La Input.Device

di Maurizio Mangrella - Eboli

Sono le undici di sera... dopo estenuanti compilazioni, mi ritrovo con gli occhi sbarrati di fronte al monitor... Accidenti, non trovo i sorgenti... ah eccoli qua, sembrano vecchi di tre anni, eppure li ho stampati proprio ora, si vede che la mia scrivania-ciclone invecchia le cose (e anche me)... Sì, ne è valsa la pena: la input.device merita altro che un'intera giornata di compilazioni... Però, che giornata...

Impossessiamoci dell'Amiga

Credere che retribuire il rivenditore di fiducia nell'acquistare un Amiga sia sufficiente a «possederlo» è un errore grossolano. Il «possesso» di un oggetto consiste, a mio parere, nel conoscerne — e saperne sfruttare — tutte le caratteristiche.

Un valido espediente — sebbene non facilissimo — per possedere l'Amiga è aprire la input.device e aggiungere un InputHandler nella catena del S.O. Mi spiego meglio: tramite la input.device potremo inserire un segmento del nostro programma in una «nicchia» privilegiata e far sì che il Sistema Operativo lo chiami ogni volta che si verifica qualcosa (in gergo un «evento»). Premetto subito che l'argomento non è dei più semplici: anzitutto bisogna avere un po' di dimestichezza con la programmazione in Assembly. Per acquisire questa praticità è necessario fare molto esercizio: prima programmi brevi, poi sempre più lunghi e complessi, fino... alla input .device, che non è affatto un traguardo finale, ma una buona Maglia Rosa. Chi, poi, è più esperto, può addolcirsi la vita con un po' di programmazione multilinguaggio, che non guasta mai. Nel corso dell'articolo accenneremo anche a questo aspetto.

Comando	Valore	Funzione
IND_ADDHANDLER	9	Aggiunge un Input.Handler alla catena
IND_REMHANDLER	10	Toglie un InputHandler
IND_WRITEEVENT	11	Scrive un evento
IND_SETTHRESH	12	Setta la soglia (di che??)
IND_SETPERIOD	13	Setta il periodo (di che??)
IND_SETMPORT	14	Setta la porta del mouse
IND_SETMTYPE	15	Setta il tipo di controller del mouse
IND_SETMTRIG	16	Setta il trigger del mouse

Tabella A

La Input.Device

Per aprire la input.device la procedura è la solita: da C daremo quanto è rappresentato in figura 1.

Noterete le funzioni della libreria amiga.lib (CreatePort() e CreateStdIO()) nonché l'assoluta inesistenza di una inputevent.device. Il file INCLUDE "devices/inputevent.h", infatti, non corrisponde a un omonimo device, ma contiene parecchie definizioni interessanti, che ci saranno utili in seguito.

I comandi della input.device sono riportati in tabella A.

Nel corso dell'articolo preciseremo le modalità di funzionamento di ciascuno dei comandi sopra visti, con particolare riguardo alla possibilità di aggiungere un Handler.

```
#include <exec/ports.h>
#include <exec/io.h>
#include <devices/input.h>
#include <devices/inputevent.h>

main() {
  struct MsgPort *myport;
  struct IOStdReq *myreq;

  myport = (struct MsgPort *)CreatePort("MyPort",0);
  myreq = (struct IOStdReq *)CreateStdIO(myport);
  OpenDevice("input.device",0,myreq,0);

}
```

Figura 1

Aggiungiamo un InputHandler

Come abbiamo detto, un InputHandler è un segmento di programma in Linguaggio Macchina (come risulta dall'Assembling o dalla compilazione) che il S.O. chiama automaticamente ogniqualvolta succede qualcosa, QUALUNQUE cosa. Nel caso specifico dell'Amiga, più InputHandler si uniscono per formare una «catena»: ogni «anello» della catena viene chiamato prima del successivo. L'ordine in cui gli InputHandler vengono chiamati è dettato da una loro priorità: gli InputHandler a priorità bassa vengono chiamati per ultimi. Se proprio ci tenete a saperlo, sappiate che la priorità di Intuition è 50: pertanto tutti gli InputHandler con priorità inferiore a 50 verranno chiamati dopo Intuition, mentre quelli con priorità superiore a 50 verranno chiamati prima.

Anzitutto, vediamo come aggiungere un InputHandler alla catena. Prepariamo una struct Interrupt, così conformata:

```
struct Interrupt {
    struct  Node is_Node;
    APTR    is_Data;
    VOID    (*is_Code)();
}

  Il nodo is_Node, cosi' strutturato:

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

Contiene informazioni interessanti, ma l'unica realmente utile è la priorità (il cui significato abbiamo già visto). is_Data contiene un dato a 32 bit (di qualunque genere: ricordo che APTR è un puntatore a un non meglio definito array di long-word, dunque potete farci quello che volete) che ci verrà passato nel registro A1 (vedi dopo), mentre is_Code è l'indirizzo del nostro InputHandler. Se, ad esempio, il nostro InputHandler si chiama HandlerInterface() e l'is_Data di cui abbiamo bisogno è l'indirizzo di una struttura XYZ, da C daremo:

```
struct Interrupt HandlerInt;
struct XYZ ( eccetera, eccetera... )
  .
  .
HandlerInt.is_Code = (VOID *)HandlerInterface;
HandlerInt.is_Data = (APTR)&XYZ;
HandlerInt.is_Node.ln_Pri = 51;
```

Notate i cast (per evitare fiere proteste del compilatore) e notate inoltre il modo di settare la priorità: con il valore 51 il nostro Handler sarà eseguito subito prima di Intuition.

Infine, aggiungeremo l'Handler così:

```
myreq->io_Data = (APTR)&HandlerInt;
myreq->io_Length = sizeof(struct Interrupt);
myreq->io_Command = IND_ADDHANDLER;
DoIO(myreq);
```

Tutti saprete che ogni microprocessore è dotato di registri: quelli del 68000 (lasciando da parte quelli speciali) sono 16, 8 dati (numerati da D0 a D7) e 8 indirizzi (numerati da A0 a A7). A7 è anche lo Stack Pointer (SP), ovvero il puntatore ad un'area di memorizzazione temporanea dei dati con funzionamento LIFO (Last-In-First-Out), proprio come una «catasta» (stack in inglese significa catasta). Quando il S.O. chiama il nostro InputHandler ci passerà l'evento corrente in A0, sotto forma di indirizzo ad una struct InputEvent conformata come in figura 2.

Class è il parametro principale, indicante il tipo di evento accaduto; in figura 3 tutti i possibili valori.

Come vedete, la scelta è ampia. Ricordo che per «rinfrescare» una finestra si intende restaurarne il contenuto dopo una operazione di resizing o di spostamento. Ricordo anche che alcune Class hanno associato, in ieEventAddress, un indirizzo: in particolare, quando le Preferences vengono cambiate, viene passato l'indirizzo delle nuove. Così, per le Classes degli eventi legati a una finestra, viene passato l'indirizzo del Window Record relativo alla finestra interessata. Tenete comunque presente che tutti questi eventi (quelli con Class maggiore di 6) sono disponibili solo dopo il pre-processing di Intuition: per riceverli dovete settare, per il vostro InputHandler, una priorità inferiore a 50.

SubClass, nelle attuali versioni del S.O., è sempre a 0. Code contiene un codice riguardante l'evento accaduto; i valori interessanti per il nostro discorso sono rappresentati in figura 4.

Ad ogni tasto corrisponde un Code particolare, che — purtroppo — non è ASCII: ho raffigurato i codici in figura 5. Gli altri valori si autocommentano, dun-

