il task primario a creare il blocco di controllo, il programma sa dove si trova il blocco stesso.

Trovare un process

Sfortunatamente trovare un process non è così facile come trovare un task. Nonostante l'AmigaDOS (nella Release 1.1) utilizzi la lista dei task per tenere traccia dei process, esso non inizializza il campo name del task con il quale riferirsi al process generato. Pertanto non è possibile identificare un process che stia girando nel sistema, esaminando i nomi sulla lista dei task.

Perciò, se si deve identificare un certo programma al quale si debbono passare dei dati o con il quale si deve comunicare in qualche modo, allora è consigliabile usare un task piuttosto che un process.

Trovare le porte

Se si crea e si dà un nome a una porta di messaggio e la si aggiunge alla lista di sistema, allora qualsiasi task o qualsiasi process sarà in grado di trovare la porta più tardi mediante la funzione FindPort:

```
myport = FindPort("unaporta");
```

Se la funzione ritorna un valore NULL, allora non vi è alcuna porta con quel nome nella lista di sistema. Se il valore ritornato non è NULL, allora myport punta al nodo di lista della porta di messaggio avente il nome trovato nella lista di sistema delle porte.

Dovetti usare questa funzione per un programma grafico dimostrativo dove sei diversi programmi utilizzavano uno schermo costituito da quattro piani di bit per dare la dimostrazione di una delle tante funzioni grafiche disponibili su Amiga. Uno di questi programmi è chiamato Rectangles; esso crea una varietà di rettangoli colorati in una window grafica. Un'altro è chiamato Lines e disegna delle linee casuali multicolori. Un terzo è chiamato Wallpaper e, anch'esso, disegna dei rettangoli multicolori.

Gli schermi Custom possono occupare un sacco di memoria, pertanto decisi di far sì che ogni programma controllasse se vi era un altro programma dello

stesso tipo che stesse girando parallelamente. Se un altro programma aveva già creato lo schermo personalizzato, allora invece di aprire un nuovo schermo Custom, creava una propria window nello schermo già aperto. Ogni finestra possedeva il proprio gadget di chiusura, con il quale si poteva terminare il programma. Ogni window aveva la propria barra di controllo, sulla quale erano listati i nomi tutti i programmi che utilizzavano contemporaneamente lo stesso schermo.

Per poter mettere in comunicazione i vari programmi, creai una versione personalizzata della porta di messaggio:

```
typedef struct {
   struct MsgPort normalMsgPort;
   int usersOfScreen;
   struct Screen *screen;
   } MYCUSTOMPORT;
```

Inizializzando in modo appropriato normalMsgPort, tale porta poteva essere aggiunta alla lista di sistema delle porte utilizzando la funzione AddPort. Si ricordi che il sistema non si preoccupa di quanto sia lunga tale lista, purché i vari elementi siano inizializzati correttamente.

Il primo programma (cioè il primo del gruppo che veniva caricato e lanciato) avrebbe allocato la memoria per questo schermo lo avrebbe aperto, ne avrebbe copiato l'indirizzo nella struttura dati della porta personalizzata e avrebbe inizializzato UsersOfScreen a 1. Si poteva a questo punto aggiungere questa porta alla lista di sistema e aprire una Window sul nuovo schermo.

I programmi che fossero stati aperti successivamente avrebbero potuto trovare la porta, utilizzando FindPort, e mediante il valore di UsersOfScreen, calcolare dove porre la propria window in modo da non oscurare completamente l'output grafico dei programmi già aperti in quello schermo. Ogni nuovo programma che trovasse la porta e aprisse una window avrebbe incrementato il valore di UsersOfScreen.

Ogni qualvolta un programma avesse chiuso la propria window, sarebbe stato decrementato il valore di UserOfScreen. Quando tale valore raggiungeva zero, a prescindere da quale fosse, l'ultimo programma avrebbe cancellato lo schermo, cancellato la porta, e sarebbe terminato esso stesso.

