coordinamento di Andrea de Prisco

Programmare in C su Amiga (16)

di Dario de Judicibus

Ed eccoci finalmente alla tanto sospirata IDCMP, che ci permetterà di comunicare con Intuition e gestire appieno l'interfaccia utente per mezzo di menu, quadri ed altri oggetti che questo componente del sistema operativo dell'Amiga ci mette a disposizione

Prima di passare agli argomenti di questa puntata, di cui il piatto forte è l'introduzione all'IDCMP, vediamo una possibile soluzione all'esercizio proposto nella scorsa puntata.

Si trattava di scrivere un programma grafico di tipo *interattivo*, in cui cioè l'utente avesse un certo controllo sul procedere del programma stesso, lanciando comandi o richieste di informazioni. Questo scheletro può essere estremamente utile per chi voglia sviluppare programmi grafici il cui scopo è di effettuare *macrooperazioni* grafiche, senza perdere tempo a gestire menu ed eventi da tastiera o dal mouse. Se infatti Intuition è certamente molto semplice da usare da parte degli utenti Amiga, può risultare abbastanza macchinoso per il programmatore che è più interessato al risultato pratico del programma che l'aspetto esterno dello stesso. Amiga infatti può essere un grosso strumento di produttività nel campo grafico. Se poi dotato di una scheda acceleratrice (68020, ad esempio), può essere molto interessante per lo studio di insiemi matematici, frattali, strutture soggette a sforzi e via dicendo. Se state sviluppando una vostra idea, e quindi non esiste già un programma ad hoc per voi, potreste non voler perder tempo a metter su un'interfaccia carina se non addirittura spettacolare, per concentrarvi piuttosto sul problema che state analizzando. Questa tecnica, riportata in figura 1, ha il vantaggio di essere semplice da scrivere, da verificare, e soprattutto da estendere.

Facendo riferimento alla figura in questione, e saltando le tecniche già descritte in precedenza (come quella della maschera di apertura e chiusura **mask**), vediamo che il tutto si realizza aprendo due finestre, una grafica, su cui verranno effettuate le operazioni richieste, l'altra di tipo CLI, per l'immissione dei comandi e la visualizzazione di eventuali informazioni desiderate. Per semplicità la finestra grafica è di tipo GZZ, ma si può utilizzare anche una finestra normale, a condizione di gestire opportunamente i bordi e le operazioni di *refresh*. L'apertura di questa finestra è effettuata attraverso la funzione di Intuition **OpenWindow()**, come di consueto. La finestra di tipo testo, invece, è aperta tramite la funzione interna del Lattice C **fopen()**, come abbiamo visto nelle prime puntate di questa serie. Dato che tale finestra viene aperta come un file in scrittura e lettura, è **necessario** operare un *riavvolgimento* virtuale dello stesso ogni qual volta si alternano operazioni in lettura ed in scrittura. A tale scopo viene utilizzata la funzione **rewind()**. Ovviamente l'aggancio al file viene realizzato tramite il puntatore ad una struttura **FILE**, piuttosto che ad un File Handle dell'AmigaDOS.

Il cuore del programma è formato da un ciclo *[loop]* che *legge* dalla finestra CLI il comando immesso dall'utente, tramite la funzione interna del C **fgets()**, lo analizza con la **sscanf()**, e lo confronta con una serie di comandi validi, suddivisi a seconda del numero di parametri accettati in ingresso. Questo pezzo di codice può essere scritto in decine di modi diversi, a seconda della complessità della sintassi definita per i nostri comandi e delle possibilità offerte all'utente. Un programma sufficientemente complesso può risultare molto simile al CLI a cui siamo abituati. Notare che, avendo aperto la finestra CLI con CON:, possiamo sfruttare tutte le possibilità offerte da **ConMan**. Potevamo aprire anche una finestra di tipo NEWCON: ovviamente, e sfruttare così lo **Shell** del WorkBench 1.3.

Non entrerò nei particolari scelti per questo esempio, dato che il codice in figura è abbastanza semplice. Giusto due parole sul comando **EXIT**. Il modo di uscire dal programma poteva essere scelto usando svariate tecniche. Quello qui presentato prevede la possibilità che il programma, ricevuto il comando **EXIT**, visualizzi sulla finestra CLI alcune informazioni statistiche sulle operazioni grafiche effettuate (ad esempio generazioni in un sistema cellulare), e quindi si dà la possibilità all'utente di leggere con calma tali informazioni prima di chiudere realmente tutto con il gadget della finestra grafica. Non ho ritenuto necessario riportare tutto il codice relativo, che fa parte di un mio studio sugli automi cellulari, tuttavia ho pensato che fosse interessante mostrare tale possibilità. Ovviamente si poteva anche evitare il gadget di chiusura, aggiungere un normale messaggio del tipo «*Sei sicuro? (S/N)*» dopo il comando **EXIT** e chiudere il tutto subito in caso di risposta affermativa.