```
struct InputEvent {
    struct  InputEvent *ie_NextEvent;
    UBYTE   ie_Class;
    UBYTE   ie_SubClass;
    UWORD   ie_Code;
    UWORD   ie_Qualifier;
    union {
    struct {
       WORD    ie_x;
       WORD    ie_y;
    } ie_xy;
    APTR    ie_addr;
    } ie_position;
    struct timeval ie_TimeStamp;
}

#define   ie_X              ie_position.ie_xy.ie_x    /* Definizioni utili */
#define   ie_Y              ie_position.ie_xy.ie_y
#define   ie_EventAddress   ie_position.ie_addr
```

Figura 2

Class	Valore	Evento
IECLASS_NULL	0x00	Nessun evento (interrupt spurio)
IECLASS_RAWKEY	0x01	Pressione/rilascio di un tasto
IECLASS_RAWMOUSE	0x02	Spostamento del mouse
IECLASS_EVENT	0x03	Mandato quando una window viene attivata
IECLASS_POINTERPOS	0x04	Posizione del puntatore del mouse
IECLASS_Unused	0x05	Non usato
IECLASS_TIMER	0x06	Timer Event (mandato ogni decimo di secondo)
IECLASS_GADGETDOWN	0x07	Un gadget e' stato premuto
IECLASS_GADGETUP	0x08	Un gadget e' stato lasciato
IECLASS_REQUESTER	0x09	E' stato attivato un AutoRequester
IECLASS_MENULIST	0x0A	E' stata selezionata una voce da un Menu
IECLASS_CLOSEWINDOW	0x0B	Una finestra e' stata chiusa
IECLASS_SIZEWINDOW	0x0C	Una finestra e' stata ridimensionata
IECLASS_REFRESHWINDOW	0x0D	Una finestra e' stata "rinfrescata"
IECLASS_NEWPREFS	0x0E	Le Preferences sono state cambiate
IECLASS_DISKREMOVED	0x0F	Un dischetto e' stato espulso
IECLASS_DISKINSERTED	0x10	Un dischetto e' stato inserito
IECLASS_ACTIVEWINDOW	0x11	Una finestra e' stata attivata (*)
IECLASS_INACTIVEWINDOW	0x12	Una finestra e' stata disattivata (*)

(*) Solo per il KickStart 1.2

Figura 3

```
*****************************************************************
* InputEvent Test      by Maurizio Mangrella  1989              *
*                                                               *
*  Questo programma in Assembler e un demo della input.device,  *
* ed e una riscrittura di un demo di Rob Peck, Commodore-Amiga, Inc. *
*  La dimostrazione consiste nel salvare a un file gli eventi passati *
* al programma attraverso un Handler aggiunto alla 'catena' di  *
* Intuition, con priorita maggiore di quella di Intuition: per-  *
* tanto il nostro Handler sara eseguito prima di Intuition (quale *
* onore!!!).                                                    *
*---------------------------------------------------------------*
*  USO: SaveEvents > (nome del file)                            *
*                                                               *
*  Salva gli eventi verificatisi al file (nome del file). Per como- *
* dita di programmazione ho fatto uso della redirezione del DOS. *
*  NOTA: Se non specificate la redirezione ('>') gli eventi verranno *
* indirizzati su _stdout, con le conseguenze immaginabili.      *
*---------------------------------------------------------------*
*  Potrete notare come non si sia fatto uso delle direttive     *
* STRUCTURE: in effetti mi e sembrata un'inutile complicazione. Cosi *
* com'e, il programma gira perfettamente anche link-ato con la sola *
* amiga.lib. Ricordo che le routine di questa libreria sono copyright *
* della Commodore-Amiga, Inc.                                   *
*---------------------------------------------------------------*
*                 ! c by Maurizio Mangrella  1989 !             *
*****************************************************************

* Costanti di uso generale

_SysBase        EQU       4                   ;Riferimento assoluto a Exec

ID_ADDHANDLER   EQU       9                   ;Comando di aggiunta di un handler
ID_REMHANDLER   EQU       10                  ;Comando di rimozione handler

IECLASS_TIMER   EQU       6                   ;Class di un Timer Event
IECLASS_RWKEY   EQU       1                   ;Class di un RAWKey
IECODE_ESC      EQU       $45                 ;Code del tasto ESC rialzato

* Offsets per una IOStdReq

io_Message      EQU       0
io_Device       EQU       20
io_Unit         EQU       24
io_Command      EQU       28
io_Flags        EQU       30
io_Error        EQU       31
io_Actual       EQU       32
io_Length       EQU       36
io_Data         EQU       40
io_Offset       EQU       44

* Routines di cui faremo uso

                XREF      _LVOOpenLibrary
                XREF      _LVOOutput
                XREF      _LVOOpenDevice
                XREF      _CreatePort
                XREF      _CreateStdIO
                XREF      _LVODoIO
                XREF      _LVOWrite
                XREF      _LVOCloseLibrary
                XREF      _LVOCloseDevice
                XREF      _DeletePort
                XREF      _DeleteStdIO
                XREF      _LVOForbid
                XREF      _LVOCopyMem
                XREF      _LVOPermit

                XDEF      _SysBase            ;Per le routines della amiga.lib


* Sezione di inizializzazione dello
* Handler. Vengono aperte librerie,
* ports e devices e  opportunamente
* inizializzate. Fa uso di funzioni
* della libreria amiga.lib  (Commo-
* dore-Amiga, Inc.).