Il listato 9.6 mostra segmenti di programmi che sono parti dei programmi grafici sopra menzionati. Queste parti dimostrano come la porta Custom serva da punto di incontro di tutti i programmi, quando tutti stanno girando contemporaneamente nel sistema, alla ricerca di uno schermo sul quale istallarsi.

```
/* frammento di programma numero 1 - porta personalizzata */

/* Si noti che non tutte le dichiarazioni vengono fatte qui. Questo
 * esempio viene fornito al potenziale utilizzatore del multitasking
 * perché si possa fare un'idea
 */

struct MyPort {                 /* porta di messaggio personalizzata */
      struct MsgPort mp;        /* porta di messaggio standard */
      struct Screen *Screen;    /* dove è il mio schermo */
      int Users;                /* quanti programmi stanno usando lo schermo */
      int RangeSeed;            /* quale è il valore corrente del
                                 * generatore
                                 * di numeri Random per il l'utente più
                                 * recente... altrimenti RangeSeed parte
                                 * da zero per TUTTI
                           */
      char portname[64];
      char screenname[64];
};

struct Screen *
Setup()                    /* Ritorna un puntatore ad uno schermo nuovo
                            * o ad uno preesistente
                            */
{
    int junk;
    struct MyPort *myp, *sysmyp;
    struct Screen *screen;
    GfxBase = OpenLibrary("graphics.library", 0);
    if (GfxBase == NULL)
    {
      problem = 1001;          /* Non si può aprire la Library gfx */
        return(0);
    }
    IntuitionBase = OpenLibrary("intuition.library", 0);
    if (IntuitionBase == NULL)
    {
      problem = 1002;    /* Non si può aprire la Library intuition */
        CloseLibrary(GfxBase);
        return(0);
    }

/* Lo schermo Custom è già stato aperto da qualche altra
 * applicazione? Se è così, usalo, altrimenti aprilo.
 * Ferma il multitasking.
```

```
 * Il controllo dell'esistenza della porta legata allo schermo
 * può avvenire solo per un programma alla volta, perché se
 * non esiste deve essere creata
 */


/* Comincia la creazione di una porta di messaggio Custom per lo
 * schermo, nel caso che il sistema non l'abbia ancora. In questo
 * modo l'avrà pronta e potrà aggiungerla e non dovrà operare
 * Disable() per un tempo lungo
 */

    myp = (struct MyPort *)AllocMem(sizeof(struct MyPort),MEMF_CLEAR);
    if (myp == 0)
    {
      CloseLibrary(IntuitionBase);
      CloseLibrary(GfxBase);

      return(0);
    }
    else
    {
      /* Setta i parametri della porta */

      myp->mp.mp_Node.ln_Pri = 0;
      myp->mp.mp_Node.ln_Type = NT_MSGPORT;
      NewList(&(myp->mp.mp_MsgList));

      /* Fissa il nome della porta, dentro la porta stessa */

      strcpy( &(myp->portname[0]) , "lowres.16.color" );
      myp->mp.mp_Node.ln_Name = &(myp->portname[0]);

      /* Fissa l'intestazione dello schermo, nella porta */

      strcpy( &(myp->screenname[0]), "TestScreen" );
      ns.DefaultTitle = (UBYTE *)&(myp->screenname[0]);

      /* Numero di utenti dello schermo */

      myp->Users = 1;     /* Un solo utente per ora, appena aperta */

      /* Calcola un valore qualsiasi per modificare RangeSeed */

      junk = RangeRand(100);

      /* Valore di partenza per il generatore di numeri Random */
      /* per l'utente successivo */

      myp->RangeSeed = RangeSeed;

      /* **** Inizio della sezione di Interrupt disabilitati ******** */

      Disable();
      /* impedisce l'alternanza dei task durante questa operazione */
```

```
* Siamo pronti ad aggiungere la nostra porta Custom al sistema.
 * c'è già qualcosa di simile? Se è così, disalloca la nostra.
 * se no, apri uno schermo Custom, e finisci di inizializzare
 * la nostra porta con il suo indirizzo. Poi aggiungila alla
 * lista di sistema
 */
     sysmyp = (struct MyPort *)FindPort("lowres.16.color");
     if(sysmyp == 0)
     {
               screen = OpenScreen(&ns);
               if (screen == NULL)
               {
               problem = 1003;     /* Non si può aprire lo schermo */
              Enable();
          /* Riabilita gli Interrupt e l'alternanza dei task */

              FreeMem(myp, sizeof(struct MyPort));
              CloseLibrary(IntuitionBase);
              CloseLibrary(GfxBase);
              return(0);
               }

          /* Mostra dove trovare lo schermo appena aperto */

        myp->Screen = screen;

        /* Lo aggiunge al sistema in modo che gli altri lo trovino */

        AddPort(myp);
        Enable();
                /* Riabilita gli Interrupt e l'alternanza dei task */
        return(screen);
     }
     else        /* Il sistema HA già questa porta Custom */
     {
        /* Se il sistema ha una porta simile, non ha */
                /* bisogno della nostra */

        FreeMem(myp, sizeof(struct MyPort));

        /* Aggiunge 1 al numero degli utenti */
        sysmyp->Users += 1;

        /* Prende il valore RangeSeed dell'utente precedente */
        RangeSeed = sysmyp->RangeSeed;
        junk = RangeRand(100);    /* Cambia il valore di RangeSeed */

        /* Salva un nuovo valore per il prossimo utente*/
        sysmyp->RangeSeed = RangeSeed;
        Enable();                 /* Abilita l'alternanza */

        /* E ritorna la locazione dello schermo */
        return(sysmyp->Screen);
       }
```

```
    }
}

/* Frammento di programma numero 2 */

/* Chiude la Window Custom e decrementa il numero degli utenti
 * dello schermo. Se il valore raggiunge zero, cancella lo schermo,
 * la porta, libera la memoria e chiude lo schermo
 */

int
EndTest()
{
    struct MyPort *myp;
    if (w != NULL) {
      ClearMenuStrip(w);
      CloseWindow(w);
    }
    Forbid();
            /* interrompe momentaneamente l'alternanza dei task */
    myp = (struct MyPort *)FindPort("lowres.16.color");
                              /* Dovrebbe seguire ciò */
    if(myp)
    {
      if((myp->Users -= 1)==0)
      {
                if (myp->Screen != NULL) CloseScreen(myp->Screen);
          RemPort(myp);
          FreeMem(myp, sizeof(struct MyPort));
      }
    }
    Permit();                         /* Riabilita l'alternanza dei task */
    return(TRUE);}
```