Questo è tutto. Naturalmente l'esempio riportato si limita a operazioni grafiche elementari, tuttavia, i comandi possono riguardare qualsiasi tipo di macroistruzione. Ad esempio, nella generazione di frattali, si può chiedere lo zoom di una parte del piano o la modifica della tavolozza dei colori. In uno studio di forme tridimensionali, si può cambiare l'angolo di vista, ruotare o spostare un oggetto, deformare la superficie di rotazione. Infine, nel caso di analisi degli sforzi su strutture di vario tipo (ad esempio, capriate), si può modificare il vettore degli sforzi su un elemento della struttura, od aggiungere nuovi elementi.

```
/* ********************************************************************** */
/* Esercizio 15  (2 Settembre 1989) (c) Dario de Judicibus                */
/*                                                                        */
/*   Esempio di programma interattivo che non fa uso della IDCMP, ma      */
/*   sfrutta una finestra CLI come meccanismo di immissione dati.         */
/*                                                                        */
/* ********************************************************************** */

/*
**  INCLUDE di sistema, prototipi delle funzioni interne, alcune costanti
**  e variabili globali.
*/

 #include "exec/types.h"
 #include "intuition/intuition.h"
 #include "graphics/gfxmacros.h"
 #include "proto/exec.h"
 #include "proto/intuition.h"
 #include "proto/graphics.h"
 #include "stdio.h"
 #include "stdlib.h"
 #include "string.h"

 void help(void);
 void StartAll(void);
 void CloseAll(void);

 #define IREV 0
 #define GREV 0
 #define INAME "intuition.library"
 #define GNAME "graphics.library"
 #define WFLAGS  WINDOWCLOSE|WINDOWDEPTH|WINDOWDRAG|WINDOWSIZING

 struct IntuitionBase *IntuitionBase;
 struct GfxBase *GfxBase;
 FILE *dosfp;

 #define WWW 300
 #define HHH 150
 struct NewWindow nw =
 {
   20, 20, WWW, HHH,
   0, 1,
   CLOSEWINDOW,
   WFLAGS|GIMMEZEROZERO|SMART_REFRESH|ACTIVATE,
   NULL, NULL, "Graphics",
   NULL, NULL, 0, 0, 0, 0,
   WBENCHSCREEN
 };
 struct Window *w;
 struct RastPort *rp;
 #define IMASK 0x01
 #define GMASK 0x02
 #define WMASK 0x04
 #define DMASK 0x08
 unsigned char mask = 0x00;

 char buf[80];
 int done = FALSE;

/*
**  Qui inizia il programma vero e proprio. Non ci sono parametri d'ingresso.
*/

 void main()
 {
   int i;
   int x = 0, y = 0;
   int oldx = 0, oldy = 0;
   char cmd[5], *p;

  /*
  **  Apriamo le varie librerie e le due finestre: quella grafica e quella
  **  di tipo testo (CLI) per l'immissione dei comandi.
  */
   StartAll();

  /*
  **  Selezioniamo il modo grafico
  */
   SetDrMd(rp,JAM1);

  /*
  **  Questo è il ciclo principale che "legge" dalla finestra CLI i comandi
  **  immessi e, se validi, chiama le operazioni grafiche associate.
  */
   do
   {
    /*
    **  Questo "rewind" è fondamentale quando si apre un file in lettura e
    **  scrittura, ogniqualvolta si alterna una operazione in ingresso con
    **  una in uscita.
    */
     rewind(dosfp);
    /*
    **  Il "prompt" della nostra CLI
    */
     fprintf(dosfp,"\x1b[33mCMD>\x1b[0m ");
     rewind(dosfp);
```

Il tutto vedendo in tempo reale il risultato di una certa operazione. Per operazioni complesse, suggerisco di mantenere il ricordo dello stato precedente dell'ambiente su cui si sta operando, in modo da poter implementare il comando **UNDO** per restaurare la situazione precedente ad una operazione errata o non voluta.

Introduzione

E passiamo ora agli argomenti del mese.

Innanzi tutto termineremo il discorso su *LMK* descrivendo i parametri e le opzioni di tale comando, nella versione 1.05 (quella fornita col Lattice C 5.02, tanto per intenderci). Mostreremo anche un semplice trucco per mantenere un semplice processo di compilazione senza utilizzare *LMK*.

Passeremo quindi ad introdurre **IDCMP**, in modo da poter parlare al più presto di menu, gadget e requester.

Infine una novità: *la scheda tecnica*. Si tratta di una scheda contenente delle informazioni molto importanti per chi lavora con l'Amiga. Volta per volta potrà trattarsi una lista contenente informazioni sull'Amiga, oppure trucchi di vario genere per sfruttare meglio la macchina, od anche notizie di prima mano di *fixes* o *patches* da applicare al vostro software, per risolvere alcuni problemi del sistema operativo o del compilatore. Questo mese parleremo di un problema che affligge i possessori di Amiga PAL.

LMK

Terminiamo la nostra trattazione di *LMK* riportando le opzioni che possono essere usate con questo programma:

-a Ricostruisce *tutti* i bersagli, indipendentemente dalla data di creazione o dell'ultima modifica effettuata.

-b file Specifica *file* come file di default, al posto di **lmk.def**.

-c Crea un file di tipo *batch* a partire dalla sessione corrente di *LMK*. Tale file verrà quindi eseguito con il comando

execute lmkfile.bat

e rimarrà nel direttorio corrente quale traccia storica delle azioni svolte e dei risultati conseguiti.

-d Visualizza tutta una serie di informazioni di debug sull'esecuzione della sessione corrente di *LMK*.

-e Cancella tutti i bersagli obsoleti, prima di rigenerarli.

-f file Utilizza *file* come **lmkfile** d'ingresso.

-h Visualizza informazioni guida *[help]*.

-i Ignora tutte le segnalazioni di errore a fronte delle singole azioni effettuate.

-k Ignora tutte le segnalazioni di errore a fronte delle singole azioni effettuate come restituite da AmigaDOS, più tutte quelle ritornate da *LMK* riguardanti l'impossibilità di generare un certo bersaglio.