Main            MOVE.L    _SysBase,A6         ;Preleva SysBase
                MOVEQ     #0,D0               ;Qualunque versione va bene
                LEA       DosName,A1          ;Nome della library (dos.library)
                JSR       _LVOOpenLibrary(A6) ;Apre la libreria
                MOVE.L    D0,A6               ;Prende il puntatore ...
                MOVE.L    A6,DOSBase          ;... e lo mette in DOSBase
                JSR       _LVOOutput(A6)      ;'Initial Output FileHandle'
                MOVE.L    D0,FileHandle       ;Lo conserva in FileHandle
                MOVE.L    #0,-(SP)            ;Flags = PA_SIGNAL
                PEA       PortName            ;Nome del port (InputDevPort)
                JSR       _CreatePort         ;Crea il port
                MOVE.L    D0,MsgPort          ;Conserva il puntatore in MsgPort
                ADDQ.L    #8,SP               ;Aggiusta lo stack
                MOVE.L    D0,-(SP)            ;Indirizzo del port
                JSR       _CreateStdIO        ;Crea una IOStdReq
                MOVE.L    D0,IOStdReq         ;Conserva il puntatore in IOStdReq
                ADDQ.L    #4,SP               ;Aggiusta lo stack
                LEA       DevName,A0          ;Nome del device (input.device)
                MOVE.L    IOStdReq,A1         ;Indirizzo della IOStdReq
                MOVEQ     #0,D0               ;Flags = 0
                MOVEQ     #0,D1               ;Unit = 0
                MOVE.L    _SysBase,A6         ;Routine di Exec
                JSR       _LVOOpenDevice(A6)  ;Apre il device
                LEA       InputEvent,A0       ;L'indirizzo del nostro InputEvent
                MOVE.L    A0,is_Data          ;Sistema l'indirizzo in is_Data
                LEA       MyHandler,A0        ;Il nostro Handler
                MOVE.L    A0,is_Code          ;Ne sistema l'indirizzo in is_Code
                MOVE.L    IOStdReq,A1         ;La nostra IORequest
                LEA       Interrupt,A0        ;Il nostro Interrupt Stuff
                MOVE.L    A0,io_Data(A1)      ;Indirizzo di Interrupt in io_Data
                MOVE.W    #ID_ADDHANDLER,io_Command(A1) ;Comando
                JSR       _LVODoIO(A6)        ;Aggiunge l'handler

* Loop di pre-attesa. Attende l'ar-
* rivo del  primo evento e ne salva
* il TimeStamp.

                MOVE.L    DOSBase,A6          ;Per le routines della dos.library
InitLoop        TST.W     NewEvent            ;Attende l'arrivo di un evento
                BEQ.S     InitLoop
                CLR.W     NewEvent            ;Resetta il flag
                LEA       ie_Time_Secs,A0     ;Forma l'indirizzo di timeval
                MOVE.L    A0,D2               ;D2 = Output Buffer
                MOVE.L    FileHandle,D1       ;Preleva FileHandle
                MOVEQ     #8,D3               ;Salva due long-words
                JSR       _LVOWrite(A6)       ;Salva il timeval

* Loop di attesa.  Salva gli eventi
* passati dall'Handler. Il program-
* ma finisce quando viene premuto e
* rilasciato il tasto ESC.

Loop            TST.W     NewEvent            ;E' pronto un nuovo evento?
                BEQ.S     Loop                ;No: aspetta
                CLR.W     NewEvent            ;Resetta il flag
                CLR.L     D0                  ;D0.L = 0
                MOVE.B    ie_Class,D0         ;Preleva Class
                CMPI.B    #IECLASS_TIMER,D0   ;E' un Timer Event?
                BEQ.S     Loop                ;Si: non lo salva
                CMPI.B    #IECLASS_RWKEY,D0   ;E' un RAWKey Event?
                BNE.S     Continues           ;No: continua
                MOVE.W    ie_Code,D0          ;Preleva Code
                CMPI.W    #IECODE_ESC,D0      ;E' il tasto ESC rialzato?
                BEQ.S     Ending              ;Si: termina il programma
Continues       MOVE.L    FileHandle,D1       ;Preleva il File Handle
                LEA       InputEvent,A0       ;Preleva l'indirizzo di InputEvent
                MOVE.L    A0,D2               ;D2 = Output Buffer
                ADDQ.L    #4,D2               ;Salta ie_NextEvent
                MOVEQ     #18,D3              ;sizeof(struct InputEvent)-4
                JSR       _LVOWrite(A6)       ;Salva l'evento
                BRA.S     Loop                ;Ricomincia daccapo

* Fine  programma. Il programma to-
* glie l'Handler  sistemato, chiude
* tutte  le strutture,  libraries e
* devices  aperte e restituisce  il
* controllo al DOS.

Ending          MOVE.L    DOSBase,A1          ;DOS Base Library Pointer
                MOVE.L    _SysBase,A6         ;Una routine di Exec
                JSR       _LVOCloseLibrary(A6) ;Chiude la dos.library
                LEA       Interrupt,A0        ;Indirizzo di Interrupt
                MOVE.L    IOStdReq,A1         ;Indirizzo della nostra IOStdReq
                MOVE.L    A0,io_Data(A1)      ;Indirizzo in io_Data
                MOVE.W    #ID_REMHANDLER,io_Command(A1) ;Togli l'Handler!
                JSR       _LVODoIO(A6)        ;Esegue il comando
                MOVE.L    IOStdReq,A1         ;IOStdReq Address in A1
                JSR       _LVOCloseDevice(A6) ;Chiude la input.device
                MOVE.L    MsgPort,-(SP)       ;MsgPort Address sullo stack
                JSR       _DeletePort         ;Chiude il port
                ADDQ.L    #4,SP               ;Aggiusta lo stack
                MOVE.L    IOStdReq,-(SP)      ;E adesso chiudiamo ...
                JSR       _DeleteStdIO        ;... la IOStdReq
                ADDQ.L    #4,SP               ;Riaggiusta lo stack
                MOVEQ     #0,D0               ;RETURN_OK
                RTS                           ;Buonanotte!

* L'Handler.  Copia il primo evento
* della catena che gli viene passa-
* ta.
* L'uso dello  stack ha creato pro-
* blemi  in  run-time,  pertanto ho
* ripiegato  su una  memorizzazione
* temporanea   nell'area  dati  del
* programma.

