di Sergio Polini

Programmi residenti

Fin dai tempi del glorioso SideKick, i programmi residenti hanno esercitato un fascino irresistibile su tutti noi: basta premere un tasto, o una particolare combinazione di tasti, per uscire temporaneamente dal programma che stiamo eseguendo, per consultare un calendario o fare calcoli o prendere appunti su un blocco elettronico, e poi tornare rapidamente a quello che stavamo facendo, senza che il programma interrotto ne risenta più di tanto. Ma il mistero è sempre stato almeno pari al fascino: mamma Microsoft ha tenuto a lungo nascosti molti aspetti del suo MS-DOS, ostinatamente e faticosamente ricercati da nugoli di irriducibili appassionati. Sono così stati svelati tanti segreti, ma in maniera spesso confusa e contraddittoria. Negli ultimi due anni, tuttavia, qualcosa è cambiato...

Quanta confusione! Sono stati pubblicati (negli USA) numerosi libri e articoli dedicati alla scrittura di programmi residenti, sia in Assembler che in linguaggi come il C e il Turbo Pascal. Purtroppo, a causa anche della mancanza di informazioni «ufficiali», ogni autore ha seguito una sua propria strada, spesso trascurando o sottovalutando aspetti fondamentali. Se questo poteva essere accettato fino a poco tempo fa, ormai non possiamo più ignorare che nel 1988 la Microsoft ha pubblicato la sua *MS-DOS Encyclopedia*, uno stupendo volume di oltre 1500 pagine ricco di utilissime informazioni, compreso un capitolo sui programmi residenti.

Chiunque si sia avvicinato al problema, sa che è estremamente arduo scrivere certi programmi senza avvalersi di caratteristiche «non documentate» del DOS (il flag InDOS e la funzione 34h dell'INT 21h, l'INT 28h, ecc.), ignorate o etichettata con un «reserved» perfino in altre esaurienti pubblicazioni della stessa Microsoft Press, quali l'eccellente *Advanced MS-DOS* di Ray Duncan. Bene. Queste benedette «caratteristiche non documentate» vengono finalmente illustrate nell'Enciclopedia; viene perfino proposto il sorgente in Assembler completo di un programma residente.

Come si può prescinderne? Come si può trascurare l'indicazione, ad esempio, che va intercettato l'INT 13h del BIOS piuttosto che gli INT 25h e 26h del DOS, come invece alcuni propongono? È vero, in quel sorgente non c'è tutto: si trascura l'eventuale coprocessore numerico, si trascura soprattutto la necessità di salvare e ripristinare l'apparenza del video. Non solo: non viene fornita alcuna indicazione sull'uso di certi meccanismi in linguaggi di alto livello (e meno che mai del Turbo Pascal...). Ma integrare ed adattare è ben diverso che ignorare.

Il nostro obiettivo sarà appunto di adattare ed integrare le informazioni contenute nella Enciclopedia, allo scopo di scrivere programmi residenti in Turbo Pascal 5.x usando quanto meno Assembler possibile. Procediamo per gradi.

Il programma residente più breve del mondo

Quando un normale programma DOS termina, la memoria che occupava viene liberata; ciò vuol dire che, quando ne eseguiamo un altro, istruzioni e dati di questo si sovrappongono a istruzioni e dati del primo, che diventano così inaccessibili. Tra i diversi modi in cui un programma può terminare ve ne sono però due che hanno un effetto diverso: se viene eseguito un INT 27h, oppure la funzione 31h dell'INT 21h, quella memoria non viene liberata; il controllo ritorna al DOS, ma ogni programma caricato successivamente verrà collocato in un'area di memoria più alta, senza ricoprire il programma rimasto «residente». Questo può quindi essere ancora usato.