-n Visualizza le azioni senza realmente effettuarle.

```
/*
**  Vediamo che ci chiede di fare l'utente del programma...
*/
 p = fgets(buf,sizeof(buf),dosfp);
/*
**  INVIO a vuoto? Forse non sa cosa fare... Diciamoglielo.
*/
 if (p == NULL)
 {
  help();
  continue;
 }
/*
**  OK. E' arrivato qualcosa. Vediamo se è un comando valido.
**  I comandi sono divisi in categorie a seconda del numero di parametri
**  che accettano in ingresso.
*/
 i = sscanf(buf,"%s %ld %ld\n",cmd,&x,&y);
 switch (i)
 {
  /*
  **  Nessun parametro (solo il comando). Può essere solo EXIT.
  */
   case  1: if (strcmpi(cmd,"EXIT") == 0)
            {
              rewind(dosfp);
              fprintf(dosfp,"\x1b[32mClick on close gadget to end\x1b[0m");
              done = TRUE;
            }
     /* --
     ** -- Questo è un esempio di un comando per ottenere alcune
     ** -- informazioni su alcuni parametri grafici (modo, penne,
     ** -- posizione corrente). Ovviamente, se implementato, sarebbe
     ** -- il caso di fornire anche dei comandi per variare tali dati:
     ** -- ad esempio SET APEN, SET BPEN, SET MODE...
     ** --
     ** -- else if (strcmpi(cmd,"INFO") == 0)
     ** -- {
     ** --   fprintf(dosfp,"\x1b[32mPOSITION\x1b[0m  %ld %ld\n",
     ** --                 rp->cp_x ,rp->cp_y );
     ** --   fprintf(dosfp,"\x1b[32mPENS COL\x1b[0m  %ld %ld %ld\n",
     ** --                 rp->FgPen,rp->BgPen,rp->AOlPen);
     ** -- }
     */
           else help();
           break;
```

```
    /*
    **  Due parametri (nessun comando da uno solo). Per ora sono stati
    **  implementati MOVE, DRAW e RECT.
    */
     case  3: if (strcmpi(cmd,"MOVE") == 0)
              {
                Move(rp,x,y);
              }
              else if (strcmpi(cmd,"DRAW") == 0)
              {
                Draw(rp,x,y);
              }
              else if (strcmpi(cmd,"RECT") == 0)
              {
                Draw(rp,oldx,y);
                Draw(rp,x,y);
                Draw(rp,x,oldy);
                Draw(rp,oldx,oldy);
              }
              else help();
              break;
     default: help();
              break;
    }
    /*
    **  OK. Salviamo la posizione corrente, dato che serve a semplificare
    **  il comando RECT, che altrimenti avrebbe QUATTRO parametri.
    */
     oldx = x, oldy = y;
  } while (!done);

/*
**  Andiamo sul semplice, per quello che riguarda la chiusura: EXIT chiede
**  di usare il gadget di chiusura per chiudere tutto.
*/
 Wait(1<<w->UserPort->mp_SigBit);
 CloseAll();
}



/* ------------------------------------------------------------------- */
/*  Procedura di chiusura. L'ordine è IMPORTANTE!                      */
/* ------------------------------------------------------------------- */
 void CloseAll()  /* ordine inverso!!! */
 {
   if (mask & DMASK) (void)fclose(dosfp);
   if (mask & WMASK) CloseWindow(w);
   if (mask & GMASK) CloseLibrary(GfxBase);
   if (mask & IMASK) CloseLibrary(IntuitionBase);
   Exit(0);
 }

/* ------------------------------------------------------------------- */
/*  Procedura di apertura: librerie e finestre.                        */
/* ------------------------------------------------------------------- */
 void StartAll()
 {
  /*
  **  Open libraries (Intuition & Graphics) and windows
  */
   IntuitionBase = (struct IntuitionBase *)OpenLibrary(INAME,IREV);
   if (IntuitionBase == NULL) CloseAll();
   mask |= IMASK;
   GfxBase = (struct GfxBase *)OpenLibrary(GNAME,GREV);
   if (GfxBase == NULL) CloseAll();
   mask |= GMASK;
   w = (struct Window *)OpenWindow(&nw);
   if (w == NULL) CloseAll();
   mask |= WMASK;
   rp = w->RPort;
   dosfp = fopen("CON:320/20/300/150/Commands","w+");
   if (dosfp == NULL) CloseAll();
   mask |= DMASK;
 }

/* ------------------------------------------------------------------- */
/*  Procedura di visualizzazione aiuto.                                */
/* ------------------------------------------------------------------- */
 void help()
 {
   rewind(dosfp);
   fputs("\x1b[32mCommand    Description\x1b[0m\n",dosfp);
   fputs("MOVE x y   Move to point (x,y)\n",dosfp);
   fputs("DRAW x y   Draw to point (x,y)\n",dosfp);
   fputs("RECT x y   Draw a box to (x,y)\n",dosfp);
   fputs("EXIT       Leave the application\n",dosfp);
 }
```

Figura 1 Soluzione dell'esercizio E15.

-p Visualizza la descrizione dei bersagli e le macro espanse.

-q Verifica se un bersaglio è obsoleto o meno. Nel primo caso ritorna uno **«0»**, altrimenti ritorna un **«1»**. In ogni caso non esegue alcuna azione di generazione.

-s Non visualizza le azioni prima di effettuarle.

-t Modifica la data dei bersagli in modo da impostarla al valore corrente, senza peraltro effettuare alcuna azione.

-u Come **-a**.

