a cura di Andrea de Prisco

Questo mese due programmi di due abili lettori. Il primo, scritto nientepopodimeno che in Modula 2, permette di aggiungere un simpatico (poi capirete perché) click alla tastiera del vostro Amiga. Purtroppo il listato che pubblichiamo ha solo valore didattico (chissà che non vi venga la voglia di provare a lavorare con questo interessantissimo linguaggio) in quanto per funzionare ha bisogno di una apposita libreria che, naturalmente, non possiamo pubblicare. Se poi morite dalla voglia di far «suonare» i vostri tasti, potete sempre richiedere il dischetto in redazione con libreria e file eseguibile.
Il secondo programma è un vero e proprio suicidio. Beh, non esageriamo: serve semplicemente per togliere lock al sistema operativo. Assolutamente da non utilizzare per gioco (raffiche di guru a più non posso) ma in alcuni casi può salvarci da situazioni sgradevoli tipo non riuscire a salvare un file perché un altro processo l'ha lockato senza sbloccarlo dopo l'uso. Un consiglio: dopo aver utilizzato tale comando, salvate il salvabile e rieseguite il boot della macchina, non si sa mai...

Click

di Giovanni Christen - Zurigo

Il programma «Click» è un'appendice all'articolo del lettore Maurizio Mangrella, pubblicato sul numero di ottobre di MC e dedicato alla Input Device. Non aggiungo nulla alla esauriente trattazione di Mangrella, ma penso sia di interesse generale un esempio scritto in un linguaggio superiore (nel mio caso Modula 2).

Inoltre, non avendo ancora visto programmi scritti in Modula 2 apparire sulle pagine di MC, vorrei mostrare che, almeno nel caso di Amiga, si tratta di un'ottima alternativa al C. Nonostante il parere di alcuni, Modula 2 è un linguaggio estremamente flessibile, proponendosi spesso quale alternativa allo stesso Assembler. Nel caso particolare della realizzazione di un Handler, secondo Mangrella, si deve rinunciare al C. Ma non si deve, aggiungo io, rinunciare al Modula 2!

