di Sergio Polini

«Start-up» di un programma residente

Ricapitoliamo. A gennaio abbiamo introdotto l'argomento «programmi residenti» vedendo velocemente come questi debbano e possano essere attivati mediante un interrupt, ma anche come ciò vada effettuato con grande attenzione, al fine di evitare comportamenti erratici del nostro povero PC. Abbiamo visto anche un esempio di programma residente in grado di... inchiodare la macchina. A febbraio siamo scesi nelle pieghe più nascoste del DOS, onde scovare quei suoi flag interni e quei suoi meccanismi di cui bisogna tenere conto per scrivere programmi più affidabili. Il mese scorso, infine, abbiamo visto come procedere per evitare eventuali problemi con il BIOS, ma anche come usare il BIOS per attivare routine residenti, ovvero, nel nostro caso, per pervenire all'attivazione della procedura EseguiTSR. Ora vedremo cosa succede quando il TSR assume il controllo delle operazioni. Ricordo a tutti che la unit TSR completa è già a disposizione nel file TSRTP100.ZIP su MC-Link

Quando un normale programma Turbo Pascal parte, prima del codice che noi abbiamo scritto viene eseguito un codice di «start-up», che provvede ad impostare lo stack e lo heap, a settare adeguatamente alcune variabili, a reindirizzare alcuni interrupt. Un lavoro necessario a far sì che poi il nostro programma si comporti secondo le aspettative.

La procedura *EseguiTSR* (figura 1) svolge la stessa funzione per un programma residente; come prima cosa, ad esempio, gli riassegna il suo stack, ovvero i valori che i registri SS e SP avevano quando il programma era stato installato (e che la procedura *Installa* aveva ottenuto dalle funzioni predefinite *SSeg* e *SPtr*), facendo ricorso a quella procedura *NuovoStack* che avevamo visto a febbraio.

Stato del video

Dopo di ciò, si passa a controllare che il video sia in modo testo con 25 righe e 80 colonne, l'unica situazione in cui la unit TSR accetta di operare (la gestione di situazioni diverse richiederebbe operazioni più complesse, che meriterebbero — e magari in un prossimo futuro meriteranno — una trattazione a parte).

La procedura *GetInfoVideo* (figura 2) si incarica di individuare il tipo di video installato. Per far ciò prova per prima cosa se si tratta di un VGA, chiamando la funzione 1Ah dell'INT 10h. Questa ritorna varie informazioni se eseguita su una macchina con VGA, ma soprattutto, in questo caso, ritorna il valore 1Ah nel registro AL per confermare di aver potuto operare; se non troviamo questo valore,

```
procedure EseguiTSR;
begin
  NuovoStack;
  GetInfoVideo(SchedaVideo, Modo, Righe);
  if (Modo in [2,3,7]) and (Righe = 25) then begin
    GetStatoVideo(SchedaVideo, Modo);
    GetIntVec($1B, Addr(PrevInt1B));
    SetIntVec($1B, Addr(NuovoInt1B));
    GetIntVec($23, Addr(PrevInt23));
    SetIntVec($23, Addr(NuovoInt23));
    GetIntVec($24, Addr(PrevInt24));
    SetIntVec($24, Addr(NuovoInt24));
    GetCBreak(PrevBreak);
    SetCBreak(FALSE);
    GetInfoEstesaErrori(InfoEstesaErrori);
    GetPSP(PrevPSP);
    SetPSP(NuovoPSP);
    GetDTA(SegPrevDTA, OfsPrevDTA);
    SetDTA(NuovoPSP, $80);
    TSRProg;
    SetDTA(SegPrevDTA, OfsPrevDTA);
    SetPSP(PrevPSP);
    SetInfoEstesaErrori(InfoEstesaErrori);
    SetCBreak(PrevBreak);
    SetIntVec($1B, Addr(PrevInt1B));
    SetIntVec($23, Addr(PrevInt23));
    SetIntVec($24, Addr(PrevInt24));
    SetStatoVideo(SchedaVideo, Modo)
  end
  else
    Beep;
  PrevStack;
  InTSRKey := FALSE
end;
```

Figura 1 - La procedura EseguiTSR, alla quale viene passato il controllo non appena il programma residente viene attivato.