-x Per compatibilità con UNIX. Se usata, *LMK* verifica se il file di ingresso è perfettamente compatibile con il comando UNIX **make**. Eventuali incompatibilità sono evidenziate.

È inoltre possibile modificare il valore di una macro direttamente dal comando *LMK* come riportato qui di seguito:

1> lmk -f mio.lmk miamacro=NUOVO

```
echo ; /*  Dichiara "echo" come una variabile intera esterna
 lc:lc -cwusf -v -y -j121 -O qsp.c
 lc:blink lib:c.o qsp.o TO qsp LIB lib:amiga.lib,lib:lc.lib SD SC ND VERBOSE
 quit
*/

/*
 *  Qui inizia il programma C
 *
 *  Module:    qsp  (Query SPace)
 *
 */

 #include "libraries/dos.h"
 #include "libraries/dosextens.h"
 .
 .
 .
 VOID _main(cmd)
 char *cmd;
 {
   .
   .
   .
 }
```

```
struct MsgPort
{
  struct Node   mp_Node;    /* Nodo di aggancio                          */
  UBYTE         mp_Flags;   /* Azioni da prendere in caso di messaggio   */
  UBYTE         mp_SigBit;  /* Numero del segnale in caso di PA_SIGNAL   */
  struct Task  *mp_SigTask; /* Task da segnalare o struttura interruzione */
  struct List   mp_MsgList; /* Lista dei messaggi arrivati               */
}

struct Message
{
  struct Node     mn_Node;      /* Nodo di aggancio                      */
  struct MsgPort *mn_ReplyPort; /* Porta per la risposta                 */
  UWORD           mn_Length;    /* Lunghezza del messaggio in byte       */
}

struct IntuiMessage
{
  struct Message       ExecMessage;     /* Messaggio di tipo "classico"  */
  ULONG                Class;           /* Contiene i flag IDCMP         */
  USHORT               Code;            /* -- Vedi testo dell'articolo -- */
  USHORT               Qualifier;       /* -- Vedi testo dell'articolo -- */
  APTR                 IAddress;        /* Puntatore ad un oggetto       */
  SHORT                MouseX, MouseY;  /* Posizione del mouse           */
  ULONG                Seconds, Micros; /* Orologio dell'Amiga           */
  struct Window       *IDCMPWindow;     /* Finestra a cui msg appartiene */
  struct IntuiMessage *SpecialLink;     /* Riservato per il sistema      */
}
```

Figura 3 - Strutture MsgPort, Message ed IntuiMessage.

◀ *Figura 2 - Integrazione di sorgente C e script AmigaDOS.*

Non sono ammessi spazi ai lati del segno di uguale. Il nuovo valore di **miamacro** sostituirà quello specificato in **mio.lmk.**

Ci sarebbe altro da dire su *LMK*, ma questo toglierebbe spazio ad altri argomenti. Concludiamo quindi ricordando che anche se all'inizio questo programma può risultare un pochino ostico, i vantaggi che comporta man mano che si acquisisce una certa dimestichezza, compensano di gran lungo il tempo necessario a prenderci la mano.

Un trucco

Benché l'utilità di *LMK* sia indubbia quando il processo di generazione è particolarmente complesso, questo strumento può risultare eccessivo per processi semplici, come la compilazione di un programma a partire da un singolo sorgente. Fermo restando che è importante mantenere traccia dei parametri di compilazione e *link* utilizzati, si pone il problema di come associare al programma tali informazioni senza creare un altro file con le informazioni suddette. Un trucco è stato sviluppato da John Toebes, ed utilizzato in molti dei programmini che ho scritto, di cui alcuni (**eco, qsp**) possono essere trovati su MC-Link. Si tratta di integrare il sorgente del programma con il file **script** che genera lo stesso, come riportato in figura 2. In pratica la prima istruzione (**echo ;/* ...**) viene interpretata dal C come la dichiarativa di una variabile esterna di tipo **int**, ed eseguita dal DOS visualizzando una stringa nulla a terminale. Tale istruzione serve a nascondere al DOS la sequenza di *inizio commento* del C grazie alla presenza del punto e virgola, ed il cui scopo è commentare tutte le istruzioni seguenti che vengono viceversa interpretate dal DOS per generare il programma. Questi ritorna al CLI grazie al comando **quit** prima di arrivare alla sequenza di *fine commento* del C. Quindi, il compilatore è in grado di compilare correttamente il sorgente (ad esempio **qsp.c**), dato che tutte le istruzioni di compilazione sono commentate ad eccezione di **echo**. Viceversa il DOS non cerca di eseguire le istruzioni C dato che termina prima grazie a **quit**.

Per compilare il programma (es. **qsp.c**), basterà allora lanciare il comando

1> execute qsp.c

IDCMP

Ed eccoci finalmente a parlare della *Intuition Direct Communications Message Port* o, in breve, *IDCMP*. Una curiosità: questo è l'unico acronimo usato dagli sviluppatori del sistema operativo, i quali, in tutti gli altri casi, hanno cercato di usare il più possibile una terminologia «parlante», al contrario di quello che succede in altri sistemi operativi. L'IDCMP è uno dei due metodi che un programma ha di comunicare con Intuition, e di fatto il solo che permette la comunicazione sia da che verso Intuition. L'altro è la *console device*.

Per chi non si ricordasse quanto detto sulle porte e sui messaggi nella 5ª puntata (MC n.78, ottobre 1988), cercherò di riepilogare brevemente il discorso.