Veniamo quindi al programma. Si tratta della generazione di un suono ad ogni pressione di tasto (normalmente la tastiera di Amiga è «muta»). Il programma gira in background, intercetta i segnali

```
MODULE Click;
(*-------------------------------------------------------------------------

     C L I C K            (c) Giovanni Christen 18.10.1989
    ============

  aggiunge 'click' alla pressione dei tasti

  cerca di caricare i files (suoni digitalizzati IFF) seguenti:

  click.sn1 : battuta (carattere)
  click.sn2 : spazio
  click.sn3 : enter
  click.sn4 : shift - Caps Lock

  Non trova click.sn1  : produce un breve beep per ogni tasto premuto

  -------------------------------------------------------------------------*)

FROM SYSTEM IMPORT ADR, ADDRESS, BYTE, TSIZE, REG;
FROM Ports IMPORT GetMsg, ReplyMsg, MsgPortPtr;
FROM PortsUtil IMPORT CreatePort, DeletePort;
FROM IODevices IMPORT IOStdReqPtr, DoIO, OpenDevice, CloseDevice;
FROM IODevicesUtil IMPORT CreateStdIO, DeleteStdIO;
FROM Interrupts IMPORT Interrupt, InterruptPtr, Forbid, Permit;
FROM Intuition IMPORT Window, WindowPtr, IDCMPFlags, IDCMPFlagsSet, ModifyIDCMP,
                      IntuiMessagePtr, CloseWindow, WindowFlags, WindowFlagsSet;
FROM InputEvents IMPORT InputEvent, InputEventPtr, IEQualifier, IEClass;
FROM InputDevice IMPORT INDAddHandler, INDRemHandler;
FROM SimpleWindows IMPORT CreateWindow;
FROM AmigaDOSProcess IMPORT Delay;
FROM AudioTools IMPORT BeginAudio, GetChannel, FinishAudio, NOTA, Suona,
                       SAMPLE, LoadSample, FreeSample, PlaySample, loadOK;

PROCEDURE InputHandler(): ADDRESS;
VAR data: POINTER TO CARDINAL; ev: InputEventPtr;
BEGIN
  ev := ADDRESS(REG(8));      (* get A0 - first event in list       *)
  data := ADDRESS(REG(9));    (* get A1 - user defined data pointer *)
  Forbid;
  WITH ev^ DO
   IF (ieClass=IEClassRawKey) AND (ieCode<128) THEN data^:=ieCode END END;
  Permit;
  RETURN ev
END InputHandler;
```

```
VAR win     : WindowPtr;
    msg     : IntuiMessagePtr;
    channel : INTEGER;
    beep    : BOOLEAN;
    dummy   : LONGINT;
    code    : POINTER TO CARDINAL;
    myPort  : MsgPortPtr;
    handlerReq   : IOStdReqPtr;
    handlerStuff : Interrupt;
    sn0,sn1,sn2,sn3,sn4: SAMPLE;

BEGIN
  BeginAudio;
  channel := GetChannel(3);
  sn1 := LoadSample('click.s1');  beep := NOT(loadOK);
  IF loadOK THEN sn2:=LoadSample('click.s2'); sn3:=LoadSample('click.s3');
                 sn4:=LoadSample('click.s4') END;

win := CreateWindow(360,0,150,10,IDCMPFlagsSet{Closewindow},
                    WindowFlagsSet{WindowDrag,WindowClose,WindowDepth},
                    NIL,NIL,ADR('Click'));
IF win<>NIL THEN
  myPort := CreatePort(ADR('ClickPort'),0);
  handlerReq := CreateStdIO(myPort^);

  IF OpenDevice(ADR('input.device'),0D,handlerReq,0D)=0D THEN
    WITH handlerStuff DO
      isNode.lnPri := BYTE(51);
      isData       := code;
      isCode       := ADR(InputHandler)
      END;
    WITH handlerReq^ DO
      ioCommand := INDAddHandler;
      ioLength  := TSIZE(Interrupt);
      ioData    := ADR(handlerStuff)
      END;

    IF DoIO(handlerReq)=0D THEN
      code^:=0D;
      REPEAT
        Delay(5D);                              (* controlla 10 volte per sec. *)
        IF code^>0 THEN
          IF beep THEN Suona(channel,la,3,64,44)     (* suona un breve beep *)
             ELSE IF ((code^>0) AND (code^<64)) OR  (* oppure un sample !  *)
                     ((code^>89) AND (code^<95)) OR (code^=74) THEN sn0:=sn1
                   ELSIF (code^=64) THEN sn0:=sn2                 (* space *)
                   ELSIF (code^=67) OR (code^=68) THEN sn0:=sn3  (* enter *)
                   ELSIF (code^>95) AND (code^<99) THEN sn0:=sn4 (* shift *)
                   ELSE code^:=0 END;                             (* tasto muto *)
                IF code^>0 THEN PlaySample(channel,sn0,1,64) END;
          END;
        code^:=0 END;
        msg := GetMsg(win^.UserPort^)      (* cerca messaggi IDCMP per win^ *)
        UNTIL msg<>NIL;          (* unico messaggio possibile: CloseWindow *)
      ReplyMsg(msg);

      handlerReq^.ioCommand := INDRemHandler;
      dummy:=DoIO(handlerReq)
      END;

    CloseDevice(handlerReq);
    DeleteStdIO(handlerReq);
    DeletePort(myPort^);

    END;

  CloseWindow(win^);
  END;

FreeSample(sn1); IF NOT(beep) THEN FreeSample(sn2); FreeSample(sn3);
                                   FreeSample(sn4) END;
FinishAudio;

END Click.
```

dalla tastiera e produce (eventualmente) un suono. In particolare, se il programma trova un file di nome click.S1 contenente un suono digitalizzato, lo utilizza per i tasti «che scrivono». Se poi trova anche i file click.S2, click.S3, click.S4 li utilizza rispettivamente per spazio, return, shift. Ad esempio è possibile trasformare la tastiera di Amiga in una Olivetti meccanica... È naturalmente possibile utilizzare altri suoni, all'unica condizione che rispettino lo standard IFF. Consiglio comunque di utilizzare solo suoni brevissimi, a meno di non avere una battuta molto lenta. Se il programma non trova suoni digitalizzati, produrrà comunque un breve suono, in questo caso alla pressione di ogni tasto.

