Questo mese più che una utility particolarmente originale (la medesima funzione è svolta ormai da diversi tool) pubblichiamo un esempio di programmazione Amiga di basso livello (non certo qualitativo, s'intende!) dal quale prendere spunti per ulteriori realizzazioni. Da segnalare, con pieno compiacimento, la ricchezza di commenti che accompagnano il listato che meglio illustrano le tecniche adoperate più di qualsiasi articolo. L'autore del lavoro pubblicato è noto nel mondo Amiga per aver rilasciato nel public domain alcuni programmi abbastanza interessanti, di cui abbiamo già parlato sulle pagine di MC nella apposita rubrica. Buona lettura... e un attento studio del listato!

Amiga-M

di Nicola Salmoria - Siena

Come noto, ci sono due combinazioni di tasti particolari, Amiga sinistro-N e Amiga sinistro-M, che Intuition sfrutta per gestire la posizione dello schermo del WorkBench in relazione agli altri schermi; Amiga-N porta il WB davanti a tutti, Amiga-M lo spedisce dietro. Questa possibilità si rivela particolarmente utile con certi programmi che aprono schermi privi degli usuali gadget di profondità. Però il comportamento di questi due tasti non è sufficientemente flessibile quando sono aperti più schermi: se ad esempio uno schermo «sordo» si trova davanti a un altro, non c'è

```
/*************************************************************************

Programma..... Amiga-M

Versione ..... 1.0, 30-Aug-90

Autore ....... Nicola Salmoria
               Via Piemonte 11
               53100 Siena ITALY

Funzione ..... Mandare il primo schermo diatro a tutti alla pressione di
               Amiga sinistro-M

Compilatore..... Aztec C Compiler V5.0a

Compilazione..... cc Amiga-M.c -3
                  ln Amiga-M.o -lc16

*************************************************************************/

/* NOTA IMPORTANTE: questo programma puo` essere compilato solo con la versio
ne
     5.0a (o successive) dell'Aztec C, in quanto sfrutta le nuove #pragma
     disponibili solo da questa versione. Per compilare con Lattice, e` necessa
rio
     sostituire la #pragma con l'equivalente; il resto non dovrebbe aver bisogn
o
     di grosse modifiche.
     Per compilare con un compilatore privo delle istruzioni #pragma, bisogna
     inserire una piccola routine in assembler come spiegato su MC89. */

#include "exec/ports.h"
#include "exec/interrupts.h"
#include "intuition/intuitionbase.h"
#include "devices/input.h"
#include "devices/inputevent.h"
#include "libraries/dosextens.h"
#include "functions.h"


struct IntuitionBase *IntuitionBase;


/* Questo e` il prototipo dell'handler che inseriremo nella catena. */
struct InputEvent *myhandler(struct InputEvent *,struct MemEntry **);

/* E questa e` la #pragma che ci permette di non scrivere la funzione in */
/* assembler. Inoltre il registro A4 verra` inizializzato correttamente allo
*/
/* ingresso nella funzione, senza bisogno di chiamare esplicitamente geta4().
*/
#pragma intfunc(myhandler(a0,a1))


BYTE portname[] = "Amiga-M.port";       /* Il port verra` usato per spedire i
l */
        /* comando all'input.device, quindi il signal bit sara` liberato e */
        /* il port rimarra` 'muto' solo per segnalare la nostra presenza, ed
*/
        /* evitare che l'handler sia aggiunto piu` volte. */

struct MsgPort inputdevport;
struct IOStdReq inputreqblock;
struct Interrupt interrupt;


/* Visto che non abbiamo bisogno del parsing della linea di comando e che */
/* non usiamo funzioni del C, possiamo usare _main() invece di main(). */
_main()
{
register BPTR outp;
register LONG sig;
register struct Task *thistask;
```

modo di riportare davanti quest'ultimo. Amiga-M (il programma) modifica leggermente il comportamento della omonima combinazione di tasti: invece di mandare dietro il WB, manda dietro il PRIMO schermo, qualunque esso sia. In questo modo, premendo più volte Amiga-M si possono scorrere tutti gli schermi in sequenza. Per ottenere questo, Amiga-M aggiunge un handler alla catena dell'input.device. L'input .device è già stato descritto da Maurizio Mangrella su MC89, quindi non ci ritornerò sopra; il listato mostra però qualcosa di nuovo, cioè come usare le nuove # pragma dell'Aztec C 5.0a per aggiungere un handler senza scrivere una sola riga di Assembler — senza ricorrere, cioè, alla classica routine che mette i parametri nello stack e chiama la funzione in C. Il listato è ampiamente commentato, quindi non ci dovrebbero essere problemi per la sua comprensione; vorrei però richiamare l'attenzione su una linea, la penultima di _main(). Quando si vogliono caricare in memoria in maniera permanente dei programmi, che ad esempio modificano qualche funzione del sistema operativo o, come in questo caso, aggiungono un handler all'input.device, ci sono svariati modi di procedere. Il più semplice è senz'altro quello di inserire una Wait(0L) alla fine del programma, dimodoché questo non terminerà mai; in questo modo, però, c'è lo svantaggio che il programma deve essere lanciato con «run»; inoltre, occuperà permanentemente una CLI, e questo non è bene visto che il massimo numero di CLI che si possono aprire contemporaneamente è 20 (in realtà si può arrivare a 255, ma non è questo il luogo per discuterne).