Una *porta messaggi [message port]* è una struttura dati mantenuta in memoria del sistema operativo, alla quale un task può mandare un messaggio. Questi altri non è che un blocco nella memoria del task *che spedisce il messaggio* e che contiene informazioni che possono venir analizzate da altri task. In genere la porta è posseduta da un determinato task. Quando un messaggio arriva ad una porta, il task che lo possiede può o meno venire avvisato dell'evento. Nel primo caso, il sistema operativo può segnalare la ricezione del messaggio al task destinatario, od addirittura far partire una interruzione software dello stesso. Nel secondo caso, il messaggio è semplicemente aggiunto alla lista dei messaggi associati a quella porta, e fornito dietro richiesta al task destinatario. È buona norma rispondere **sempre** ad un messaggio. Meglio ancora, salvare le informazioni che ci interessano nella memoria locale, e rispondere prima ancora di avere analizzato i dati.

L'IDCMP è in realtà formata da due porte messaggi distinte, una per il programma, ed una per Intuition. Per aprire automaticamente queste porte, basta impostare ad un valore non nullo il campo **NewWindow.IDCMPFlags**. È comunque sempre possibile aprire successivamente (e chiudere) IDCMP per mezzo della funzione **ModifyIDCMP()**. Se tuttavia si chiude IDCMP prima di aver risposto a tutti i messaggi arrivati, Intuition se li cancella deallocando la memoria occupata, senza aspettare il **Reply()** del programma destinatario. Se questi risponde dopo la chiusura di IDCMP, si può andare in GURU. Vedremo che è anche possibile chiedere ad Intuition di usare come porta IDCMP

utente una porta precedentemente aperta dal programma o dallo stesso Intuition.

La porta utente fornita automaticamente da Intuition si chiama appunto **UserPort**, e il suo puntatore è memorizzato nella struttura **Window** generata all'apertura della finestra. Esistono due istruzioni equivalenti per mettersi in attesa sulla porta utente. Entrambe sono riportare in figura 4. Si sconsiglia vivamente di utilizzare al loro posto un ciclo di **GetMsg()**, dato che questo andrebbe a togliere tempo di CPU ad altri task che stanno girando. L'altra porta si chiama invece **WindowPort**, e serve ad Intuition a ricevere le risposte ai messaggi inviati. Che io sappia, il programma non ha mai la necessità di inviare *direttamente* un messaggio a questa porta. L'unico utilizzo che conosco è appunto quello indiretto, via **ReplyMsg()**.

Il programma, tramite i flag IDCMP, specifica ad Intuition quali sono gli eventi a cui è interessato, e di cui vuole essere notificato per mezzo di un messaggio. Questo può riferirsi ad i seguenti eventi (vedi note 1 ed 2):

Finestre
- Un utente ha attivato, disattivato, ridimensionato o chiuso una finestra.
- Una finestra ha bisogno di essere restaurata.

Dischi

Un disco è stato inserito o rimosso da una unità dischi.

Menu

Un certo elemento (o sottoelemento) di un menu è stato selezionato.

Quadri

Un quadro sta per apparire, è apparso od è stato cancellato da una finestra (vedi nota 3).

Gadget
- Un gadget è stato selezionato o rilasciato.
- Il gadget di chiusura di una finestra è stato selezionato.

Figura 4
Istruzioni di attesa sulla porta utente.

```
/*                              *** 1 ***
**  "w" è il puntatore alla finestra. Il messaggio può essere recuperato e
**  rimosso con la GetMsg().
*/
 Wait(1<<w->UserPort->mp_SigBit);

/*                              *** 2 ***
**  "imsg" è il puntatore al messaggio. Questi deve comunque essere poi
**  rimosso con la GetMsg().
*/
 imsg = WaitPort(w->UserPort);
```

Mouse
- Uno dei due pulsanti del mouse è stato premuto o rilasciato.
- La posizione del mouse relativa alla finestra viene riportata in corrispondenza della selezione di un gadget, o sempre e comunque, sia come valori assoluti che come delta relativi alla posizione precedente.

Tastiera
- Un tasto della tastiera è stato premuto ed il codice originale *[raw key]* viene riportato nel messaggio.
- Un tasto della tastiera è stato premuto ed il codice filtrato *[mapped key]* viene riportato nel messaggio.

Contatore

Intuition può mandare un messaggio dieci volte al secondo, circa, dietro richiesta del programma. Utile per evitare di rimanere in attesa infinita su un evento che, per un qualche motivo, non può avere luogo.

Configurazione del Sistema

L'utente od un altro programma ha modificato la configurazione del sistema.

I messaggi scambiati sulle porte della IDCMP sono una estensione del classico messaggio EXEC. La struttura **IntuiMessage** è riportata in figura 3. Essa è formata dai seguenti campi:

ExecMessage.

La struttura **Message** che serve ad EXEC a gestire il messaggio stesso.

Class

Questa variabile di tipo ULONG contiene praticamente i bit corrispondenti ai flag IDCMP.

Code.

Essa serve a specificare ulteriormente la classe del messaggio, e contiene valori differenti a seconda del tipo di messaggio. Ad esempio il codice relativo al menu selezionato, oppure il contenuto del campo **InputEvent** impostato dal device di ingresso (es. carattere in caso di classe VANILLAKEY, o *raw key* in caso di classe RAWKEY).

Qualifier

Copia del campo **ie_Qualifier** trasmesso ad Intuition dal device di ingresso, utilizzato in genere per i messaggi di classe RAWKEY.

IAddress

Puntatore ad un eventuale oggetto Intuition (gadget o finestra, ad esempio) interessato dal messaggio stesso.