Ma come si fa ad usare un programma diverso da quello che si sta eseguendo? La risposta non può che essere: mediante un interrupt. Quando viene eseguita una istruzione INT, il microprocessore salva nello stack i flag e i registri CS e IP, disabilita ulteriori interrupt, e prosegue con la routine il cui indirizzo è contenuto in una tabella posta nella parte più bassa della memoria (segmento zero, offset uguale al numero dell'interrupt moltiplicato quattro); esegue cioè l'equivalente di un PUSHF, CLI e CALL FAR. Quella routine terminerà poi con un IRET, analogo ad un POPF più RETF. Poiché l'IRET ripristina sia CS che IP, l'esecuzione può proseguire da dove era stata «interrotta». Rimane il problema di come ottenere questa interruzione.

Ne vediamo un primo esempio nel

```
CODE            SEGMENT
                ASSUME  CS:CODE,DS:CODE,ES:CODE
                ORG     100h

Start:          JMP     SHORT Installa

NuovoInt5       PROC    FAR
                IRET
NuovoInt5       ENDP

FineParteRes    LABEL   BYTE

Installa        PROC    NEAR
                MOV     AX,CS
                MOV     DS,AX
                MOV     DX,OFFSET NuovoInt5
                MOV     AX,2505h
                INT     21h
                MOV     ES,ES:[2Ch]
                MOV     AH,49h
                INT     21h
                LEA     DX,FineParteRes
                INT     27h
Installa        ENDP

CODE            ENDS
                END     Start
```

Figura 1 - Il sorgente di NOINT5.ASM, un brevissimo programma residente che non fa altro che «annullare» l'INT 5 (quello del Print Screen). Se assemblato con il TASM della Borland, si può poi usare TLINK con l'opzione «/t» per produrre un file COM; se invece si usano il MASM e il LINK della Microsoft, il file EXE prodotto va convertito in COM con EXE2BIN.

brevissimo programmino in figura 1. Ogni volta che premiamo il tasto *Print Screen*, viene inviata alla stampante una copia di quanto vediamo sul video; quel tasto infatti causa un INT 5, al quale è normalmente associata una routine che si occupa appunto di questo. Possiamo ottenere l'indirizzo di quella routine con la funzione 35h del DOS (segmento e offset in ES:BX), possiamo sostituirla con un'altra mettendone segmento e offset in DS:DX e chiamando la funzione 25h del DOS; in ambedue i casi il numero dell'interrupt va in AL. Il programma NOINT5 fa proprio questo, associando all'INT 5 una procedura che, contenendo solo un IRET, elimina gli effetti della pressione di quel tasto (magari utile per chi abbia una di quelle macchine che si impallano se la stampante non è collegata). Soprattutto il programma termina con un INT 27h: questo richiede in CS il segmento del *Program Segment Prefix* e in DX l'offset dell'ultimo byte del codice che deve rimanere residente; nel nostro caso, prima dell'INT 27h viene chiamata la funzione 49h del DOS per liberare la memoria occupata dall'environment (il cui segmento, ricordiamo, è contenuto all'offset 2Ch del PSP), col risultato di «lasciare residenti» solo 272 byte: i 256 del PSP più i 3 del primo JMP e dell'IRET, arrotondati a 16, cioè ad un paragrafo. I programmi eseguiti dopo NOINT5 verranno caricati in modo da non coprire qui 272 byte.

Nella figura 2 vediamo come si può ottenere lo stesso effetto in Turbo Pascal: la procedura *SetIntVec* si incarica di associare una nuova routine all'INT 5, la procedura *Keep* di lasciarla residente. L'occupazione di memoria che ne risulta è ovviamente superiore a quella ottenibile in Assembler, anche perché non c'è modo di lasciare residente solo una parte del programma: rimarranno in memoria, oltre al PSP e al nuovo INT 5, anche il codice, i dati, lo stack e l'heap. L'unica cosa che possiamo (e *dobbiamo*) fare è agire sulla direttiva M per contenere la misura dello stack e dello heap.

```
(*$M 1024,0,0*)
program NoInt5;
uses Dos;
var Reg: registers;
procedure NuovoInt5; interrupt;
  begin
  end;
begin
  SetIntVec(5, Addr(NuovoInt5));
  Reg.AH := $49;
  Reg.ES := MemW[PrefixSeg:$2C];
  MsDos(Reg);
  Keep(0)
end.
```

Figura 2 - La versione in Turbo Pascal del programma della figura 2.