Un altro metodo è quello di allocare un po' di memoria e copiarci la parte di programma che deve rimanere in memoria; sotto l'aspetto del consumo di memoria è senz'altro il metodo migliore, ed in Assembler è il più usato; col C però ci possono essere dei problemi con la gestione delle variabili. Il metodo che ho usato io è un compromesso fra i due, ed è una semplificazione di quello che fanno i moduli di startup 'detach.o' dell'Aztec e 'cback.o' del Lattice. Quando si carica un programma da CLI, il DOS memorizza nella corrispondente struttura CommandLineInterface il puntatore alla lista dei segmenti occupati. Quando il programma termina, il DOS recupera quel valore e libera la memoria corrispondente. Prima di uscire, io azzero quel campo, cosicché il DOS non liberi la mia memoria. In questo modo l'handler e i suoi dati rimangono pacifi-

```
    outp = Output();        /* Prendiamo l'handle di output. */

    if (!(IntuitionBase =
                (struct IntuitionBase *)OpenLibrary("intuition.library",0L)))
        Exit(20L);

    /* Disabilito il multitasking per assicurarmi che un'altra copia del */
    /* programma non aggiunga il port DOPO che ho controllato che non ci sia */
    /* e PRIMA che lo aggiunga io... ho reso l'idea? Perche` uno dovrebbe */
    /* fare una cosa cosi` perversa e` un altro discorso... */
    Forbid();

    if (FindPort(portname))
    {
    /* Amiga-M e` gia` stato lanciato, quindi usciamo senza aggiungere l'handler.
 */
        Write(outp,"Amiga-M already installed!\n",27L);
        Permit();       /* Riabilitiamo il multitasking. */
        Exit(5L);       /* Solo un WARN, nulla di grave. */
    }

    /* Allochiamo un signal bit per il port. */
    if ((sig = AllocSignal(-1L)) == -1)
    {

        CloseLibrary(IntuitionBase);
        Permit();
        Exit(20L);
    }

    /* Troviamo l'indirizzo di ... noi stessi. */
    thistask = FindTask(NULL);

    /* Adesso inizializziamo il port e aggiungiamolo alla lista dei port pubblici
. */
    inputdevport.mp_Node.ln_Name = portname;
    inputdevport.mp_Node.ln_Type = NT_MSGPORT;
    inputdevport.mp_Flags = PA_SIGNAL;
    inputdevport.mp_SigBit = sig;
    inputdevport.mp_SigTask = thistask;
    AddPort(&inputdevport);

    /* Ora possiamo riabilitare il multitasking. */
    Permit();


    /* Adesso prepariamo l'IOStdRequest da spedire all'input.device. */
    inputreqblock.io_Message.mn_Node.ln_Type = NT_MESSAGE;
    inputreqblock.io_Message.mn_Length = sizeof(struct IOStdReq);
    inputreqblock.io_Message.mn_ReplyPort = &inputdevport;


    /* Apriamo l'input.device. */
    if (OpenDevice("input.device",0L,(struct IORequest *)&inputreqblock,0L))
    {
        FreeSignal(sig);
        RemPort(&inputdevport); /* togliamo il port, riproveremo... */
        CloseLibrary(IntuitionBase);
        Exit(20L);
    }


    /* La struttura Interrupt fornisce all'input.device nome, indirizzo e */
    /* priorita` dell'handler che vogliamo inserire. */
    interrupt.is_Node.ln_Pri = 51;  /* Subito prima di Intuition. */
    interrupt.is_Node.ln_Name = "Amiga-M";
    interrupt.is_Data = NULL;       /* Non abbiamo bisogno di dati aggiuntivi. */
    interrupt.is_Code = (PVF)myhandler;

    /* Finalmente aggiungiamo l'handler alla lista. */
    inputreqblock.io_Command = IND_ADDHANDLER;
    inputreqblock.io_Data = (APTR)&interrupt;

    DoIO((struct IORequest *)&inputreqblock);

    /* L'input.device non ci serve piu`, chiudiamolo. */
    CloseDevice((struct IORequest *)&inputreqblock);

    /* Il port non ci servira` piu` tranne che per indicare la nostra presenza, *
/
    /* quindi liberiamo il signal bit  e cambiamo la struttura di conseguenza. */
    /* E` importante porre i flags a PA_IGNORE, visto che tra poco questo Task */