MouseX, MouseY

Posizione del mouse relativa all'angolo superiore sinistro della finestra.

Seconds, Micros

Copia del valore corrente dell'orologio di sistema *[system clock]*. La gamma di valori per i secondi (32 bit) copre 139 anni, quella per i microsecondi va da 0 a 999.999. Dato che non possono esistere due coppie di campi uguali (a meno che il vostro programma non giri per più di 140 anni!), questi campi formano anche un identificativo univoco del messaggio stesso.

IDCMPWindow

Puntatore alla finestra a cui il messaggio si riferisce. Questo campo è particolarmente utile quando alla stessa porta arrivino più messaggi relativi a finestre differenti. Vedremo in seguito come questo sia possibile.

SpecialLink

Campo riservato.

Come si vede si tratta di una struttura particolarmente flessibile, adatta a supplire un gran numero di informazioni differenziate e corrispondenti ad i vari possibili eventi.

Vediamo ora come si comporta Intuition al verificarsi di un certo evento. Innanzi tutto verifica se uno o più programmi sono interessati all'evento in questione. Quindi controlla se aveva già

Note

1. Useremo in seguito il termine *quadro* al posto dell'inglese *requester* mentre manterremo il termine inglese *gadget* dato che i vari tipi di gadget si prestano ad essere riportati con nomi differenti in italiano, ed il termine generico *aggeggio* o *congegno* richiama alla mente, in italiano, più qualcosa di fisico, meccanico, piuttosto che logico. Si tratta ovviamente di una scelta soggettiva e quindi opinabile.

2. La finestra a cui si fa riferimento è ovviamente quella a cui è associato il messaggio ricevuto sulla porta IDCMP.

3. Dalla versione 1.2 del WorkBench, gli eventi relativi alla comparsa od alla cancellazione di un quadro vengono emessi per *ogni* quadro, e non solo per il primo e l'ultimo rispettivamente, come nelle versioni precedenti.

allocato in precedenza un messaggio del tipo necessario a riportare tale evento. In caso affermativo il messaggio in questione viene riusato, altrimenti ne viene allocato uno nuovo. Dato che un messaggio può essere riusato solo se il programma destinatario lo ha rilasciato, ha cioè risposto allo stesso, è bene rispondere al più presto ai messaggi che si riceve da Intuition, per evitare che questi consumi tutta la memoria libera disponibile. Questo è tanto più vero quanto più è frequente il messaggio richiesto, come nel caso del tracciamento della posizione del cursore del mouse. In tal caso è bene duplicare subito le coordinate richieste e rilasciare il messaggio prima di qualunque altra elaborazione debba essere effettuata.

Conclusione

Bene. Anche per questa volta è tutto. Nella prossima puntata entreremo nel vivo di Intuition cominciando a costruire uno scheletro di programma per la gestione dei messaggi via IDCMP e che sarà la base di tutti i discorsi seguenti su menu, quadri e gadget. Come promesso vedremo come si può associare a più finestre la stessa porta utente. Parleremo inoltre di un altro programma di utilità legato allo sviluppo di codice: **grep**. Vi suggerisco nel frattempo di sviluppare l'esercizio proposto nella scorsa puntata, cercando di renderlo più modulare e generalizzato, in modo da metter su uno scheletro in cui integrare volta per volta differenti logiche, creando così un valido strumento per l'analisi di oggetti matematici rappresentabili graficamente. Per chi fosse interessato al campo in questione, suggerisco vivamente di andarsi a leggere la serie *(Ri)Creazioni al calcolatore* di A. K. Dewdney pubblicata sulla rivista *Le Scienze* (o nella versione originale su *Scientific American*). E come al solito... buon lavoro!

MC

La scheda tecnica

La tecnica descritta qui di seguito è stata sviluppata da Martin Brenner (Germania). Il problema riguarda il fatto che una volta su trenta capita che lo schermo del WorkBench non si apra sugli Amiga di tipo PAL con l'altezza corretta, e cioè di 256 pixel in media risoluzione, bensì con quella tipica degli schermi NTSC, cioè 200 pixel. Questo problema è abbastanza critico per quegli utenti che hanno nella loro startup-sequence dei comandi di tipo **PopCLI** per aprire successivamente una finestra **CLI** più alta di 200 pixel, e che di conseguenza chiudono la finestra iniziale (640x200) senza aver caricato il WorkBench. Per giunta, un qualunque programma che cerca di aprire una finestra nel formato PAL manda molto facilmente l'Amiga in GURU!