Listato 9.6

Le capacità multitasking di Amiga offrono grosse possibilità a chi volesse sviluppare del software. I programmi qui forniti hanno solo mostrato la superficie di questo sistema. Spero, comunque, che questo capitolo abbia dato una veduta d'insieme, anche se parziale, di come operi il sistema multitasking, e quindi un punto d'appoggio per uno sviluppo futuro di programmi in questo senso.

Appendice A

II Text Editor (ED)

Il programma ED è un semplice editor di testi che può essere usato per creare i file sorgente per il compilatore C. Per coloro che siano poco avvezzi all'uso di un editor di testi per creare un programma, questa appendice sarà un'utile guida.

Una volta lanciato ED, esso si presenta con uno schermo vuoto sul quale è possibile inserire linee di testo. Una volta finita la digitazione di queste linee, esse possono essere salvate in un file testo.

L'ED è di fatto un editor di stringhe, non un Word Processor. Quando lo si usa si deve ricordare che, nonostante si presenti con uno schermo di testo sul quale lavorare, esso considera ogni singola linea come un'entità individuale. Per esempio, un blocco di testo può essere costituito da una o più linee. Un segnalatore di blocco di testo, in ED, non può essere posto nel mezzo di una linea. Similmente, quando si sposta o si copia un blocco, ED lo inserisce tra la linea dove si trova correntemente il cursore e la linea immediatamente soprastante.

Prendendo confidenza con l'uso dell'ED, si può tendere a usare un numero sempre maggiore di comandi. Però, all'inizio, bisogna solo ricordare alcune regole basilari e alcuni comandi fondamentali per poterlo effettivamente usare. Ecco alcuni punti da ricordare :

- Ci si trova sempre in condizione di inserimento. Ovunque sia localizzato il cursore, se si digita un carattere permesso (non un comando), l'ED inserirà quel carattere in quella precisa posizione e spingerà in avanti tutto ciò che si trova alla sua destra. Se viene dato un Return a metà di una linea, la linea viene spezzata.
- I tasti del cursore funzionano nel solito modo. Ci si può muovere all'interno del file solo grazie a questi tasti.
- Il tasto di BDelete (BackSpace) cancella il carattere immediatamente a sinistra del cursore.
- Il tasto Esc viene usato per inserire i comandi. Se si preme Esc, il cursore si sposta temporaneamente in fondo allo schermo e attende che si inserisca un comando.

Per lanciare l'ED, ci si assicuri che il Workbench o il CLI siano nel drive interno. Questo disco non deve essere protetto per permettere a ED di creare un file di lavoro nella directory SYS:T. Dalla window del CLI si scriva:

```
ED hello.c
```

e si prema Return. Si aprirà una nuova window e l'ED segnalerà:

```
Creating a new file
```

Il cursore ora è in cima alla window. Si inseriscano le seguenti linee esattamente come mostrato, premendo Return alla fine di ognuna di esse:

```
main()
{
   printf("\nCiao a tutti\n");
}
```

Poi si prema Esc X e si dia il Return. Ora il programma è stato salvato in modo che possa essere letto successivamente dal compilatore.

La tabella A.1 riassume i comandi dell'ED. Nella tabella, la notazione ^ indica che si deve premere il tasto Ctrl e poi premere la lettera specificata per eseguire quel particolare comando. Nonostante la lettera appaia in maiuscolo, può anche essere inserita come minuscolo, non cambia nulla.

Comandi di spostamento del cursore

<fs>	sposta in alto di una linea
<fg>	sposta in basso di una linea
<fr>	sposta a destra di un carattere
<fl>	sposta a sinistra di un carattere
^I	(Tab) sposta alla posizione successiva di tabulazione
^R	sposta alla fine della parola precedente
^T	sposta all'inizio della parola precedente
^D	opera uno scroll verso il basso

^U	opera un scroll verso l'alto
^E	sposta in fondo o in cima allo schermo
^]	sposta all'inizio o alla fine della linea
^M	(Return) sposta di una linea in basso sul margine destro
Esc B	sposta alla fine del file
Esc T	sposta all'inizio del file
Esc N	sposta all'inizio della linea successiva
Esc P	sposta all'inizio della linea precedente
Esc CE	sposta alla fine della linea corrente
Esc CB	sposta all'inizio della linea corrente
Esc M <Line-number>	sposta a un determinato numero di linea del file