    /* non esistera` piu` e un eventuale segnale non avrebbe una valida struttura
 */
    /* Task dove essere memorizzato. */
    inputdevport.mp_SigBit = 0;
    inputdevport.mp_Flags = PA_IGNORE;
    inputdevport.mp_SigTask = NULL;
    FreeSignal(sig);

    /* Messaggio di copyright... notare l'uso delle sequenze ANSI per cambiare il
 */
    /* colore del testo. */
    Write(outp,"\033[33mAmiga-M V1.0\033[31m Copyright ) 1990 by Nick! Salmoria\n
Via Piemonte 11, 53100 Siena ITALY   (0577)54164\n",107L);

    /* Questa linea e` piuttosto criptica. 1 minuto di tempo per capire cosa fa!
 */

    ((struct CommandLineInterface *)BADDR(((struct Process *)thistask)->pr_CLI))-
>cli_Module = NULL;

    /* Capito il trucco? Attraverso le perverse strutture del DOS risaliamo al */
    /* puntatore del nostro codice e lo azzeriamo. In questo modo, all'uscita il
 */
    /* DOS non liberera` la memoria allocata. Questo trucchetto ci permette di */
    /* risparmiare la fatica di allocare memoria per l'handler e ricopiarcelo. */
    /* Ovviamente in questo modo si spreca un po' di memoria, ma con programmi */
    /* cosi` piccoli la cosa e` ammissibile. */
```

```
    Exit(0L);       /* Fine. Ma il codice rimarra` in memoria. */
    }


    /* Questo e` l'handler che aggiungiamo alla catena */
    /* Il compilatore provvedera` automaticamente a generare codice in modo che *
/
    /* i parametri siano riconosciuti passati nei registri A0, A1 e che il */
    /* registro A4 sia inizializzato al corretto valore. */

    struct InputEvent *myhandler(register struct InputEvent *ie,struct MemEntry *
*me)
    {
    register struct InputEvent *first,*prev;

    /* Cerchiamo attraverso tutta la catena di eventi quelli che ci interessano.
*/
    for (first = ie,prev = 0;ie;ie = ie->ie_NextEvent)
    {
            if (ie->ie_Class == IECLASS_RAWKEY && ie->ie_Code == KEYCODE_M &&
                        (ie->ie_Qualifier & AMIGALEFT))
            {
    /* Tasto M premuto insieme ad Amiga sinistro. Notare che non converto il */
    /* codice del tasto tramite la keymap del sistema. Quindi l'handler entra in
*/
    /* azione solo se si preme il tasto che nella tastiera americana e` una M, a
*/
    /* prescindere dal valore che ha realmente. Non converto il codice perche` */
    /* neanche Intuition lo fa! */

                    if (!(ie->ie_Qualifier & IEQUALIFIER_REPEAT))
    /* Portiamo lo schermo dietro solo se il tasto e` stato effettivamente */
    /* premuto; il repeat non conta. */
                        ScreenToBack(IntuitionBase->FirstScreen);

    /* Rimuoviamo questo messaggio dalla catena, non deve arrivare a Intuition. *
/
                    if (prev) prev->ie_NextEvent = ie->ie_NextEvent;
                    else first = ie->ie_NextEvent;
            }
            else prev = ie;
    }

    return(first);
    }
```

camente in memoria (in effetti, non è più possibile rimuoverli). C'è lo svantaggio che rimane allocata tutta la memoria occupata dal programma, compresa quindi quella che non serve più. Con programmi così piccoli, comunque, la cosa è senz'altro accettabile. Bene, è tutto. Questo sorgente può essere senz'altro un buon punto di partenza per aggiungere le vostre routine all'input-.device: tutto quello che dovete fare è modificare la routine myhandler(). Ricordate che da dentro myhandler() potete chiamare quasi tutte le funzioni di Amiga (purché prima apriate le librerie, ovviamente) tranne quelle della dos.library. Infatti myhandler() viene chiamata da 'input.device', che è un semplice task (ATTENZIONE: è diverso dal DEVICE input.device! Questo TASK viene lanciato dal DEVICE quando viene aperto la prima volta), mentre le funzioni del DOS non possono essere chiamate da nulla di meno di un process.