Vi sono anche altre differenze. La procedura *Keep*, ad esempio, non usa l'INT 27h ma (più correttamente) la funzione 31h del DOS; questa consente tra l'altro di far terminare il programma con un codice a beneficio di un eventuale file batch, e di lasciare in memoria anche più di 64K. Più interessante la procedura *NuovoInt5*, per quello che «nasconde». Come spiega il manuale, dichiarando **interrupt** una procedura, questa viene poi compilata come indicato nella figura 3: all'inizio vengono salvati tutti i registri, si crea spazio nello stack per eventuali variabili locali, si assegna a DS il valore del segmento dati del programma (per consentire l'accesso alle sue variabili globali); alla fine si ripristinano tutti i registri e si dà un IRET. Otteniamo sì un IRET, ma abbiamo anche tutta una serie di operazioni sui registri che, se quasi sempre è preziosa, in alcuni casi può risultare poco comoda. Ma di questo riparleremo.

Estensioni del DOS

Gli interrupt si dividono in due grandi categorie: quelli hardware e quelli software. I primi vengono attivati da dispositivi fisici: l'INT 8 dall'orologio interno (ad un ritmo di circa 18,2 volte al secondo), l'INT 9 dalla tastiera. I secondi offrono invece accesso alle funzioni del BIOS e del DOS, e vengono quindi attivati da un programma, quasi come fossero normali subroutine. La differenza tra un software interrupt e una subroutine è soprattutto che per chiamare una subroutine dobbiamo conoscerne l'indirizzo, per attivare un software interrupt basta conoscerne il numero. Ne segue anche che, poiché possiamo associare la routine che vogliamo ad un dato interrupt, possiamo virtualmente riscrivere tutto il BIOS e tutto il DOS senza per questo dover intervenire sui programmi che fanno uso delle loro funzioni.

Non solo: alcuni interrupt (da 60h a 66h) sono lasciati a disposizione dell'utente, nel senso che, non essendo utilizzati dal sistema operativo, è possibile associare a loro routine che non interferiscono con il normale andamento delle cose. Si tratterebbe in questi casi di vere e proprie «subroutine universali», a disposizione cioè di tutti e soli i programmi che siano consapevoli della loro esistenza e ne conoscano il numero;

```
(* codice d'entrata *)
push    ax
push    bx
push    cx
push    dx
push    si
push    di
push    ds
push    es
push    bp
mov     bp,sp
sub     sp,SpazioPerVariabiliLocali
mov     ax,SEG DATA
mov     ds,ax

(* codice d'uscita *)
mov     sp,bp
pop     bp
pop     es
pop     ds
pop     di
pop     si
pop     dx
pop     cx
pop     bx
pop     ax
iret
```

Figura 3 - Il codice prodotto dal Turbo Pascal all'inizio e alla fine di una procedura dichiarata «interrupt».

```
(*$M 1024,0,0*)
program Int60;
uses
  Dos;
var
  Reg: registers;
  PrevInt60: procedure;
procedure NuovoInt60(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
interrupt;
begin
  Writeln('Software Interrupt 60h');
  Writeln('Valore del registro AX: ',AX);
  Writeln('Valore del registro BX: ',BX)
end;
begin
  GetIntVec($60, Addr(PrevInt60));
  if Addr(PrevInt60) = nil then begin
    SetIntVec($60, Addr(NuovoInt60));
    Reg.AH := $49;
    Reg.ES := MemW[PrefixSeg:$2C];
    MsDos(Reg);
    Keep(0)
  end
  else
    Writeln('INT 60h gia'' usato')
end.
```

Figura 4 - Un esempio di programma che installa un software interrupt, associando una routine ad uno degli interrupt «liberi» (quelli dal 60h al 66h).

con il duplice vantaggio che potrebbero essere «installate» (rese cioè residenti) una sola volta invece che ripetute in (cioè linkate a) tutti quei programmi, e che sarebbe possibile modificarle senza nemmeno toccare questi.