Comandi di inserimento e cancellazione

^A	inserisce una linea dopo quella corrente
^B	cancella la linea nella quale è locato il cursore
^H	(BackSpace) cancella il carattere a sinistra della posizione del cursore e sposta tutto indietro di uno spazio
Del	cancella il carattere sul quale si trova il cursore
^O	se il cursore si trova su di un carattere vuoto (spazio), allora cancella gli spazi sino alla successiva parola della linea. Se il cursore non si trova su di un carattere vuoto, allora lo cancella assieme agli altri caratteri a destra finché non trova uno spazio
^Y	cancella la fine della linea, compreso il carattere sul quale si trova
Esc A / *ga/*	inserisce questa stringa sulla linea precedente quella corrente
Esc I / *ga/*	inserisce questa stringa come se fosse una linea seguente quella corrente
Esc D	cancella la linea corrente
Esc DC	cancella il carattere sul quale si trova il cursore

Esc IF !pathname!	inserisce il file con questo nome nella posizione corrente del cursore

Comandi di Tabulazione e impostazione margini

Esc SL	setta il margine sinistro alla colonna specificata nel parametro
Esc SR	setta il margine destro alla colonna specificata nel parametro. (Il Default è 80. Il massimo è 255, perché questo è il massimo valore per la lunghezza di una linea ammesso da ED, e dall'AmigaDOS)
Esc ST	setta la distanza di tabulazione al valore del parametro
Esc EX	estende il margine destro (come sulle macchine da scrivere)

Comandi di Find e Replace

Esc F / *ga/*	cerca in avanti la stringa specificata
Esc BF / *ga/*	cerca all'indietro la stringa specificata
Esc E / *ga1//* *ga2/*	rimpiazza la stringa1 con la stringa2, senza conferme
Esc EQ / *ga1//* *ga2/*	localizza la stringa1 e chiede una conferma, se è OK allora la rimpiazza con la stringa2
Esc LC	tratta i caratteri maiuscoli e minuscoli in modo differente nella ricerca

Comandi riguardanti il blocco di testo

Esc BS	segna questa linea come la prima del blocco
Esc BE	segna questa linea come l'ultima del blocco
Esc DB	cancella l'intero blocco
Esc IB	copia il blocco segnato alla posizione corrente del cursore

Esc SB	mostra la prima linea del blocco segnato come prima linea dello schermo (permette di spostarsi velocemente in una posizione segnata all'interno del file)
Esc WB !pathname!	scrive il blocco segnato su di un file specificato. Se il nome del Path non è quello attuale, lo si può specificare completamente, compresi gli slash (i punti esclamativi vengono usati per delimitare il nome del path)
Comandi di Save e di Quit	
Esc SA	salva il file con il nome corrente e continua l'Edit
Esc Q	termina senza salvare nulla di ciò che è stato modificato. L'ED chiede conferma
Esc X	termina salvando tutti i cambiamenti apportati al file attuale
Comandi Vari	
Esc J	unisce la linea corrente e quella successive in un unica linea
Esc S	spezza la linea corrente nella posizione del cursore. Trasforma il carattere alla posizione corrente del cursore nel primo carattere della linea successiva
Esc SH	mostra lo status dell'editor
Esc V	ridisegna lo schermo

Tabella A.1

Si noti che sia Esc SA sia Esc X accettano un nome diverso di quello usato per aprire il file. Essi si scrivono:

```
Esc X !athname!
```

oppure

```
Esc SA !athname!
```

Questi comandi possono essere utili per salvare forme intermedie di ciò che si sta inserendo.

Una volta che è stato premuto il tasto Esc, la linea di comando può contenere comandi multipli, purché separati da un punto e virgola. Per esempio:

```
F /una certa frase/; E /vecchiaparola/nuovaparola/
```

cerca una certa stringa, "una certa frase", e poi, trovatala, cerca al suo interno "vecchiaparola" e la sostituisce con "nuovaparola".

Si possono specificare quante volte si vuole che un certo comando (o una linea di più comandi) venga ripetuto piazzando un numero davanti al comando stesso. Oppure, invece del numero, si può specificare RP, che ripete tale comando sino alla fine del file.

Appendice B

Creare e utilizzare un file Make

In questa appendice viene spiegato come utilizzare il compilatore C Amiga, che è un derivato di quello della Lattice. Quei lettori che hanno già familiarità con il compilatore della Lattice su altri sistemi non troveranno grosse difficoltà ad adattarsi a questa versione.

Ho cercato di non introdurre nel libro, per quanto mi è stato possibile, parti riguardanti le caratteristiche specifiche del compilatore però per avere una base comune ho sempre usato il compilatore Amiga C per tutti gli esempi. E' chiaro che se si usano altri compilatori potrebbe essere necessario adattare i programmi del libro perché girino correttamente. Comunque tutti gli esempi sono stati mantenuti corti e diretti, nel limite del possibile, per facilitare un tale compito.