vuol dire che non c'è VGA. Assumiamo quindi che sia presente una scheda EGA, e proviamo con la sottofunzione 10h della funzione 12h dello stesso INT 10h: questa ci riporta in BL dei valori da 0 a 3 per indicare se sulla scheda EGA sono installati 64K, 128K, 192K o 256K di memoria; se però ritroviamo in BL lo stesso valore che vi avevamo messo (10h), allora vuol dire che non c'è neppure una EGA. A questo punto possiamo andare a leggere la word del BIOS che contiene l'indicazione della porta di I/O attraverso la quale si accede al *Crt Controller*: se è 3B4h vuol dire che siamo in presenza di una scheda monocromatica (quella che consente solo il modo testo a 80 colonne), se invece è 3D4h vuol dire che sulla macchina è montata una CGA.

A questo punto possiamo determinare il modo del video: se non è «testo 25x80» (ricordo che i modi 2, 3 e 7 del BIOS corrispondono alle costanti *BW80*, *CO80* e *Mono* della **unit** *Crt*) usciamo subito; *EseguiTSR* prenderà atto della situazione e restituirà senza indugi il controllo al programma interrotto. Altrimenti proseguiamo per controllare se, nonostante il BIOS ci abbia riferito che ci sono solo 25 colonne di testo, non ce ne siano in realtà di più grazie alla versatilità delle schede EGA e VGA.

Se tutto conferma che siamo nel modo desiderato, si chiama la procedura *GetStatoVideo* (figura 2) perché provveda a salvare nell'array *BuffVideo* (definito nella **interface** come array di 2000 *word*) quanto appare sullo schermo e nelle variabili *Riga* e *Colonna* la posizione del cursore. Nel caso sia installata una scheda CGA, si provvede a «spegnere» il video prima di leggerne il contenuto e poi a «riaccenderlo», onde evitare un fastidioso sfarfallio.

La variabile *BuffVideo* è dichiarata nella **interface** allo scopo di metterla a disposizione di programmi che usino la **unit**, ad esempio per salvare su disco l'apparenza del video.

La procedura *SetStatoVideo* (sempre in figura 2) opera «al contrario»: riporta

```
procedure GetInfoVideo(var Scheda: TipoScheda; var Modo, Righe: byte);
var
  Reg   : registers;
begin
  Reg.AX := $1A00;
  Intr($10,Reg);
  if Reg.AL = $1A then
    Scheda := VGA
  else begin
    Reg.AH := $12;
    Reg.BL := $10;
    Intr($10, Reg);
    if Reg.BL <> $10 then
      Scheda := EGA
    else if MemW[$0040:$0063] > $3D0 then
      Scheda := CGA
    else
      Scheda := MDA
  end;
  Reg.AH := $0F;
  Intr($10,Reg);
  Modo := Reg.AL;
  if not (Modo in [2,3,7]) then    (* accettati solo modi testo 25x80 *)
    Exit;
  if (Scheda = EGA) or (Scheda = VGA) then begin
    Reg.AX := $1130;
    Reg.BH := 0;
    Intr($10,Reg);
    Righe := Reg.DL + 1
  end
  else
    Righe := 25
end;

procedure GetStatoVideo(Scheda: TipoScheda; Modo: byte);
var
  p: ^BuffVideo;
begin
  if Modo = 7 then p := Ptr($B000,0)
  else p := Ptr($B800,0);
  Riga := WhereY;
  Colonna := WhereX;
  if Scheda = CGA then             (* "spegne" il video *)
    Port[$3D8] := $25;
  Move(p^, MemoriaVideo, 4000);
  if Scheda = CGA then             (* "riaccende" il video *)
    if Modo = 2 then Port[$3D8] := $2D
    else Port[$3D8] := $29
end;

procedure SetStatoVideo(Scheda: TipoScheda; Modo: byte);
var
  p: ^BuffVideo;
begin
  if Modo = 7 then p := Ptr($B000,0)
  else p := Ptr($B800,0);
  if Scheda = CGA then             (* "spegne" il video *)
    Port[$3D8] := $25;
  Move(MemoriaVideo, p^, 4000);

  if Scheda = CGA then             (* "riaccende" il video *)
    if Modo = 2 then Port[$3D8] := $2D
    else Port[$3D8] := $29;
  GotoXY(Colonna, Riga)
end;
```

Figura 2 - Le routine che controllano il «modo» del video, ne salvano la configurazione e poi la ripristinano prima che il TSR torni al programma interrotto.

sul video quanto salvato nell'array e rimette al suo posto il cursore, in modo da far sì che, al termine della esecuzione del programma residente, questo lasci le cose come le aveva trovate.