L'errore si trova nella **graphics.library**. Questa contiene una procedura che determina leggendo più volte la posizione verticale del fascio di elettroni, se le 56 linee in più tipiche degli schermi PAL esistono o meno. Qui di seguito è riportata tale routine. Gli indirizzi sono validi **solo** per il Kickstart 1.2/33.180.

```
*
* Procedura per definire se il sistema è PAL o NTSC
*

fcb058:4e9b jsr     $fc5e62          ; leggi la posizione verticale del fascio
fcb05e:2200 move.l  D0,D1
fcb060:0c80 cmpi.l  #$0000010e,D0    ; posizione del fascio maggiore di $10e?
fcb066:6f04 ble.s   $fcb06c          ; no, -> salta
fcb068:7004 moveq   #$04,D0          ; si, imposta il bit PAL
fcb06a:6028 bra.s   $fcb094          ; ed esci

fcb06c:4eb9 jsr     $fc5e62          ; leggi la posizione verticale del fascio
fcb072:2200 move.l  D0,D1
fcb074:7264 moveq   #$64,D1
fcb076:b280 cmp.l   D0,D1            ; posizione del fascio maggiore di $64?
fcb078:6ef2 bgt.s   $fcb06c          ; no, riprova ->

fcb07a:4e9b jsr     $fc5e62          ; leggi la posizione verticale del fascio
fcb080:2200 move.l  D0,D1
fcb082:0c81 cmpi.l  #$0000010e,D1    ; posizione del fascio maggiore di $10e?
fcb088:6f02 ble.s   $fcb08c          ; no, -> salta
fcb08a:60dc bra.s   $fcb068          ; si, imposta il bit PAL (->) ed esci

fcb08c:7032 moveq   #$32,D0
fcb08e:b081 cmp.l   D1,D0            ; posizione del fascio maggiore di $32?
fcb090:61e8 blt.s   $fcb07a          ; si, verifica ancora per $10e ->
fcb092:7001 moveq   #$01,D0          ; no, imposta il bit NTSC
fcb094:4e75 rts

*
* Carica la posizione verticale del fascio in D0 (l'errore è QUI!)
*

fc5e62:2039 move.l  $dff004,D0       ; leggi VPOSR+VHPOSR (longword)
fc5e68:e080 asr.l   #8,D0
fc5e6a:0280 andi.l  #$000001ff,D0    ; posizione del fascio (9-bit) in D0
fc5e70:4e75 rts
```

L'errore è situato nella parte che legge la *longword* da VPOSR+VHPOSR (ricordo che si tratta di due registri Hardware dell'Amiga. n.d.a.). Il valore relativo alla posizione verticale del fascio **dovrebbe** essere un valore compreso fra 0 ed $1ff. Il problema può insorgere quando il fascio viene a trovarsi all'estrema destra della linea $ff allorché viene effettuata la lettura. In tale situazione è infatti possibile che il nono bit relativo alla posizione verticale del fascio, contenuto in **VPROSR** venga letto come 0, dato che tale posizione è descritta dal valore binario **0 1111 1111**. A questo punto, se il contatore scatta a $100 prima della letura di VHPOSR, il byte più alto di questo registro risulta essere $00, per cui il valore risultante viene ad essere (in binario) **0 0000 0000**, piuttosto che $100, il che porta all'esecuzione delle istruzioni $fcb08e, $fcb090 e $fcb092, che impostano il bit NTSC, piuttosto che quello PAL.
Da notare che il problema è tuttora presente nel Kickstart 1.3/34.5!
Martin propone la seguente routine al posto di quella Commodore:

```
*
* Da applicare all'indirizzo $fcb058 del Kickstart 1.2/33.180
*

Start:
            MOVEQ  #$04,D1       ; PAL è il default
WaitLoop1:
            JSR    $fc5e62       ; leggi la posizione verticale del fascio
            CMP.B  #$20,D0       ; posizione del fascio == $20 o $120?
            BNE.S  WaitLoop1     ; no, riprova ->
            AND.W  #$100,D0      ; posizione del fascio == $120?
            BNE    Ready         ; si, è PAL: esci ->
WaitLoop2:
            JSR    $fc5e62       ; leggi la posizione verticale del fascio
            CMP.B  #$1F,D0       ; posizione del fascio == $1F o $11F?
            BNE.S  WaitLoop2     ; no, riprova ->
            AND.W  #$100,D0      ; posizione del fascio == $11F?
            BNE    Ready         ; si, è PAL: esci ->
            MOVEQ  #$01,D1       ; no, è NTSC: imposta il bit relativo
Ready:
            MOVE.L D1,D0         ; restituisci il tipo di video
            RTS
```

Notare che l'autore di questa routine **assume** che il multitasking sia disabilitato nel periodo di esecuzione della stessa. A suo dire il tutto funziona egregiamente. Ovviamente non basta scrivere e compilare il tutto. È necessario dare in pasto il programma ad un *Kickstart changer*, cioè ad un programmino che sostituisca questa procedura a quella originale nel Kickstart, badando a modificare opportunamente il checksum del blocco interessato. Spero che la spiegazione sia stata chiara. Quelli che, almeno a me, sono risultati meno chiari, sono i criteri di scelta per i valori di controllo, sia quelli usati dalla Commodore ($32, $64, $10e), sia quelli utilizzati da Martin ($20, $1F, $120, $11F). Da parte mia aspetterò che la Commodore si decida a risolvere ufficialmente il problema una volta per tutte...