Uso del compilatore Amiga C

Una volta creati i sorgenti (mediante Ed, MicroEmacs, o altri text editor) vi sono tre fasi separate da eseguire prima di poter ottenere un programma funzionante.

Prima fase di compilazione

La prima fase è chiamata LC1. Attraverso di essa, il sorgente viene tradotto in un codice intermedio di compilazione. E' proprio durante questa fase che vengono controllate la strutturazione e la sintassi. Un tipico modo per lanciare la prima fase consiste nello scrivere questa linea di comando:

```
CD DF1:C ;si sposta nella directory di comando del disco dell'Amiga C

LC1 -i:include/df0:hello.c to df0:hello.q1
```

(si presume di operare su un Amiga avente 512K di ram e un drive esterno contenente il disco dell'Amiga C e si presume che il disco con il programma da compilare si trovi nel drive interno e si chiami hello.c.)

Questa sequenza di comandi crea un file di nome hello.q sul disco interno nella directory principale. La frase

```
-i:include/
```

dice al compilatore in quale directory vanno cercati i file che devono essere inclusi durante la compilazione; i due punti stanno ad indicare la directory principale del disco (df1:) e /include rappresenta il prefisso per la ricerca di un qualunque file Include da inserire nella fase di compilazione.

Il file creato verrà chiamato automaticamente hello.q (a meno che non si specifichi un nome diverso). Quì è stato specificato per maggior chiarezza. L'estensione .q sta per file quad, che è il nome usato dall'Amiga C per riconoscere i file generati dalla prima fase di compilazione.

Seconda fase di compilazione

La seconda fase è chiamata LC2. Durante tale fase, il linguaggio intermendio prodotto dalla prima fase viene tradotto in codice oggetto. Dando per scontato che il file quad sia stato prodotto correttamente, si può lanciare la fase due inserendo questa linea di comando

```
LC2 df0:hello.q to df0:hello.o
```

Questo produce il file oggetto corrispondente al sorgente originale.

Terza fase di compilazione

Il file oggetto non rappresenta però la versione finale del programma. Esso contiene tutte le istruzioni fornite dal programmatore tradotte in linguaggio macchina, ma non contiene il codice corrispondente a tutte le routine che sono state invocate. Per creare un programma completamente funzionante, si deve eseguire un'ultima fase, chiamata ALINK, che unisce il codice del file oggetto con il co-

dice di tutte le funzioni di libreria usate dal programmatore (debug.lib o amiga.lib) e con il codice corrispondente a programmi precedentemente compilati.

A questo punto si evidenzia il vantaggio di utilizzare un linguaggio ad alto livello come il C. Il risultato di tutti questi sforzi è infatti un programma che consiste in un codice oggetto completamente rilocabile che l'AmigaDos può caricare in qualunque punto della memoria del sistema e può far girare come un process insieme agli altri task nel sistema. Pertanto, qualunque programma creato è in grado di condividere le risorse del computer con altri task che girano contemporaneamente.

Una volta completate con successo le prime due fasi di compilazione, si può invocare la fase ALINK con un comando di questo tipo

```
ALINK FROM df0:hello.c + Lstartup.obj to df0:hello

LIBRARY LC.LIB,AMIGA.LIB,DEBUG.LIB
```

(Si scriva tutto ciò sulla stessa linea di comando). Se tutto funziona correttamente, il risultato della sequenza delle tre fasi è un programma che si può lanciare semplicemente scrivendo:

```
cd df0:

hello
```

Sommario delle operazioni di compilazione

Ecco un breve sommario che raccoglie ciò che è necessario fare per compilare un programma con il compilatore C dell'Amiga:

1. Si crei il sorgente usando un qualsiasi text editor

2. Si esegua la fase uno di compilazione (LC1) che crea un file avente l'estensione .q.

3.Si esegua la fase due di compilazione (LC2) che crea un file avente l'estensione .o.

4. Si esegua la fase tre di compilazione (ALINK), che linka il programma con le funzioni di libereria e con i programmi precedentemente compilati.

La quantità di comandi e di tempo necessaria a compiere separatamente le tre fasi di compilazione può sembrare decisamente alta, soprattutto se paragonata alla complessità, veramente minima, del programma che è stato compilato. L'Amiga C, però, comprende un file che permette di evitare questa noiosa sequenza. Questo particolare file Execute viene detto file MAKE perché può essere usato per creare, o dirigere la creazione, degli altri file. Il nome di questo file è proprio MAKE e si trova nella directory degli esempi sul disco del C. (Nelle prime versioni può essere stato chiamato Makesimple, oppure può darsi che su determinati dischi il file MAKE compia solo le prime due fasi della compilazione. Un file di nome link compie la terza. Se si sta creando un programma composto da parecchi file C, si dovrebbe fare in modo di compilarli separatamente, e poi si dovrebbero linkare i vari file, una volta che sono stati tutti compilati. Ecco una tipica sequenza di operazioni utilizzabili per compilare uno qualsiasi dei programmi di esempio presente in questo libro:

1. Si inserisca il disco della CLI nel drive interno. Si inserisca il disco contenente il programma da compilare nel drive esterno. Poi si scriva il comando seguente

```
copy hello.c to ram:
```

2. Una volta finito il caricamento, si rimuova il disco contenente il programma e si inserisca il disco dell'Amiga C nel drive esterno. Si dia il comando

```
cd df1:
```

Questo dice all'AmigaDOS che il drive esterno deve essere usato come directory primaria per il prelevamento di qualsiasi comando e qualsiasi file.

3. Si dia il comando

```
execute examples/makesimple ram:hello
```

Con questo singolo comando, si chiede all'AmigaDOS di usare il programma MAKE per eseguire tutte e tre le fasi di compilazione.

L'AmigaDOS fa si che venga compilato il file hello.c che si trova nella ramdisk (la directory ram:) e, nel caso la compilazione abbia avuto successo, produce un programma eseguibile di nome hello (anche esso nella directory ram:). Per provare il programma, si deve soltanto scrivere

```
run ram:hello
```

```
ram:hello
```

Poi, per salvare il programma, si tolga il disco dell'Amiga C dal drive esterno, e vi si inserisca il disco di partenza del programma dando il seguente comando

```
copy ram:hello to df1:hell
```

A questo punto il programma eseguibile si trova sul disco inserito nel drive esterno. Sarà bene però dare un'occhiata più da vicino al contenuto di un tipico file MAKE. Comprendendo cosa fa, il lettore sarà in grado poi di generare il proprio file MAKE per fare ciò che desidera.

Contenuto di un file MAKE

Il programma di esempio MAKE è, di fatto, un file contenente dei comandi eseguibili da AmigaDOS. Quando si da il comando

```
execute <nomeFile>
```

l'AmigaDOS carica il comando EXECUTE e preleva i comandi in input di cui necessita direttamente da questo file, come se fosse l'utente a inserirlo. In più, EXECUTE può operare una sostituzione di parametri in modo da poter eseguire lo stesso file di comandi per più programmi diversi, come se stesso eseguendo realmente file diversi. I file contenenti dei comandi che possono essere eseguiti da EXECUTE vengono detti file Execute.I file Execute possono contenere parecchi tipi diversi di linee di input. tra questi troviamo i commenti, i comandi di sosti-

tuzione dei parametri, e i comandi per l'esecuzione di altri programmi. Si noti che se la prima colonna di ogni riga contiene un punto e virgola o un punto, l'AmigaDOS tratterà tale linea come un commento. Il comando EXECUTE tratta una linea che inizia con un punto e virgola come un commento e invece una linea che inizia con un punto come un comando.

Sostituzione dei parametri in un file Execute

Si può indicare al comando EXECUTE che deve prepararsi a sostituire dei parametri attraverso l'uso dell'istruzione key. Quando si fornisce questa istruzione, l'EXECUTE fa una copia del file di comando fornitogli e la pone nella directory T: del disco corrente, preleva i parametri dalla linea di comando immediatamente seguenti il nome del comando stesso, e sostituisce la stringa fornitagli all'interno del file di comando nella posizione in cui veniva richiesta una sostituzione di parametri. Si immagini di avere un file di comando di nome typeit, che contiene le seguenti linee :

```
.key primopar,secondopar
IF EXISTS "<primopar>"
TYPE "<primopar>"
ENDIF
IF EXISTS "<secondopar>"
TYPE "<secondopar>"
ENDIF
```

Se si digita il comando

```
execute typeit nomeFile1 nomeFile2
```

allora il comando EXECUTE crea una copia su disco del file di comando, sostituendovi la stringa rappresentante il primo parametro, qui chiamata nomeFile1, ogni volta che trova <primopar> (la specificazione del nome del parametro dell'istruzione key), e la stringa rappresentante il secondo parametro ogni volta che trova <secondpar> :

```
IF EXISTS NOMEFILE1
TYPE NOMEFILE1
ENDIF
IF EXISTS NOMEFILE2
```

```
TYPE NOMEFILE2
ENDIF
```