La comunicazione tra un programma e tali subroutine residenti avverrebbe proprio come con gli interrupt del BIOS e del DOS, attraverso i registri del processore. Non è di questo che ora vogliamo occuparci, ma un esempio può essere interessante: il programma INT-60.PAS (figura 4) installa un INT 60h, associando a questo una routine che si limita a visualizzare il contenuto dei registri AX e BX; possiamo verificarne il funzionamento con TEST60.PAS (figura 5), che chiama l'INT 60h dopo aver messo 1234 in AX e 5678 in BX.

Naturalmente in genere si usa questa tecnica per cose un po' più serie. Ad esempio si possono installare interrupt attraverso i quali gestire file di dati e relativi indici, un vero e proprio «Database Toolbox residente». Quello che ora va sottolineato è che, per quanto complicate possano essere le routine così installate, la loro attivazione è estremamente semplice: avviene come una normalissima chiamata di subroutine. Quelli che invece vengono comunemente chiamati «programmi residenti» richiedono un'attenzione nettamente maggiore.

```
Program Test60;
uses
  Dos;
var
  Reg: registers;
begin
  Reg.AX := 1234;
  Reg.BX := 5678;
  Intr($60,Reg)
end.
```

Figura 5 - Un semplice test dell'INT 60h come installato dal programma della figura 4.

Routine non rientranti

Le funzioni del DOS salvano prima e poi ripristinano i registri del microprocessore, in modo tale che il programma che le usa (tranne che in alcuni casi) non deve preoccuparsi di possibili alterazioni del suo stato. Per far ciò però quelle funzioni non usano un vero e proprio stack, ma aree di memoria chiamate (non vi fate ingannare dal nome!) *IOStack*, *DiskStack* e *AuxStack*. Il fatto che non si tratta di un vero e proprio stack ha una importante conseguenza. Supponiamo di aver installato un programma residente che usi una funzione del DOS; se il programma «scatta» mentre è in esecuzione una funzione del DOS che usi la stessa area di memoria, alle informazioni salvate in que-

```
program Crash;
uses
  Dos;
var
  Reg: registers;
  PrevInt5: procedure;
  DOSStr: string;
procedure NuovoInt5; interrupt;
begin
  Reg.AX := $0100;          (* immissione da tastiera *)
  MsDos(Reg)
end;
begin
  GetIntVec(5, Addr(PrevInt5));
  SetIntVec(5, Addr(NuovoInt5));
  Writeln('Sostituito INT 5 con NuovoInt5');
  Writeln('Digita dei caratteri, poi Print-Screen');
  Writeln('e un altro carattere: otterrai un bel crash!');
  DOSStr[0] := Chr(80);
  Reg.DS := DSeg;
  Reg.DX := Ofs(DOSStr);
  Reg.AX := $0A00;          (* immissione di una stringa *)
  MsDos(Reg);
  Writeln;
  Writeln('Se non hai "osato" premere Shift-PrtSc, possiamo');
  Writeln('rimettere a posto INT 5');
  SetIntVec(5, Addr(PrevInt5));
  Writeln('Fatto.')
end.
```

Figura 6 - Un programma che dimostra cosa succede se si dimentica che le funzioni del DOS non sono «rientranti».

```
(*$M 1024,0,0*)
program NumCapsScroll;
(* NB: Programma poco affidabile!!! *)
uses
  Dos, Crt;
var
  Reg: registers;
  PrevInt8: procedure;

procedure CLI; inline($FA);         (* disabilita gli interrupt *)

procedure PUSHF; inline($9C);       (* mette i flag nello stack *)

procedure NuovoInt8; interrupt;
var
  KBFlag: byte;
  PrevX, PrevY: integer;
begin
  PrevX := WhereX;
  PrevY := WhereY;
  KBFlag := Mem[$0040:$0017];
  GotoXY(60,25);
  if (KBFlag and $40) <> 0 then Write('Caps')
  else Write('    ');
  if (KBFlag and $20) <> 0 then Write(' Num')
  else Write('    ');
  if (KBFlag and $10) <> 0 then Write(' Scroll')
  else Write('       ');
  GotoXY(PrevX, PrevY);
  PUSHF;
  CLI;
  PrevInt8
end;
begin
  GetIntVec(8, Addr(PrevInt8));
  SetIntVec(8, Addr(NuovoInt8));
  Keep(0)
end.
```