In verità le routine della figura 2 potrebbero essere un po' più ricche; potrebbero ad esempio conservare e poi rispristinare anche la forma del cursore. Non ho voluto tuttavia appesantire la trattazione (già non breve...) anche con i molti dettagli che sarebbero stati necessari per una più completa analisi dei possibili stati del video (ad esempio: che succede se il programma interrotto stava scrivendo su una pagina video diversa dalla zero?). Fare questo, infatti, avrebbe comportato lo sviluppo di un tema nel tema, e quindi ho preferito rimandare ad un eventuale futuro argomenti che meriterebbero un autonomo approfondimento. Quello che si può fare è comunque fornire un esempio di lettura e scrittura delle righe di scansione del cursore, in quanto sono solo poche righe di codice (figura 3). Quanto al resto, potreste scegliere di controllare in *GetInfoVideo* quale è la pagina video attiva (chiamando l'INT 10h con $0F in AH; il risultato sarà in BH) e inibire l'attivazione del TSR se questa non è la pagina zero.

```
procedure NuovoInt1B; interrupt;
begin
  CtrlBreak := TRUE;
end;

procedure NuovoInt23; interrupt;
begin
  CtrlC := TRUE;
end;

procedure NuovoInt24(AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
interrupt;
begin
  ErroreCritico := (DI and $00FF) + 150;   (* codice d'errore *)
  AX := AX and $FF00;                      (* "ignora" *)
end;
```

Figura 4 - Le procedure che il programma residente associa agli INT 1Bh, 23h e 24h.

Control-C, Control-Break, Errori critici

Se un programma normale non può permettersi di subire passivamente brusche interruzioni della sua attività, le cose si fanno anche più delicate nel caso di un TSR. Un programma normale può non riuscire a chiudere ordinatamente i suoi file, un TSR aggiunge a danni di questo genere anche un ben probabile inchiodamento della macchina.

Per la **unit** TSR ho scelto la soluzione più semplice e, credo, anche la più comoda per chi programmi in Turbo Pascal: con le procedure predefinite *GetIntVec* e *SetIntVec* si salvano gli indirizzi delle routine associate agli interrupt 1Bh (Ctrl-Break), 23h (Ctrl-C) e 24h (gestione errori critici) in apposite variabili, e si associano agli stessi interrupt altre routine (figura 4). *NuovoInt1B* e *NuovoInt23* non fanno altro che assegnare il valore TRUE alle variabili *CtrlBreak* e *CtrlC*; *NuovoInt24* assegna invece alla variabile *ErroreCritico* il codice descrittivo dell'errore (contenuto nel byte «basso» del registro DI) aumentato di 150, in modo da renderlo uguale al codice che verrebbe ritornato da un qualsiasi programma Turbo Pascal, e pone uno zero in AL. Quest'ultima operazione ha lo stesso effetto che avrebbe la scelta della opzione «Ignora» qualora l'INT 24h non fosse intercettato e il DOS proponesse il messaggio «Annulla, Riprova, Ignora?».