MyHandler       MOVE.L    A0,SaveA0           ;Salva i registri
                MOVE.L    A1,SaveA1
                MOVE.L    _SysBase,A6         ;Una routine di Exec
                JSR       _LVOForbid(A6)      ;Disabilitiamo gli Interrupts
                MOVEQ     #22,D0              ;22 bytes da copiare
                JSR       _LVOCopyMem(A6)     ;Copia l'Event
                JSR       _LVOPermit(A6)      ;Riabilita gli Interrupts
                MOVE.L    SaveA0,A0           ;Riprende i registri
                MOVE.L    SaveA1,A1
                MOVE.L    A0,D0               ;Ritorna l'indirizzo dell'Event
                MOVE.W    #1,NewEvent         ;Setta il flag di 'Nuovo Evento'
                RTS


* Area dati

DOSBase         DC.L      0                   ;Spazio per DOS Library Pointer

MsgPort         DC.L      0                   ;Spazio per MsgPort Pointer
IOStdReq        DC.L      0                   ;Spazio per IOStdReq Pointer

FileHandle      DC.L      0                   ;Spazio per FileHandle Pointer

SaveA0          DC.L      0                   ;Memorizzazione dei registri
SaveA1          DC.L      0

Interrupt:                                    ;Il nostro Interrupt Vector
is_Node_Succ    DC.L      0
is_Node_Pred    DC.L      0
is_Node_Type    DC.B      0
is_Node_Pri     DC.B      51                  ;Un gradino piu su di Intuition
is_Node_Name    DC.L      0
is_Data         DC.L      0
is_Code         DC.L      0

InputEvent:                                   ;Il nostro Input Event
ie_NextEvent    DC.L      0
ie_Class        DC.B      0
ie_SubClass     DC.B      0
ie_Code         DC.W      0
ie_Qualifier    DC.W      0
ie_X            DC.W      0
ie_Y            DC.W      0
ie_Time_Secs    DC.L      0
ie_Time_mScs    DC.L      0

NewEvent        DC.W      0                   ;Flag di 'Nuovo Evento'

DosName         DC.B      'dos.library',0     ;Il nome della dos.library
PortName        DC.B      'InputDevPort',0    ;Il nome del nostro port
DevName         DC.B      'input.device',0    ;Il nome della input.device

                CNOP      0,2

                END
```

que li tralasceremo.

Qualifier «qualifica» in maniera più specifica l'evento in corso: qualificatori importanti sono i tasti speciali e quelli del mouse. In figura 6 i Qualifiers, uno per uno.

Più Qualifiers possono essere passati contemporaneamente OR-andoli, o — il che è lo stesso, nel caso specifico — sommandoli.

Quando un tasto viene premuto, viene inviato il suo normale Code; quando viene rilasciato, viene di nuovo inviato il suo Code, ma maggiorato di IECODE_UP_PREFIX (80 hex., 128 dec.). In questo modo è possibile sapere quando un tasto viene rilasciato o premuto. Il tasto di Caps Lock ha un comportamento particolare: riporta il valore 62 hex. quando viene premuto (accendendo il LED), e viene considerato premuto fino a quando non viene premuto di nuovo per spegnere il LED: in tal caso viene mandato il codice E2 hex.

```
*****************************************************************
* InputEvent Test      by Maurizio Mangrella  1989              *
*                                                               *
*  Questo programma in Assembler e un demo della input.device,  *
* ed e una riscrittura di un demo di Rob Peck, Commodore-Amiga, Inc.  *
*  La dimostrazione consiste nel salvare a un file gli eventi passati  *
* al programma attraverso un Handler aggiunto alla 'catena' di  *
* Intuition, con priorita maggiore di quella di Intuition: per-  *
* tanto il nostro Handler sara eseguito prima di Intuition (quale  *
* onore!!). Il presente programma riprende gli eventi salvati con  *
* SaveEvents e li ripropone (quasi) uguali.                     *
*---------------------------------------------------------------*
*  USO: LoadEvents < (nome del file)                            *
*                                                               *
*  Ripropone gli eventi registrati al file (nome del file). Per como-  *
* dita di programmazione ho fatto uso della redirezione del DOS.  *
*  NOTA: Se non specificate la redirezione ('<') gli eventi verranno  *
* ripresi da _stdin, con le conseguenze immaginabili.           *
*---------------------------------------------------------------*
*  Potrete notare come non si sia fatto uso delle direttive     *
* STRUCTURE: in effetti mi e sembrata un'inutile complicazione. Cosi  *
* com'e, il programma gira perfettamente anche link-ato con la sola  *
* amiga.lib. Ricordo che le routine di questa libreria sono copyright  *
* della Commodore-Amiga, Inc.                                   *
*---------------------------------------------------------------*
*                    : c by Maurizio Mangrella  1989 :          *
*****************************************************************

* Costanti di uso generale

_SysBase        EQU       4                   ;Riferimento assoluto a Exec

ID_WRITEVENT    EQU       11                  ;Comando di scrittura di un evento

* Offsets per una IOStdReq

io_Message      EQU       0
io_Device       EQU       20
io_Unit         EQU       24
io_Command      EQU       28
io_Flags        EQU       30
io_Error        EQU       31
io_Actual       EQU       32
io_Length       EQU       36
io_Data         EQU       40
io_Offset       EQU       44

* Routines di cui faremo uso

                XREF      _LVOOpenLibrary
                XREF      _LVOInput
                XREF      _LVOOpenDevice
                XREF      _CreatePort
                XREF      _CreateStdIO
                XREF      _LVODoIO
                XREF      _LVORead
                XREF      _LVODelay
                XREF      _LVOCloseLibrary
                XREF      _LVOCloseDevice
                XREF      _DeletePort
                XREF      _DeleteStdIO

                XDEF      _SysBase



* Sezione di inizializzazione della
* input.device.  Vengono aperte li-
* brerie,  porta e devices e oppor-
* tunamente  inizializzate. Fa  uso
* di    funzioni   della    libreria
* amiga.lib (Commodore-Amiga, Inc.).

Main            MOVE.L    _SysBase,A6         ;Preleva SysBase
                MOVEQ     #0,D0               ;Qualunque versione va bene
                LEA       DosName,A1          ;Nome della library (dos.library)
                JSR       _LVOOpenLibrary(A6) ;Apre la libreria
                MOVE.L    D0,A6               ;Prende il puntatore ...
                MOVE.L    A6,DOSBase          ;... e lo mette in DOSBase
                JSR       _LVOInput(A6)       ;'Initial Input FileHandle'
                MOVE.L    D0,FileHandle       ;Lo conserva in FileHandle
                MOVE.L    #0,-(SP)            ;Flags = PA_SIGNAL
                PEA       PortName            ;Nome del port (InputDevPort)
                JSR       _CreatePort         ;Crea il port
                MOVE.L    D0,MsgPort          ;Conserva il puntatore in MsgPort
                ADDQ.L    #8,SP               ;Aggiusta lo stack
                MOVE.L    D0,-(SP)            ;Indirizzo del port
                JSR       _CreateStdIO        ;Crea una IOStdReq
                MOVE.L    D0,IOStdReq         ;Conserva il puntatore in IOStdReq
                ADDQ.L    #4,SP               ;Aggiusta lo stack
                LEA       DevName,A0          ;Nome del device (input.device)
                MOVE.L    IOStdReq,A1         ;Indirizzo della IOStdReq
                MOVEQ     #0,D0               ;Flags = 0
                MOVEQ     #0,D1               ;Unit = 0
                MOVE.L    _SysBase,A6         ;Routine di Exec
                JSR       _LVOOpenDevice(A6)  ;Apre il device

* Inizializzazione del loop di load
* degli  eventi. Viene caricato  il
* 'tempo'  del primo  evento e con-
* servato in luogo ... fresco  e a-
* sciutto ...