Figura 7 - Un programma che associa all'INT 8, quello attivato dall'orologio interno, una routine che mostra lo stato dei tasti Num Lock, Caps Lock e Scroll Lock. Poiché l'INT 8 viene eseguito circa 18,2 volte al secondo, l'informazione che appare sul video sembra aggiornata «in tempo reale». Le funzioni WhereX e WhereY, come la procedura GotoXY, usano però le funzioni del BIOS, e nulla si fa per tener conto della «non rientranza» di queste: di qui la scarsa affidabilità del programma.

sta dalla funzione interrotta si sovrappongono quelle salvate dalla funzione «residente», con il risultato di impedire il corretto ripristino dello stato del programma interrotto. Si dice in gergo che ciò è causato dal fatto che le funzioni del DOS non sono «rientranti».

Le funzioni da 01h a 0Ch, ad esempio, usano normalmente l'*IOStack*; se se ne chiama una mentre un'altra è in esecuzione il disastro è praticamente assicurato. Tanto per consentirvi di toccare con mano questa dura realtà, vi propongo nella figura 6 un programma dal nome abbastanza significativo: CRASH.PAS. Questo sostituisce al solito INT 5 una routine che chiama la funzione 01h del DOS (immissione di caratteri dallo standard input); poi vi chiede di digitare una stringa e si predispone a riceverla mediante la funzione 09h, invitandovi però a premere, tra gli altri tasti, anche Print-Screen. In questo modo verrà chiamata una funzione che usa *IOStack* mentre ne è in esecuzione un'altra che usa (o meglio, crede di poter usare) la stessa area di memoria. Il risultato sarà un inchiodamento del PC. Provare per credere.

Il meccanismo delle funzioni del BIOS è un po' diverso, ma non troppo; in una parola, possiamo dire che anche queste sono «non rientranti». In realtà non capita sempre di imbattersi in simili problemi; potete forse prendere ad esempio l'ultimo programma che vi propongo questo mese. NUMCAPS.PAS, nella figura 7, illustra un altro possibile tipo di attivazione di un interrupt: la routine *NuovoInt8* viene infatti associata a quell'INT 8 che, come dicevamo prima, scatta circa 18,2 volte al secondo su iniziativa dell'orologio interno. C'è anche un'altra novità: poiché non si vogliono eliminare gli effetti dell'INT 8 originale, viene salvato in una variabile di tipo **procedure** l'indirizzo della routine che era associata all'interrupt; *NuovoInt8* quindi, dopo aver fatto quello che deve fare (visualizza lo stato dei tasti Num Lock, Caps Lock e Scroll Lock, letto nell'area dati del BIOS), chiama anche la vecchia routine dopo un PUSHF e un CLI, simulando così una istruzione INT.

Non sono riuscito a provocare inchiodamenti della macchina o simili con NUMCAPS.PAS, ma è sicuro che in qualche caso l'ignorare completamente la «non rientranza» del BIOS non sarebbe senza conseguenze.

Vedremo la prossima volta come realizzare programmi più affidabili.

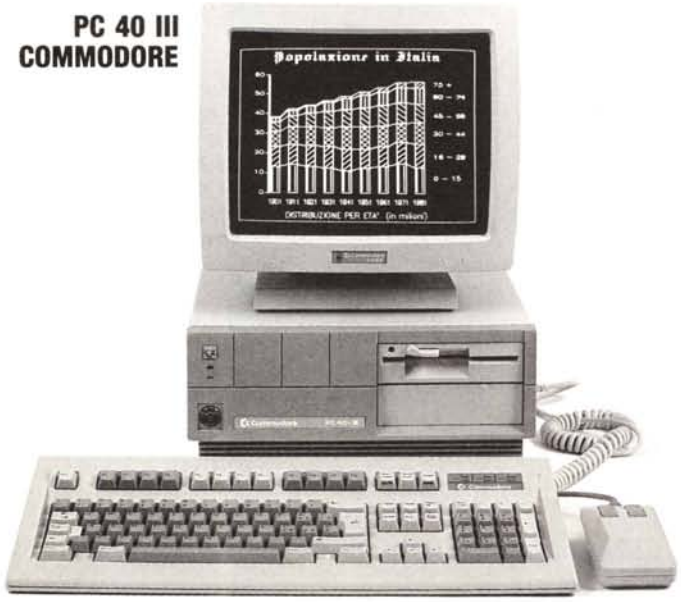
RISOLTO UN ANTICO DILEMMA.