Qui è però necessario aprire una parentesi. Quel messaggio presenta alcune varianti a partire dal DOS 3.3, che a volte propone anche una opzione «Tralascia»; questa è analoga a «Ignora», con una differenza: mentre con questa si cerca di far credere al DOS che non si è verificato alcune errore, «Tralascia» fa sì che la funzione DOS incappata nel problema ritorni con un codice d'errore. Era comunque possibile già dal DOS 3.1 scegliere «Tralascia» in una routine associata all'INT 24h (mettendo un 3 in AL), e, soprattutto, a partire dal DOS 3.1 viene esercitato un controllo sulla scelta: i bit 3, 4 e 5 di AH vengono azzerati se non sono ammissibili, rispettivamente, le scelte «Tralascia», «Riprova» e «Ignora» (non è pensabile, ad esempio, che possa essere ignorata la segnalazione di «drive non pronto» se si sta tentando un accesso al disco); se la scelta effettuata dall'utente non è accettata dal DOS, «Tralascia» viene convertita automaticamente in «Abbandona» e le altre due in «Tralascia». Molto ragionevole. Soprattutto ciò consente di scegliere comunque «Ignora» in quanto, nel caso ciò non risultasse possibile, ci penserebbe il DOS a tradurre in «Tralascia». Il problema sta solo nel fatto che le precedenti versioni del DOS non sono altrettanto sagge: accettano «Ignora» anche quando non sarebbe

```
Program RigheScansione;
uses Dos;
var
  Prima, Ultima: byte;
procedure GetRigheScansione(var Prima, Ultima: byte);
var
  Reg: registers;
begin
  Reg.AH := 3; (* funzione dell'INT 10h *)
  Reg.BH := 0; (* pagina 0 *)
  Intr($10, Reg);
  Prima := Reg.CH;
  Ultima := Reg.CL;
end;
procedure SetRigheScansione(Prima, Ultima: byte);
var
  Reg: registers;
begin
  Reg.AH := 1; (* funzione dell'INT 10h *)
  Reg.CH := Prima;
  Reg.CL := Ultima;
  Intr($10, Reg);
end;
begin
  GetRigheScansione(Prima, Ultima);
  Writeln('Righe scansione: ',Prima,' ',Ultima);
  Writeln('Premi ENTER. ');
  Readln;
  SetRigheScansione($20, 0);
  Writeln('Cursore scomparso.');
  Writeln('Premi ENTER per ripristinarlo. ');
  Readln;
  SetRigheScansione(Prima, Ultima);
end.
```

Figura 3 - Un breve programma che illustra come ottenere e impostare la prima e l'ultima riga di scansione del cursore, cioè la sua forma. Passando i valori $20 e 0 alla procedura SetRigheScansione il cursore scompare.

possibile, e soprattutto non dispongono della opzione «Tralascia». Ne segue ad esempio, come vedremo in un prossimo appuntamento, che il tentativo di scrivere su una stampante spenta provoca una lunga misteriosa attesa e poi la generazione del codice d'errore «drive non pronto»!

Meglio che niente. Meglio, in particolare, di incontrollate interruzioni dei nostri programmi residenti.

A differenza di *CtrlBreak* e di *CtrlC*, dichiarate nella interface, la variabile *ErroreCritico* viene dichiarata nella **implementation**: è cioè invisibile al programma che usi la **unit**. Sappiamo infatti che gli errori critici si verificano in occasione di errori di I/O, e che possono essere intercettati, in un normale programma Turbo Pascal, disattivando la direttiva $I e interrogando la funzione *IOResult*. La **unit** TSR propone una funzione *ErroreIO* (figura 5) da usare al posto di *IOResult*. In essa viene per prima cosa registrato nella variabile *EIO* il valore ritornato proprio da *IOResult*, ma poi si controlla subito se per caso la routine associata all'INT 24h non abbia avvalorato la variabile *ErroreCritico*; in caso affermativo, si assegna il valore di questa a *EIO* (e si riazzera *ErroreCritico*). Poiché la funzione ritorna il valore di *EIO* così determinato, se ne ottiene un comportamento del tutto analogo a quello della normale funzione *IOResult*, che può quindi essere sostituita, in un programma residente, da *ErroreIO*. Ne vedremo esempi quando esamineremo il programma TSRDEMO, al termine della illustrazione della nostra **unit**.

```
function ErroreIO: word;
var
  EIO: word;
begin
  EIO := IOResult;
  if ErroreCritico > 0 then begin
    EIO := ErroreCritico;
    ErroreCritico := 0
  end;
  ErroreIO := EIO
end;
```

Figura 5 - La funzione ErroreIO, che va usata al posto della tradizionale IOResult.