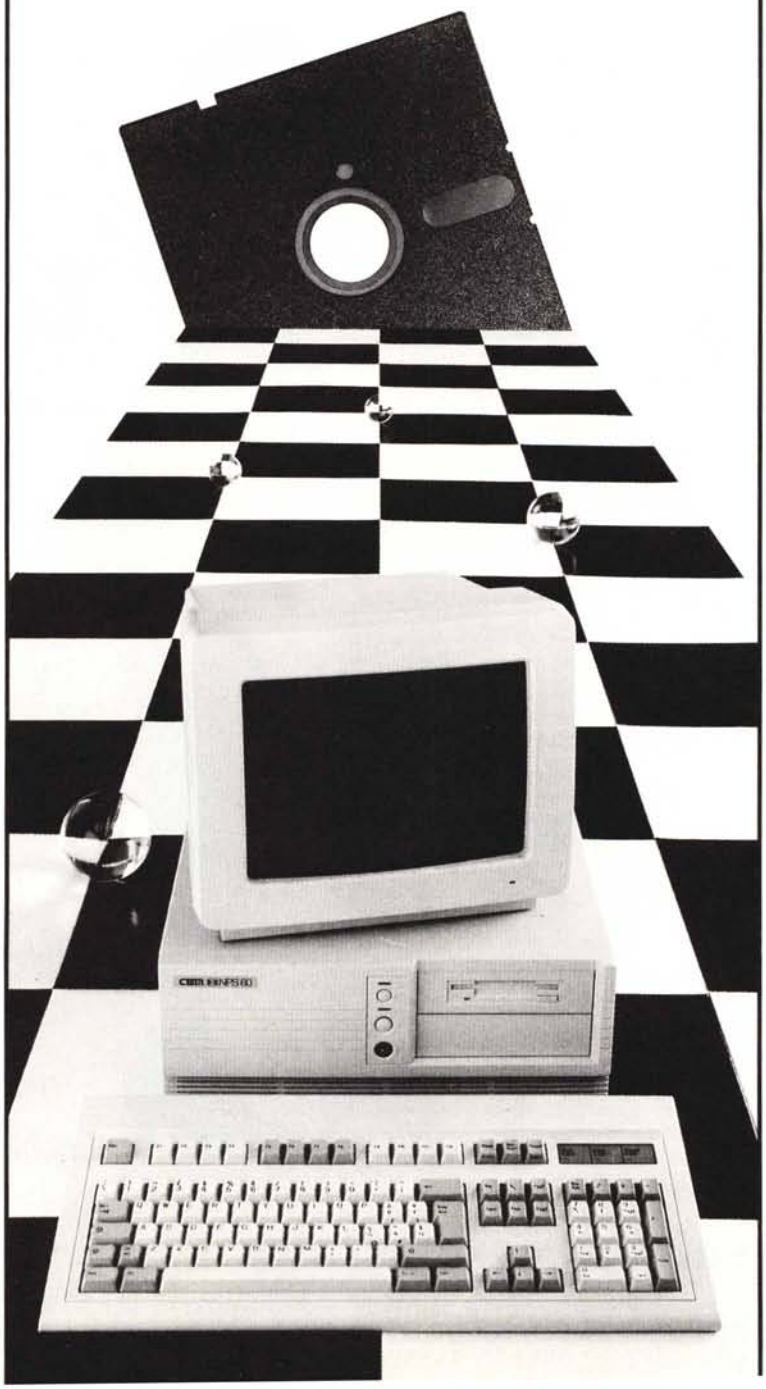
L'affidabilità (panorama)

NPC 30

NPS 60 - 10 MHz

NPS 60 - 16 MHz

HPS 70 - 20 MHz



Sicurezza completa, qualità originale, avanguardia tecnologica, eccellenti prestazioni. Particolari determinanti di un unico inconfondibile panorama: l'affidabilità.

Un concetto importante realizzato da CBM nei suoi hardware destinati a chi, affidando il suo lavoro ad un PC, sa ben distinguere i vantaggi esclusivi di certe prospettive:

- progettazione secondo criteri di avanguardia tecnologica supportati dalla ricerca autonoma di qualificati professionisti;
- costruzione secondo criteri di progettazione industriale che si avvale dell'utilizzo di componenti di altissima qualità;
- distribuzione solo dopo controlli diretti sui componenti e sull'insieme;
- compatibilità con gli standard di mercato;
- equipaggiamento di firmware originale garantito dalla sua origine.

Una vista completa sulle opportunità CBM, un azienda italiana presente da anni nel mercato dei prodotti per ufficio, con un partner colosso mondiale dell'elettronica.

Soltanto gli specialisti dell'affidabile nel particolare potevano offrire un panorama totale di affidabilità.

CBM S.p.A
DIVISIONE INFORMATICA

Distributore per l'Italia di KYOCERA stampanti laser.

Via Paolo Di Dono, 3/A - 00143 Roma - Tel. 50393.1 (R.A.)
Telex 611174 CBM SPA I - Fax 50393205

SCHLATTER & FABBRI

"un amico su cui... "CONTARE"

PRESENTI ALLO SMAU '89 PADIGLIONE 24 SALONE 1 POSTEGGIO B1/B3

IL vostro computer deve essere più di una macchina capace soltanto di contare. Deve conservare e gestire i vostri dati più preziosi. Vi deve aiutare nel lavoro, non vi deve tradire mai. Ma un computer, anche con il software più completo, resta una macchina capace di contare. Sono l'organizzazione, l'assistenza, la capacità di consigliarvi e di aiutarvi che danno vita al vostro computer e lo rendono un amico. Entrare in un negozio potendo acquistare i sistemi più attuali con garanzia totale, disporre di un servizio di assistenza rapido ed economico, scegliere una macchina assemblata su misura per il proprio lavoro con consegna immediata, è la sicurezza garantita dalla nostra organizzazione. Computer Discount vi offre tutto questo in una catena di negozi dove non si risparmia soltanto denaro...

COMPUTER DISCOUNT

COMPUTER DISCOUNT

BOLOGNA - 40139 - Viale Lenin, 12 c/d - Tel. 051/494103 - FAX 051/540293 — FIRENZE - 50121 - Viale Matteotti, 9 - Tel. 055/5000101 - FAX 055/587765 - GENOVA - 16151 - Viale D.G. Storace 4/r - Sampierdarena - Tel. 010/6459538 — MILANO - 20154 - Via Cenisio, 12 Tel. 02/33100204 - FAX 02/33100835 - PISA - 56100 - Viale A. Gramsci, 13 - Tel. 050/41580 - Fax 050/42072