CI DISPIACE PER I NOSTRI CONCORRENTI. SIA PER QUALITÀ CHE PER PREZZO.

Non è facile riuscire a produrre Personal Computer che abbiano qualità, affidabilità, prestazioni, facilità d'uso e compatibilità, senza far lievitare i prezzi. Ma, come potrete verificare, i Personal Computer Commodore hanno salvato capra e cavoli. Contribuendo ad introdurre una novità rilevante nel campo dell'informatica: il buon senso. Per sapere qual è il Concessionario Sistemi Professionali più vicino telefona al:

NUMERO VERDE 1678-27012

PC 40 III COMMODORE



MODELLO	CPU	CLOCK	RAM	DRIVES	MONITOR	PREZZO AL PUBBLICO IVA ESCLUSA
PC 10 SD3M	8088	4.77-8-10	640 Kb	1x3" ½ x 360/720 Kb	12" MONOCROM.	1.360.000
PC 10 DD3M	8088	4.77-8-10	690 Kb	2x3" ½ x 360/720 Kb	12" MONOCROM.	1.530.000
PC 10 SD5M	8088	4.77-8-10	640 Kb	1x5" ¼ x 360 Kb	12" MONOCROM.	1.360.000
PC 10 DD5M	8088	4.77-8-10	640 Kb	2x5" ¼ x 360 Kb	12" MONOCROM.	1.530.000
PC 10 SD3C	8088	4.77-8-10	640 Kb	1x3" ½ x 360/720 Kb	14" COLORE	1.675.000
PC 10 DD3C	8088	4.77-8-10	640 Kb	2x3" ½ x 360/720 Kb	14" COLORE	1.845.000
PC 10 SD5C	8088	4.77-8-10	640 Kb	1x5" ¼ x 360 Kb	14" COLORE	1.675.000
PC 10 DD5C	8088	4.77-8-10	640 Kb	2x5" ¼ x 360 Kb	14" COLORE	1.845.000
PC 20 HD5M	8088	4.77-8-10	640 Kb	1x5" ¼ x 360 Kb HD x 20 Mb	12" MONOCROM.	2.095.000
PC 20 HD5C	8088	4.77-8-10	640 Kb	1x5" ¼ x 360 Kb HD x 20 Mb	14" COLORE	2.410.000
PC 30 HD3M	80286	6-8-12	640 Kb	1x3" ⅓ x 1.44 Mb HD x 20 Mb	14" MONOCROM.	IN OFFERTA 2.490.000
PC 30 HD3C	80286	6-8-12	640 Kb	1x3" ½ x 1.44 Mb HD x 20 Mb	14" COLORE	3.305.000
PC 40 HD5M	80286	6-12	1 Mb	1x5" ¼ x 1.2 Mb HD x 40 Mb	14" VGA MONOCROM.	4.990.000
PC 40 HD5C	80286	6-12	1 Mb	1x5" ¼ x 1.2 Mb HD x 40 Mb	14" VGA COLORE	-
PC 50/40	80386SX	16-8	1 Mb	1x3" ½ x 1.44 Mb HD x 40 Mb	14" MONOCROM.	-
PC 50/100	80386SX	16-8	1 Mb	1x3" ½ x 1.44 Mb HD x 100 Mb	14" MONOCROM.	-

Commodore®

L'INFORMATICA SENZA DILEMMI.