Questo File viene poi eseguito come se fosse inserito da tastiera. Questo semplice File di Type dice in parole povere : "Cerca se c'è un certo file e, se lo trovi, stampalo".Lo stesso principio è applicabile al compilatore C. Si usi ED, o Microemacs, o un qualsiasi Text Editor per vedere il contenuto di examples/MAKE sul disco dell'Amiga C. Si vedrà che l'istruzione key contiene i parametri per i comandi di controllo del compilatore LC1 e LC2 e una linea di comando per il comando ALINK.Il listato B.1 è un tipico esempio di file MAKE. I numeri di linea che appaiono a sinistra del listato servono solo per aiutare il lettore nella comprensione del listato stesso, ma non fanno parte realmente del file. La linea 1 specifica il nome di due parametri su cui operare la sostituzione nel file di comando: sourcename e listingname. Se il file si chiama MAKE, si può eseguire il file di comando semplicemente digitando la linea

```
execute MAKE File1
```

per compilare il programma chiamato File1.c. Se il compilatore trova degli errori, il file MAKE potrebbe non funzionare. La linea 6 controlla se è stato fornito il nome di un file sorgente. Se non è così, le linee 23-25 visualizzano le istruzioni necessarie. La linea 9 controlla l'esistenza del file fornito in input. Se non lo trova, la linea 28 dice che esso non è stato trovato. Le linee 13 e 15 forniscono due metodi alternativi per specificare il comando LC1, a seconda se è presente o meno il file di Listing. L'opzione -i sulla linea di comando dice al compilatore dove trovare i file Include necessari alla compilazione. I file Include contengono le definizioni per le costanti, le macro, e le strutture dati necessarie per l'uso del software di sistema di Amiga. Il nome del Path dei file Include qui fornito è "C1.O:include", che significa la directory Include sul disco dell'Amiga C.

```
1  .key sourcename,listingname
2  ;sourcename è il nome del programma C da compilare
3  ;listingname è il nome del file per il l'output del listing
4  ;se un file di output del listing deve essere aperto
5  ;
6  IF "<sourcename>" EQ ""
7  SKIP USAGE
8  ENDIF
9  IF NOT EXISTS <sourcename>.c
10 SKIP NOTFOUND
11 ENDIF
12 IF <"listingname"> NOT EQ ""
13 LC1 <listingname> -iC1.0:include/ <sourcename>.c
14 ELSE
15 LC1 -iC1.0:include/ <sourcename>.c
16 ENDIF
```

```
17 ;
18 LC2 <sourcename>
19 ;
20 ALINK FROM <sourcename>.o+Lstartup.obj to <sourcename>LIBRARY
   LC.LIB,AMIGA.LIB,DEBUG.LIB  SKIP DONE
21 ;
22 LAB USAGE
23 ECHO "COMMAND IS: MAKE sourcename listingname"
24 ECHO "sourcename è il sorgente; rappresenta sourcename.c"
25 ECHO "listingname è opzionale"
26 SKIP DONE
27 LAB NOTFOUND
28 ECHO "<sourcename>.c NOT FOUND!!"
29 LAB DONE
```

listato B.1

Se al disco del C è stato cambiato nome mediante il comando DOS RELABEL oppure se si possiede la versione C1.1 invece della C1.0, si dovrà modificare questa linea perché contenga il nome giusto. Per controllare quale nome si deve specificare, si dia da CLI il comando INFO. Esso mostrerà il nome dato al disco in uso. Potrebbe essere, per esempio :

```
C1.0[mounted]
```

La linea 20 mostra una possibile forma del comando ALINK. Questa linea di comando linka un singolo file con il file di StartUp e cerca la altre librerie necessarie per eseguire le funzioni inserite nel programma C. Se, come accade per alcuni programmi del libro, si devono compilare certe parti separatamente e poi linkarle insieme, si dovrà avere una versione diversa del file MAKE che non usa il comando della linea 20. Allora si userà un file separato, magari chiamato Link, simile al file listato nelle linee seguenti :

```
.key linkwhat,object
ALINK FROM <linkwhat>+Lstartup.obj To <object>
LIBRARY LC.LIB,AMIGA.LIB,DEBUG.LIB
```

Poi si lancerà il programma Link scrivendo

```
execute Lin"fa.o+fb.o+fc.o" outname
```

dove i nomi dei file object vengono racchiusi tra virgolette per far si che il comando EXECUTE tratti tutti questi caratteri come un singolo parametro per la sostituzione, e outname è il nome del file eseguibile che verrà creato dal

linkaggio.Si possono trovare ulteriori esempi dell'uso del comando EXECUTE nell'AmigaDOS Users' Manual.

Come creare Makesimple.a

Nel secondo capitolo, feci riferimento a un file chiamato Makesimple.a, che è una derivazione del file examples/MAKE o examples/makesimple del disco del C. Per creare makesimple.a, si operi per gradi nel modo seguente :

1. Si copi il programma makesimple (o make) in un file di nome makesimple.a. Questo si può fare dal CLI scrivendo :

```
copy df1:examples/makesimple to df2:examples/makesimple.a
```

2. Si usi un Text Editor per operare i seguenti cambiamenti. Si cambi la linea:

```
c/lc2 <files>
```

in modo che sia:

```
c/lc2 -v <file>
```

Questo fa si che il controllo dello Stack non venga accluso al programma compilato.