Il programma può venire lanciato da Workbench o da CLI (in questo caso si consiglia di usare «RUN click»). Apre una finestra con gadget di chiusura sulla barra dei Menu. Utilizza un solo canale audio (se lo trova libero, sceglie il numero 3), rimanendo compatibile con programmi che producono suoni mediante l'audio.device. A conferma della capacità di multitasking ho preparato i suoni di esempio usando PerfectSound, controllandoli contemporaneamente con Click. Possono sorgere problemi con programmi che utilizzano direttamente i canali audio DMA. In genere si tratta di giochi che non ammettono il multitasking, oltre ad alcuni programmi musicali poco «AMIGhevoli».

Passiamo ora ad una descrizione interna del programma. Dopo aver inizializzato il modulo con le utility audio ed avere occupato un canale audio libero, sono caricati in Chip-RAM i suoni digitalizzati (se non sono disponibili la variabile logica «beep» assume valore True). Quindi viene aperta la finestra di controllo e installato l'Handler (procedura InputHandler). La preparazione dell'Handler segue il procedimento già descritto da Mangrella. Da notare il valore di handlerStuff.isData (indirizzo zona dati

È disponibile, presso la redazione, il disco con il primo programma pubblicato in questa rubrica. Le istruzioni per l'acquisto e l'elenco degli altri programmi disponibili sono a pag. 263.

per l'Handler): viene direttamente passata una variabile POINTER TO CARDINAL, ossia l'indirizzo in memoria di un numero intero positivo.

Il funzionamento dell'Handler è presto descritto: controlla se l'InputEvent proviene da tastiera e se corrisponde alla pressione di un tasto (code<128). Se così è, copia il valore di code all'indirizzo «data» (corrispondente alla variabile «code» nel programma principale).

Il passaggio di parametri allo Handler attraverso i registri A1 e A2 del 68.000 non segue ovviamente il formato delle procedure compilate (che utilizzano lo stack). Il contenuto dei due registri viene quindi letto mediante la procedura REG(). L'uso del registo DO per il risultato invece non pone problemi.

Il programma principale si «sveglia» 10 volte al secondo, per controllare se l'Handler ha modificato la variabile «code», e in caso positivo produce il suono. Quindi controlla se nel Port creato da Intuition per la finestra di controllo è apparso un messaggio: se così fosse, questo non potrà essere che un messaggio della classe WindowClose (è l'unica attiva!). Quindi procederà alla risposta (non è necessario controllare se ci sono altri messaggi collegati: non è umanamente possibile selezionare il gadget di chiusura 2 volte in meno di 1/10"), e alla chiusura di tutto ciò che è stato aperto.

Il programma è stato compilato con benchmark-Modula 2. La libreria AudioTools è uno sviluppo personale, che non allego per ragioni di spazio. Volendo utilizzare il programma si dovrà quindi usare la versione precompilata, o scrivere una propria routine per la generazione del suono. In questo ultimo caso sarà possibile scrivere una versione più compatta, soprattutto se si vorrà ricorrere al metodo «sporco» (ma tanto «smanettone») del controllo diretto DMA. Infatti il modulo AudioTools si indirizza a sonorizzazioni complesse, e inizializza automaticamente la memoria per 4 strumenti (comprende un player per formato SMUS, un player per suoni digitalizzati in doppio buffer, ecc...). Io consiglio comunque di richiedere il dischetto con il programma precompilato, che contiene anche 11-combinazioni-11 di suoni digitalizzati (Olivetti manuale, elettrica, popcorn, pollaio, flipper, ecc. ecc.).