                MOVE.L    DOSBase,A6          ;Preleva DOSBase
                MOVE.L    FileHandle,D1       ;Si: preleva FileHandle
                LEA       Time_Secs,A0        ;Preleva l'indirizzo di TimeStamp
                MOVE.L    A0,D2               ;D2 = Buffer Address
                MOVEQ     #8,D3               ;Due long words
                JSR       _LVORead(A6)        ;Legge il TimeStamp

* Loop di  loading degli eventi dal
* file.  Legge gli eventi e li  ri-
* propone fino alla fine del file.

Loop            MOVE.L    DOSBase,A6          ;Preleva DOSBase
                MOVE.L    FileHandle,D1       ;FileHandle in D1
                LEA       InputEvent,A0       ;Indirizzo di InputEvent
                ADDQ.L    #4,A0               ;Salta ie_NextEvent
                MOVE.L    A0,D2               ;D2 = Input Buffer
                MOVEQ     #18,D3              ;Lunghezza di InputEvent - 4
                JSR       _LVORead(A6)        ;Legge l'evento
                CMPI.L    #0,D0               ;Testa D0 (ritornato da Read())
                BLE.S     Ending              ;D0 <= 0 (errore/fine file)
                LEA       InputEvent,A0       ;Indirizzo di InputEvent
                MOVE.L    IOStdReq,A1         ;Indirizzo di IOStdReq
                MOVE.L    A0,io_Data(A1)      ;Ind. di InputEvent in io_Data
                MOVE.W    #ID_WRITEVENT,io_Command(A1) ;Comando di 'stampa' evento
                MOVE.L    ie_Time_Secs,D0     ;Secondi dell'evento caricato
                SUB.L     Time_Secs,D0        ;Secondi dallo scorso evento
                MULU      #50,D0              ;In cinquantesimi di secondo
                MOVE.L    ie_Time_mScs,D1     ;Microsecondi dell'evento caricato
                SUB.L     Time_Secs,D1        ;Microsecondi dallo scorso evento
                DIVS      #20000,D1           ;In cinquantesimi di secondo
                ANDI.L    #$0000FFFF,D1       ;Preleva solo il quoziente
                EXT.L     D1                  ;Estende il segno di D1
                ADD.L     D0,D1               ;Aggiunge i due risultati
                JSR       _LVODelay(A6)       ;Attende il passare del tempo
                MOVE.L    _SysBase,A6         ;Una routine di Exec
                JSR       _LVODoIO(A6)        ;'Scatena' l'evento
                MOVE.L    ie_Time_Secs,Time_Secs ;Nuovi valori per il tempo
                MOVE.L    ie_Time_mScs,Time_mScs
                BRA       Loop                ;Daccapo (fino alla fine del file)


* Fine  programma. Il programma  chiude
* tutte le  strutture,  libraries e de-
* vices  aperte  e restituisce il  con-
* trollo al DOS.

Ending          MOVE.B    #0,ie_Class         ;Mandiamo un NULL Event
                LEA       InputEvent,A0       ;Indirizzo di InputEvent
                MOVE.L    IOStdReq,A1         ;Indirizzo di IOStdReq
                MOVE.L    A0,io_Data(A1)      ;InputEvent in io_Data
                MOVE.W    #ID_WRITEVENT,io_Command(A1) ;Comando
                MOVE.L    _SysBase,A6         ;Routine di Exec
                JSR       _LVODoIO(A6)        ;Spedisce l'evento
                MOVE.L    DOSBase,A1          ;DOS Base Library Pointer
                JSR       _LVOCloseLibrary(A6) ;Chiude la dos.library
                MOVE.L    IOStdReq,A1         ;Indirizzo della nostra IOStdReq
                JSR       _LVOCloseDevice(A6) ;Chiude la input.device
                MOVE.L    MsgPort,-(SP)       ;MsgPort Address sullo stack
                JSR       _DeletePort         ;Chiude il port
                ADDQ.L    #4,SP               ;Aggiusta lo stack
                MOVE.L    IOStdReq,-(SP)      ;E adesso chiudiamo ...
                JSR       _DeleteStdIO        ;... la IOStdReq
                ADDQ.L    #4,SP               ;Riaggiusta lo stack
                MOVEQ     #0,D0               ;RETURN_OK
                RTS                           ;Buonanotte!

* Area dati

DOSBase         DC.L      0                   ;Spazio per DOS Library Pointer

MsgPort         DC.L      0                   ;Spazio per MsgPort Pointer
IOStdReq        DC.L      0                   ;Spazio per IOStdReq Pointer

FileHandle      DC.L      0                   ;Spazio per FileHandle Pointer

InputEvent:                                   ;Il nostro Input Event
ie_NextEvent    DC.L      0
ie_Class        DC.B      0
ie_SubClass     DC.B      0
ie_Code         DC.W      0
ie_Qualifier    DC.W      0
ie_X            DC.W      0
ie_Y            DC.W      0
ie_Time_Secs    DC.L      0
ie_Time_mScs    DC.L      0

Time_Secs       DC.L      0                   ;Area dati per il conto del tempo
Time_mScs       DC.L      0

DosName         DC.B      'dos.library',0     ;Il nome della dos.library
PortName        DC.B      'InputDevPort',0    ;Il nome del nostro port
DevName         DC.B      'input.device',0    ;Il nome della input.device

                CNOP      0,2

                END
```

Code	Valore	Significato
IECODE_UP_PREFIX	0x80	Prefisso per "tasto rilasciato"
IECODE_KEY_CODE_FIRST	0x00	Il minore valore corrispondente a un tasto
IECODE_KEY_CODE_LAST	0x77	Il maggior valore per un tasto
IECODE_LBUTTON	0x68	Left Button del mouse premuto
IECODE_RBUTTON	0x69	Right Button del mouse premuto
IECODE_MBUTTON	0x6A	Middle Button (non previsto)
IECODE_NOBUTTON	0xFF	Nessun bottone premuto
IECODE_NEWACTIVE	0x01	???
IECODE_REQSET	0x01	Requester Set (?)
IECODE_REQCLEAR	0x00	Requester Clear (?)

Figura 4

Il valore settato in is_Data al momento della sistemazione dell'InputHandler ci viene passato (come detto) in A1: tenete presente che potete variare a piacimento il valore di questi registri... a vostro rischio e pericolo! Al termine dell'InputHandler, comunque, dovrete passare, in D0, l'indirizzo di una struct InputEvent contenente l'evento opportunamente «cucinato» dal vostro InputHandler.

Infine, ie_TimeStamp è una struct timeval così organizzata:

```
struct timeval {
    ULONG   tv_secs;
    ULONG   tv_micro;
}
```

che riporta il numero di secondi e microsecondi trascorsi dall'accensione del computer (o dall'ultimo azzeramento).

Consigli di programmazione

Anzitutto, è bene cominciare l'InputHandler con una chiamata alla funzione Forbid() della exec.library, che inibisce il perpetramento (aiuto mamma!) di ulteriori Software Interrupt; poi li riabiliterete con una chiamata alla Permit() prima di uscire dall'Handler. Vediamo un esempio in figura 7 (vedi dopo per ulteriori chiarimenti).

Il S.O., in genere, non passa un solo evento alla volta, ma più di uno: i vari eventi sono collegati a «catena», rispettando l'ordine cronologico in cui si sono presentati. Così, il campo ie_NextEvent del primo evento punta al secondo, l'ie_NexEvent del secondo contiene l'indirizzo del terzo... e così via: potrete accorgervi della fine della «catena» quando troverete un ie_NextEvent contenente 0. Un InputHandler scritto bene dovrebbe tenere presente questa (seppur modesta) difficoltà.

Nel caso il vostro InputHandler necessiti di più dati, potete riunirli in una struttura e passare all'InputHandler, attraverso A1, l'indirizzo di questa. Un trucchetto del genere si è visto, ad esempio, nell'utility POPCLI II by The Software Distillery: in questo caso si passava all'InputHandler una struttura contenente vari importanti (vitali!) puntatori.

Nell'ambito dell'InputHandler sono disponibili (quasi) tutte le funzioni di libreria: tenete comunque presente che un InputHandler non è un processo, dunque non ha, in genere, un Input e un Output FileHandle (quelli, per intenderci, che è possibile settare da CLI all'atto della scrittura del comando con gli operatori di redirezione > e <). In genere, le funzioni di formatted print (printf(), fprintf(), sprintf() e chi più ne ha più ne metta) vanno dritte alla Guru Meditation...

Infine è bene NON scrivere InputHandler in C. Infatti, quasi tutti i compilatori disponibili per Amiga (Lattice 3.10 in primis, ma anche il 5.0!) creano un workspace sullo stack con l'istruzione LINK $FFF2,A5 — scoperta con un disassemblamento selvaggio —, cioè bloccando 65522 byte (!!) sullo stack e ponendone l'indirizzo in A5. Tutto va bene quando ci si trova nell'ambiente del programma, ma i «casini» succedono quando si va nell'InputHandler: infatti viene passato un A5 NON corrispondente a quello atteso dalla routine.

Programmazione multilinguaggio

Nell'ambito di un programma in qualunque linguaggio (in Assembly, C e Pascal sicuramente) è possibile chiamare una funzione «esterna», cioè non definita all'interno del programma stesso; in C una funzione (o un dato) si dichiara «esterna» con la Keyword extern:

```
extern struct MsgPort *CreatePort();
```

In Assembly si fa riferimento a qualcosa di esterno (più precisamente, a una label esterna) con la direttiva XREF (Cross Reference):

```
XREF        _LVOPermit
XREF        _fprintf
```

rispettivamente dichiarano esterni il riferimento alla routine Permit (LVO sta per Library Value Offset) e la funzione fprintf.

I compilatori prepongono, ai nomi delle loro funzioni, un« _ »: notate, ad

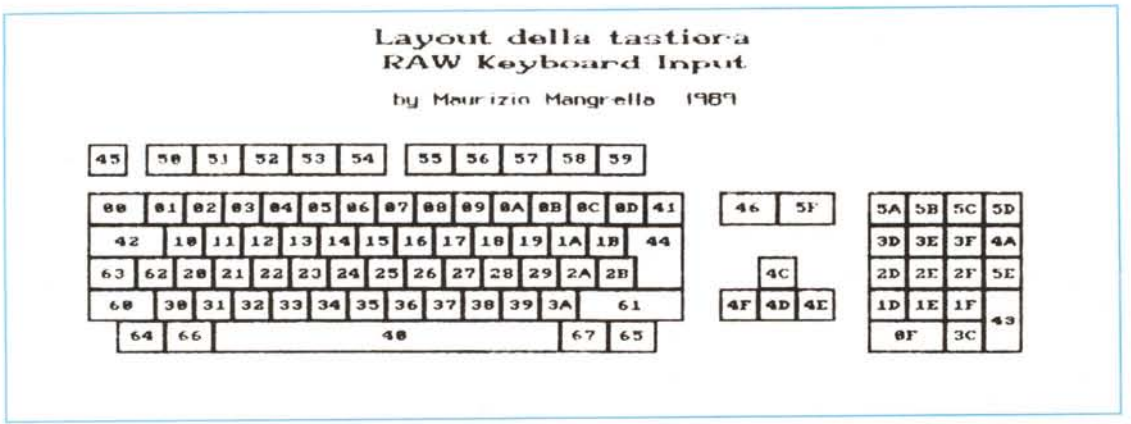


Figura 5 - Codici esadecimali restituiti dalla input.device nel field code di un InputEvent.

```
Qualifier                   Valore    Funzione

IEQUALIFIER_LSHIFT          0x0001    Left Shift premuto
IEQUALIFIER_RSHIFT          0x0002    Right Shift premuto
IEQUALIFIER_CAPSLOCK        0x0004    Caps Lock premuto
IEQUALIFIER_CONTROL         0x0008    Control premuto
IEQUALIFIER_LALT            0x0010    Left Alt premuto
IEQUALIFIER_RALT            0x0020    Right Alt premuto
IEQUALIFIER_LCOMMAND        0x0040    Left Amiga (Commodore) premuto
IEQUALIFIER_RCOMMAND        0x0080    Right Amiga premuto
IEQUALIFIER_NUMERICPAD      0x0100    Tasto del tastierino numerico
IEQUALIFIER_REPEAT          0x0200    Ripetizione di un tasto
IEQUALIFIER_INTERRUPT       0x0400    Software Interrupt (non usato)
IEQUALIFIER_MULTIBROADCAST  0x0800    Evento inviato a piu' finestre
IEQUALIFIER_MIDBUTTON       0x1000    Left Button del mouse
IEQUALIFIER_RBUTTON         0x2000    Right Button del mouse
IEQUALIFIER_LEFTBUTTON      0x4000    Mid Button (non previsto)
IEQUALIFIER_RELATIVEMOUSE   0x8000    Coordinate del mouse relative
```

Figura 6