3. Si cambi la linea :

```
   c/alink FROM LIB:LStartup.obj+<file>.o TO <file>
LIBRARY LIB:amiga.lib+LIB:lc.lib
```

in modo che sia:

```
   c/alink FROM LIB:AStartup.obj+<file>.o TO <file> LIBRARY
   LIB:lc.lib+ LIBRARY LIB:amiga.lib
```

(*Nota bene:* non si dia Return finché non è stata scritta tutta la linea di comando). Questo cambia il file di StartUp e l'ordine d'uso delle librerie. Questo significa dover accettare alcune limitazioni nelle routine di libreria fornite in amiga.lib. (Si veda la documentazione riguardante amiga.lib nell'Amiga ROM Kernel Manual per tali limitazioni). Inoltre questo spesso restringe le dimensioni del file eseguibile da 12000 byte a circa 2000 byte.

Quando si usa makesimple.a al posto del file make originale, il programma sarà linkato con le versioni di prints e sprintf presenti nell'amiga.lib, invece che con quelle della libreria della Lattice, che hanno delle limitazioni nella formattazione (la più importante consiste nell'impossibilità di formattare i dati floating-point). Questa è una delle ragioni per le quali le dimensioni del programma finale risultano ridotte. Si noti che tutto questo può avere delle conseguenze serie per alcuni programmi, quindi si tengano ben presenti le limitazioni prima di fare quanto descritto. Si può aggiungere che lo scopo principale del file makesimple.a è quello di rendere il sorgente C, soprattutto per i programmi del secondo capitolo, il più compatibile possibile con versioni diverse di compilatori (Lattice C, Aztec C, e così via). Pertanto, invece di usare delle funzioni specifiche di un Compilatore, come fopen, fread, e fclose, e le funzioni specifiche di gestioni di file, come FILE, nel secondo capitolo sono state usate le funzioni specifiche di Amiga, come Open, Read, e Close, e le funzioni di gestione file tipiche di Amiga non legate al compilatore utilizzato. Quei lettori che abbiano già esperienza nella programmazione in C, potranno usare delle funzioni di I/O specifiche del loro compilatore. Si noti, però, che la gestione dei file dell'AmigaDOS è diversa, e quindi non può essere mischiata, con quella delle funzioni tipo Unix che spesso sono fornite nelle librerie dei compilatori.

GRUPPO
EDITORIALE
JACKSON
Cod. CC795

Finito di stampare nel mese di Luglio 1989
da Grafica '85 - Rodano Millepini (MI)

Altri libri di argomento collegato al presente volume:

AUTORE	TITOLO	SUPPORTO	CODICE
David Lawrence Mark England	**AMIGA HANDBOOK**		CC320
Rita Bonelli Massimiliano Lunelli	**AMIGA 500** **GUIDA PER L'UTENTE**		CC627
Andrea Bigiarini Pierluigi Cecioni Marco Ottolini	**IL MANUALE DI AMIGA**		CZ532
Axel Plenge	**AMIGA-GRAFICA 3D**	DISCO 3½"	CZ756
Edgar Huckert Frank Kremser	**AMIGA-LINGUAGGIO C**	DISCO 3½"	CL758

Di prossima pubblicazione:

AUTORE	TITOLO	SUPPORTO	CODICE
Horst-Rainer Henning	**AMIGA-BASIC**	DISCO 3½"	CL768

AMIGA
tecniche di programmazione

Robert A. Peck

Le spettacolari capacità grafiche e sonore di Amiga non devono far credere che non sia un computer adatto ad applicazioni serie, anzi sotto alcuni punti di vista le potenzialità di Amiga sono decisamente superiori a quelle offerte da alcuni personal della fascia definita professionale, in quanto il computer della Commodore utilizza un processore molto più potente in modo decisamente innovativo.
Questo libro è una completa ed esauriente guida per il programmatore, contiene infatti tutte le informazioni necessarie per affrontare seriamente la programmazione in C su Amiga.
Cominciando dai problemi più semplici fino ad arrivare alle funzioni più sofisticate, il testo passo dopo passo sviluppa ed approfondisce i concetti fondamentali della programmazione avanzata in C, privilegiando soprattutto gli esempi pratici.
Dopo aver analizzato in modo approfondito l'AmigaDOS, il libro passa ad esaminare le varie funzioni dell'EXEC e le straordinarie capacità grafiche di Amiga, quindi illustra come agganciare le routine del software di sistema Intuition e come utilizzare in modo corretto i Device.
Il libro contiene infine preziose informazioni per sfruttare appieno le possibilità offerte dagli Sprite hardware, dalla sofisticata circuitazione audio e dalle capacità multitasking di Amiga allo scopo di ottenere effetti di animazione e sonori estremamente realistici; in appendice sono contenuti cenni sul Text Editor fornito con il computer e sull'utilizzo corretto del compilatore C.

Sommario

GRUPPO EDITORIALE JACKSON

ISBN 88-7056-920-9

9 788870 569209

L. 62.000 Cod. CC795

792

ROBERT A. PECK

AMIGA tecniche di programmazione