Sblocca.C

di Giuseppe Ghibò - Rivoli (TO)

Il programma anche se è piuttosto breve è da considerare molto utile. Il suo effetto è di rimuovere un LOCK associato ad un file. Per chi non avesse voglia di andare a consultare la rubrica «Programmare in C su Amiga» di MC numero 75 e 76 dove si spiega appunto cos'è un LOCK legga le seguenti note. Un lock, lucchetto appunto, è uno strumento indispensabile in un ambiente multitasking, poiché comunica al sistema che un determinato file è accessibile (es. in scrittura) solamente all'utente che ne ha richiesto l'accesso esclusivo. Sostanzialmente il sistema assegna automaticamente un lock ad un file ogni qualvolta lo si apre (Open) e l'accesso a quel file rimane negato ad altri task finché il programma che lo aveva ottenuto non lo chiude (Close od UnLock). Spesso capita che alcuni programmi, terminano brutalmente lasciando dei file aperti; ora questi file non sono né cancellabili né rinominabili, a meno che non si effettui un nuovo boot del sistema; questo perché il task del programma è terminato ed ha lasciato aperto il lock associato al file. Il programma che vi presento invece permette appunto di rimuovere il lock associato al file rimasto aperto e quindi di fare poi quello che si desidera con quel file (delete, rename). Nota: è sempre possibile ottenere un lock con accesso di sola lettura di un file a lock esclusivo (scrittura); è proprio su quest'ultima possibilità che si basa il programma che ora andiamo ad analizzare.

1) Si ottiene anzitutto il nome del file da argv [1] e si richiede il lock in lettura per questo file; quello che si ottiene è un puntatore alla struttura FileLock qui schematizzata:

```
struct FileLock {
    BPTR              fl_Link;
    LONG              fl_Key;
    LONG              fl_Access;
    struct MsgPort *  fl_Task;
    BPTR              fl_Volume;
};
```

La funzione Lock() restituisce un puntatore alla struttura FileLock; il fatto, in un certo senso sgradevole, è che questo puntatore è di tipo BCPL (in pratica differisce dai puntatori C per il fatto che è shiftato a destra di 2 bit, ovvero è diviso per 4). Se si desidera quindi accedere ai membri della struttura a cui il puntatore punta tramite l'operatore «→», occorre convertire il puntatore BCPL in puntatore C, effettuando uno shift a sinistra di 2 bit. Nota: occorre ricordarsi che il puntatore ottenuto come lock deve essere riconvertito in BCPL prima di passarlo alle funzioni AmigaDOS (es. UnLock, Examine, etc); a tal fine basterà effettuare una shift a destra di 2 bit (>>2); di tali conversioni si occupano le due macro BPTR_TO_C e C_TO_BPTR.

I membri della struttura FileLock sono cinque. Il primo, *fl_Link*, è un puntatore BCPL ad una struttura FileLock contenente informazioni di un altro lock aperto nello stesso volume del precedente. Seguendo, quindi, tutti questi puntatori fino a quando non se ne trova uno NULL (il tappo) si possono ottenere tutti i lock ancora aperti in un volume; il secondo, *fl_Key*, contiene il numero del blocco (File Header Block o User Directory Block) del volume nel quale risiedono le informazioni del file associato al lock (nome, data, etc). l'elemento *fl_Access*, contiene invece il tipo di accesso del lock (esclusivo o condiviso); l'elemento *fl_Task*, contiene un puntatore alla struttura MsgPort per il task corrispondente; infine l'elemento *fl_Volume*, rappresenta un puntatore BCPL alla struttura DeviceList, la quale contiene le informazioni del volume nel quale si trova il file indicato dal lock.

2) Il lock ottenuto non punterà alla stessa locazione per la struttura FileLock del primo lock di quel file, bensì ad un'altra; malgrado ciò, i due puntatori avranno uguali gli elementi *fl_Key* (lock di file uguali hanno uguali il blocco di informazioni).