Informazione estesa sugli errori

La gestione degli errori nel DOS 1.x era piuttosto primitiva: se una funzione DOS falliva, veniva semplicemente riportato un -1 (0FFh) nel registro AL. Il DOS 2.0 ha introdotto una diversa strategia: la situazione d'errore viene segnalata settando il flag di *carry*, mentre in AX viene posto un codice d'errore. Con il DOS 3.0 si è fatto un ulteriore passo in avanti, soprattutto per la necessità di fornire al programma sufficienti informazioni nel caso di problemi connessi con le nuove possibilità del sistema operativo, quali il supporto delle reti locali, compresa la possibilità di condividere risorse o di bloccarne l'accesso ad altri utenti. Si dispone infatti di una funzione 59h che, se chiamata dopo il verificarsi di un errore, ritorna in AX un codice d'errore «esteso» (che può cioè assumere valori superiori a 12h, che costituiva il massimo nelle versioni 2.x del DOS), in BH un codice relativo alla «classe» dell'errore (ad esempio: 05h per un problema hardware, 07h per un errore software del programma in esecuzione), in BL un suggerimento sulla migliore prosecuzione (ad esempio: 03h per «richiedi all'utente di immettere una informazione corretta, come la lettera del drive o il nome del file», 05h per «esci immediatamente dal programma»), in CH il luogo in cui l'errore si è verificato (ad esempio: 03h per «rete», 05h per «memoria»).

Oltre ai tre registri AX, BX e CX, la funzione 59h altera anche i registri DX, SI, DI, DS ed ES; ecco perché la procedura *GetInfoEstesaErrori* li salva tutti nella struttura *InfoEstesaErrori*, passatale da *EseguiTSR* come parametro variabile, in modo da evitare che il programma residente, se attivato prima che il programma interrotto abbia fatto uso di quelle informazioni, le alteri. Non è chiaro perché nel sorgente di SNAP.ASM (il programma con cui la *MSDOS Encyclopedia* «documenta» i programmi TSR) a quei registri vengano aggiunte tre word nulle, così come ben poco viene detto della funzione 5Dh, da usare per ripristinare l'informazione estesa degli errori. In figura 6 si sono comunque seguite fedelmente le indicazioni della *Encyclopedia*.

```
procedure GetInfoEstesaErrori(var Info: RecInfoEstesaErrori);
var
  Reg: registers;
begin
  if VersioneDOS >= $30A then begin
    Reg.AH := $59;
    Reg.BX := 0;
    MsDos(Reg);
    Info.AX := Reg.AX;
    Info.BX := Reg.BX;
    Info.CX := Reg.CX;
    Info.DX := Reg.DX;
    Info.SI := Reg.SI;
    Info.DI := Reg.DI;
    Info.DS := Reg.DS;
    Info.ES := Reg.ES;
    Info.W1 := 0;
    Info.W2 := 0;
    Info.W3 := 0
  end
end;

procedure SetInfoEstesaErrori(Info: RecInfoEstesaErrori);
var
  Reg: registers;
begin
  if (VersioneDOS >= $30A) and (VersioneDOS < $0A00) then begin
    Reg.DS := Seg(Info);
    Reg.DX := Ofs(Info);
    Reg.AX := $5D0A;
    MsDos(Reg)
  end;
end;
```

Figura 6 - Le procedure che salvano e poi ripristinano l'informazione «estesa» sugli errori, disponibile a partire dal DOS 3.0.

PSP e DTA

Abbiamo già avuto occasione di discutere del *Program Segment Prefix* e della *Disk Transfer Area*, ma lo spazio non ci è più sufficiente per illustrare i «trucchi» necessari per agire su queste strutture in un programma residente.

Vi do quindi appuntamento al mese prossimo. Parleremo non solo degli ultimi aspetti della procedura *EseguiTSR*, ma termineremo anche l'illustrazione della unit, vedendo in dettaglio la procedura *Installa* e l'uso che questa fa dell'INT 2Fh.

Microforum

MITO

PREFORMATTED & VERIFIED • 100% ERROR FREE •

DOS 3.0 OR HIGHER

New!

Nuovo da Microforum!
I dischi Mito oggi li trovi anche preformattati e verificati:
con poche lire in più ti assicuri un risparmio di tempo,
la certezza della qualità e una velocità impagabile nelle situazioni in cui devi salvare i dati senza l'obbligo di uscire dal programma.

Mito Microforum, *ancora una volta più avanti.*

Media Disk Antonelli
12, Via Ciociaria - 00162 Roma
Telefono 06/4240379

Floppy's Market
5, P.za del Popolo
56029 S.Croce sull'Arno (PI)
Telefono 0571/35124

Simple Soft
Via Cesana 6 - 20132 Milano
Telefono 02/2841141

Microforum
MANUFACTURING INC.
TORONTO - CANADA