```
_SysBase        EQU     4               ;Riferimento ad Exec
                XREF    _LVOForbid
                XREF    _LVOPermit
                .
                .
MyHandler       MOVE.L  _SysBase,A6
                JSR     _LVOForbid(A6)
                <resto dell'Handler>
                MOVE.L  _SysBase,A6
                JSR     _LVOPermit(A6)
                RTS
```

Figura 7

```
extern void HandlerInterface();
  .
  .
handlerStuff.is_Code = HandlerInterface;
  .
  .
struct InputEvent *myhandler(ev,mydata) ( etc., etc. ... )


                XREF    _myhandler
                XDEF    _HandlerInterface

_HandlerInterface:
                MOVEM.L A0/A1,-(SP)
                JSR     _myhandler
                ADDQ.L  #8,SP
                RTS
```

Figura 8

Il compito fondamentale di un linker è quello di risolvere tutti i riferimenti esterni. Avete un minuto per indovinare cosa succede in quei due programmi, una volta linkati insieme: (pausa). Non avete capito? Vabbé, non fa niente: ve lo spiego io. Il programma in C chiama la funzione HandlerInterface (ricordate il « _ » preposto): questa label è resa disponibile all'esterno con la direttiva XDEF (Cross Definition). Dunque il controllo passa alla HandlerInterface, che, dopo un po' di operazioni (che chiariremo in seguito) chiama la _myhandler, definita esterna con la XREF. La _myhandler corrisponde alla funzione myhandler() del listato C: dunque, come avrete capito (spero! Sono io che credo di non aver capito niente!) Il C chiama l'Assembly, che, a sua volta, chima il C di nuovo. Bello, no? Elegante e funzionale. Di questi trucchetti faremo maggior uso in seguito.

Per assemblare e linkare un programma (ad esempio Prog.asm) i comandi sono:

```
ASSEM Prog.asm -o Prog.o
ALINK FROM Prog.o TO Prog LIB lib:amiga.lib
```

Volendo compilare un programma che fa riferimento a un oggetto in Assembler esterno con il Lattice C 3.10 la sintassi è la seguente (posto che il sorgente C si chiami Prog_C.c e quelli in Assembly Prog_ASM.asm):

```
LC1 Prog_C
LC2 Prog_C
ASSEM Prog_ASM.asm -o Prog_ASM.o
BLINK FROM lib:c.o+Prog_C.o+Prog_ASM.o TO Prog LIB lib:lc.lib+lib:amiga.lib.
```

Ora, per coloro che hanno almeno qualche rudimento sul funzionamento del 68000, spiegherò quelle che vanno sotto il nome di «C Calling Conventions»: come si chiama una funzione esempio, la fprintf. Ogni label definita in un programma C (nome di variabile, indirizzo, funzione) è SEMPRE disponibile dall'esterno, cioè perché un altro programma possa farvi riferimento. In figura 8 uno stralcio di un programma che molto mi ha fatto penare (alla fine vi spiegherò perché).

Veramente i programmi sono due, uno in C e uno in Assembly: dopo la compilazione separata di entrambi, bisogna «linkarli» (legarli insieme, unirli) con il linker ALINK di mamma Commodore (o BLINK della The Software Distillery).

```
                XREF    _CreatePort
                  .
                MOVE.L  #0,-(SP)
                PEA     PortName
                JSR     _CreatePort
                ADDQ.L  #8,SP
                MOVE.L  D0,PortAddress

PortAddress     DC.L    0               ;indirizzo del Port
PortName        DC.B    'MyPort',0      ;stringa terminata con un CHR$(0)
```

Figura 9

Controller	Valore	Tipo
GPCT_NOCONTROLLER	0	Nessun controller (blocca tutto!)
GPCT_MOUSE	1	Mouse (default)
GPCT_RELJOYSTICK	2	Joystick relativo
GPCT_ABSJOYSTICK	3	Joystick assoluto

Figura 10

Figura 11

```
dat = GPCT_RELJOYSTICK;     /* Tanto per fare un esempio */
myreq->io_Data = (APTR)&dat;
myreq->io_Length = 1;
myreq->io_Command = IND_SETMTYPE;
DoIO(myreq);
```

```
struct GamePortTrigger mytrigger;
  .
myreq->io_Data = (APTR)&mytrigger;
myreq->io_Length = sizeof(struct GamePortTrigger);
myreq->io_Command = IND_SETMTRIG;
DoIO(myreq);
```

Figura 12

prevista per un listato C da un programma in Assembly. Anzitutto, gli argomenti vengono passati sullo stack, sotto forma di dati a 32 bit depositati su di esso in ordine inverso alla lettura (cioè dall'ultimo al primo); dunque, si chiama la funzione. Un valore eventualmente ritornato dalla funzione viene da questa passato in D0. Mi spiego con il solito esempio (vedi figura 9).

Con la MOVE.L #0, -(SP) trasferiamo sullo stack il secondo argomento (i flag del port), mentre, con la PEA portName, trasferiamo sullo stack (sempre là!) l'indirizzo effettivo del nome del port (l'istruzione PEA [Push Effective Address], analogamente alla LEA, forma un indirizzo e lo deposita sullo stack). Infine, chiamiamo la CreatePort e depositiamo l'indirizzo del port in PortAddress (una locazione dell'area dati del programma). Ecco perché, nella _HandlerInterface di cui sopra, il D0 non viene memorizzato: esso vale come ritorno per l'InputHandler.

Noterete anche che, dopo la chiamata a CreatePort, abbiamo «aggiustato» lo stack: questo perché una routine in C non preleva i dati dallo stack in modo usuale, ma «delicatamente», senza variare lo Stack Pointer (SP).

Conseguentemente, alla fine bisogna aggiungere tante volte 4 quanti argomenti abbiamo passato (nel caso di cui sopra 2*4 = 8).

Gli altri comandi

Quasi dimenticavo gli altri comandi... Dunque, potete cambiare driver per il vostro mouse. Anzitutto, per cambiare porta (giochi), potete dare questo:

```
UBYTE dat;
  .
  .
dat = 1;  /* Porta 2 (sinistra sul 500) */
myreq->io_Data = (APTR)&dat;
myreq->io_Length = 1;
myreq->io_Command = IND_SETMPORT;
DoIO(myreq);
```

Oppure potete cambiare controller: se date uno sguardo al file INCLUDE <devices/gameport.h>, troverete i tipi di controller rappresentati in figura 10.

Il joystick assoluto registra solo la direzione della leva (senza far muovere il pointer, se non di un solo pixel [!]), mentre il joystick relativo sposta effettivamente il cursore nella direzione in cui muoviamo la leva. Al solito, la procedura è rappresentata in figura 11.

Ma le possibilità di driving non terminano qui: potete decidere quali eventi prendere in considerazione e in quale modo: a questo scopo potete settare il Trigger, il quale è siffatto:

```
struct GamePortTrigger {
   UWORD gpt_Keys;
   UWORD gpt_Timeout;
   UWORD gpt_XDelta;
   UWORD gpt_YDelta;
}
```

gpt_Keys specifica la sensibilità ai tasti: specificando solo 1 (GPD_DOWNKEYS), verrà riportata la sola pressione dei microswitch; se aggiungiamo 2 (GPD_UPKEYS) verrà riportato anche il loro rilascio; ovviamente, settando solo GPD_UPKEYS riceveremo solo informazioni sul rilascio dei microswitch. gpt_Timeout è il numero di Ticks (cinquantesimi di secondo) che devono trascorrere tra un evento e l'altro perché siano considerati diversi. Infine, gpt_XDelta e gpt_YDelta non so proprio cosa siano, anche perché sembra non servano proprio a niente... Ciò nonostante, è bene settarli entrambi a 1.

La procedura è (sic!) sempre uguale (vedi figura 12).

Possiamo spedire un evento (che passerà regolarmente la catena degli Handler) con il comando IND_WRITEEVENT:

```
struct InputEvent myevent;
  .
  .
myreq->io_Data = (APTR)&myevent;
myreq->io_Length = sizeof(struct InputEvent);
myreq->io_Command = IND_WRITEEVENT;
DoIO(myreq);
```

Infine, ci togliamo dalle scatole (pardon!) l'InputHandler sistemato:

```
struct Interrupt HandlerInt;
  .
  .
myreq->io_Data = (APTR)&HandlerInt;
myreq->io_Command = IND_REMHANDLER;
DoIO(myreq);
```

Conclusioni

Il «programma che molto mi aveva fatto penare» è un demo della input.device scritto da Rob Peck della Commodore-Amiga Inc., che si limitava a riportare dati sugli eventi in corso. La «pena» derivava dall'istruzione LINK (solo troppo tardi scoperta), i cui malefici effetti si fecero sentire per circa un mese... fino alla stoica decisione: «Lo riscrivo in Assembly, tie'!!».

In conclusione, vi presento due programmi in Assembly regolarmente commentati che, rispettivamente, salvano e riprendono su/da un file tutto quello che fate con la tastiera e/o col mouse: il primo ha la sintassi

SaveEvents > (nome del file)

mentre il secondo si lancia con

LoadEvents < (nome del file)

Il programma SaveEvents salva prima il timeval del primo evento (che va dunque perso), poi gli altri eventi (18 byte per evento).

Per esigenze... didattiche (aridaje!) ho tralasciato alcuni particolari necessari in un programma curato (se ne vedono molti di questo genere in giro) l'importante è capirsi.

Tra i principali difetti annovero il salvare il solo primo evento di una catena, il non considerare la posizione iniziale del mouse e il non bloccare gli eventi per proporre quelli registrati, insieme a tante altre cosucce che sarebbero state utili... e che non ci sono.

Comunque il palinsesto è valido e le modifiche non sarebbero difficili: in effetti il mio unico scopo è quello di dimostrare l'uso del device, tutto qui.

Consiglio di salvare gli eventi in RAM: per renderne più veloce la gestione in fase di caricamento. Particolarmente interessante (se non altro per la quantità di tempo speso a farla funzionare) è, secondo me, la parte del LoadEvents che calcola il tempo di attesa tra un evento e l'altro: solo sette istruzioni (potenza del 68000!).

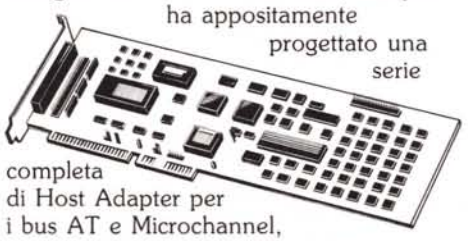
Adaptec: le nuove prestazioni di una multiutenza intelligente

Da Contradata le soluzioni multiuser per il bus AT e Microchannel

La multiutenza veloce è quella intelligente

In condizioni di multiutenza con gli attuali AT 286/386, spesso accade che il traffico dati tra CPU e periferia sia molto congestionato. In tal caso, perfino i sistemi operativi più evoluti, come XENIX e UNIX, non riescono ad esprimere tutta la loro potenziale velocità.

Per eliminare questo classico "collo di bottiglia" tra CPU e Hard Disk, Adaptec ha appositamente progettato una serie



completa di Host Adapter per i bus AT e Microchannel, nati per supportare HDD e FDD con protocollo SCSI in ambiente multiuser.

La loro capacità di gestire più comandi contemporaneamente, consente di liberare, con tecnica Mailbox in DMA, la CPU dalla gestione di tutte le operazioni di Input/Output, assegnando all'Host Adapter il compito di smaltire il trasferimento dati alle periferiche.

In ambiente UNIX e XENIX, la velocità e l'efficacia del sistema crescono in modo considerevole se confrontate con qualsiasi soluzione ESDI.

Adattabilità immediata a tutti i sistemi operativi multiutenza

I principali standard di multiutenza, tra cui:
- SCO XENIX 2.3 GT (286) per AT
- SCO XENIX 2.4 (386) per AT
- SCO XENIX 286 PS/386 PS per Microchannel
- ISC UNIX (386/IX release 2.0 e sup.)
- MICROPORT UNIX 5,

supportano in modo nativo gli Host Adapter Adaptec.

Multiutenza più efficiente anche per NOVELL

Adaptec ha realizzato un driver software per NOVELL 2.12 (escluso ELS2) e 2.15 per bus AT e Microchannel: da oggi anche questo standard così evoluto potrà godere delle grandi prestazioni assicurate dalla perfetta integrazione HW/SW firmata Adaptec.

Per ulteriori informazioni sui prodotti distribuiti da Contradata, telefonate allo 039/737015 o scrivete a Contradata srl, Via Monte Bianco, 4 - 20052 Monza (MI) telex 352830 CONTRA I - fax 039/735276 G3.

guétal

Present SMAU '89

contradata

PER COMPUTER CHE NON HANNO TEMPO DA PERDERE