3) Il programma recupera, tramite l'elemento *fl_Link* della struttura FileLock, tutti i lock aperti dello stesso volume nel quale si trova il lock del file richiesto, fino ad un massimo di 100 e, dopo averli convertiti in puntatori C, ricerca tra questi quello che presenta lo stesso elemento *fl_Key* del lock iniziale (n.d.p.

```
/*****************************************************************************\
*                                                                             *
*                                 SBLOCCA                                     *
*                                                                             *
*                   compilare con lc -L -O sblocca.c                          *
*                                                                             *
\*****************************************************************************/

#include "exec/types.h"
#include "libraries/dosextens.h"
#include "stdio.h"
#include "proto/dos.h"

#define BPTR_TO_C(strt,var) ((struct strt *)(BADDR(var)))
#define C_TO_BPTR(strt,var) ((struct strt *)(((ULONG)var)>>2))

void main(argc,argv)
int argc;
char *argv[];
|
int i,k;            /* i = numero di lock "aperti" */
struct FileLock *lock[100],*closelock=NULL;

printf("Sblocca c v1.0 1989 by Ghibò Giuseppe\n");
if (argc != 2) |
   printf("Uso: Sblocca <file bloccato>\n");
   Exit(RETURN_OK);
   |
/* prende il lock del file da sbloccare */
if ((lock[0]=(struct FileLock *) Lock(argv[1],ACCESS_READ))==DOSFALSE) |
   printf("Errore: %ld\n",IoErr());
   Exit(RETURN_ERROR);
   |
lock[0]=BPTR_TO_C(FileLock,lock[0]);      /* lock BCPL -> lock Amiga C */

/* assume max 100 lock concatenati */
for (i=1;i<100;i++) |
   if ((lock[i]=BPTR_TO_C(FileLock,lock[i-1]->fl_Link)) == NULL) break;
   |

if (i == 1) |
   printf("Volume privo di lock\n");
   lock[0]=C_TO_BPTR(FileLock,lock[0]);
   UnLock((BPTR)lock[0]);
   Exit(RETURN_OK);
   |

for (k=1;k<i;k++) |
   if (lock[k]->fl_Key == lock[0]->fl_Key) |   /* il file e' questo */
      closelock=lock[k];
      break;
      |
   |

if (closelock == NULL) |
   printf("Il file %s è già libero\n",argv[1]);
   Exit(RETURN_OK);
   |

closelock=C_TO_BPTR(FileLock,closelock);
lock[0]=C_TO_BPTR(FileLock,lock[0]);
UnLock( (BPTR) closelock);  /* sblocco lock originale file */
UnLock( (BPTR) lock[0]);    /* sblocco lock copia */

printf("File: %s liberato\n",argv[1]);
|
```

questo piccolo trucco evita di scomodare altre strutture e funzioni per la ricerca del nome a partire dal blocco header). Nota: non è necessario consultare *fl_Volume* poiché quest'elemento è lo stesso per tutti i lock raggiungibili seguendo *fl_Link*.

4) Una volta trovato il lock cercato, si riconvertono i puntatori dei due lock associati allo stesso file in puntatori BCPL e quindi si effettua una chiamata alla funzione UnLock() che provvederà allo sblocco dei lock.

Come si può notare dal listato il programma è estremamente corto, tuttavia occorre usarlo con le dovute cautele; poiché esso è in grado di sbloccare qualunque lock di file o directory occorre fare *attenzione a non usarlo su file effettivamente in uso o sui device logici di sistema* (c:, devs:,l:,etc.) *poiché il sistema andrebbe in Guru non appena qualche task tentasse di chiudere o fare riferimento a lock inesistenti* (perché li abbiamo chiusi).

Nota: se il sistema vi darà ancora l'errore 202 (object in use) anche dopo aver usato il comando SBLOCCA è possibile che siano aperti più lock di uno stesso file; in tal caso usate il comando SBLOCCA finché non sarete avvertiti che il file è privo di lock.

Note tecniche di compilazione: il programma è stato compilato con il Lattice C 5.0. Esso fa uso degli include proto, nonché dei **#pragma** contenuti in questi include. Come si sa, gli include proto, contengono quello che in C viene chiamato function prototypes, ovvero tutte quelle direttive che indicano al compilatore il tipo e i parametri di una funzione. La direttiva **#pragma**, invece, come stabilito dal nuovo standard ANSI, permette di evitare la chiamata delle cosiddette **stub routine,** cioè di quei pezzi di codice contenuti nella libreria amiga.lib, che provvedono a portare i parametri passati ad una funzione dallo stack ai registri appropriati, prima di effettuare il salto alla funzione vera e propria. I vantaggi nell'uso di questa direttiva sono sostanzialmente due: il codice risulta tanto più veloce quante più funzioni si chiamano o quante più volte si chiama una sola funzione, inoltre si risparmia del tempo durante il linking (oltre ad una decina di byte per ciascuna funzione).

MC