di Sergio Polini

La unit TSR al lavoro

Approfitto anche questa volta del corsivo iniziale per scambiare due chiacchiere con un lettore. Francesco Guarnieri, di Lastra a Signa (FI), mi ha scritto di aver copiato con grande attenzione i listati della unit EXECSWAP (ottobre, novembre e dicembre 1989), che però non funzionerebbe in modo corretto. Prima di lui anche Edgardo Firinu Missio di Monfalcone (GO) aveva lamentato un analogo inconveniente, ma poco dopo mi aveva riscritto per comunicarmi di aver finalmente risolto il problema: un MOV DS, AX gli era diventato MOV DX, AX. Nessun altro lettore, e nessun utente di MC-Link, ha mai denunciato difficoltà con la unit, i cui listati, così come tutti quelli pubblicati, non sono altro che la riproduzione fotografica di sorgenti che, una volta compilati, hanno funzionato regolarmente almeno su una macchina. Spesso (come nel caso di EXECSWAP e TSR) su più di una. Consiglio quindi a Francesco Guarnieri di controllare ancora una volta. Una sola. In caso di ulteriori problemi mi riscriva pure: troveremo comunque una soluzione

La volta scorsa abbiamo finalmente terminato l'illustrazione dei sorgenti della unit TSR, dandoci appuntamento a questo mese per vedere un paio di esempi concreti. È stato un lungo cammino: il discorso era cominciato a gennaio, ma per alcuni temi (struttura del PSP, «contesto» di un programma, ecc.) si potrebbe risalire ancora più indietro nel tempo. Visto che siamo alla fine di un lungo cammino, posso confessarvi che ero preparato ad alcune critiche: non esiste un unico modo per scrivere una unit del genere, e fin dall'inizio avevo accennato alle diverse soluzioni adottate da alcuni autori americani. Non ho mai pensato di aver risolto ogni problema nel modo migliore. Visto che però le critiche non sono arrivate (neppure dai numerosi utenti di MC-Link che da tempo hanno prelevato la unit via modem) farò da solo; si può infatti ripetere per il programmatore quanto una volta Einstein disse per lo scienziato: è lui che sa «dove la scarpa fa male»; è proprio l'autore di un programma, o di una teoria, quello che meglio ne conosce i punti deboli. Vi mostrerò quindi dove modificherei la unit per renderla più versatile e più funzionale.

SNAP.PAS

Vediamo intanto un primo esempio di uso della unit. Dal momento che, come ormai ben sapete, abbiamo attinto a piene mani dalla *MS-DOS Encyclopedia*, mi sembra giusto iniziare dalla versione in Turbo Pascal del programma residente lì illustrato. Si chiama SNAP.ASM e non fa altro che salvare su disco, in un file chiamato SNAP.IMG, una «immagine» dello schermo, cioè una copia della memoria video. La sua «traduzione» in Turbo Pascal (figura 1) è piuttosto semplice, ma già consente di mettere a fuoco importanti dettagli.

```
(*$M 4096,0,0*)
(*$S-*)
Program Snap;

uses
  Crt, TSR;

var
  f: file of BuffVideo;
  Errore: word;

(*$F+*)
procedure SalvaSchermata;
begin
  (*$I-*) Rewrite(f) (*$I+*);
  Errore := ErroreIO;
  if Errore = 0 then
    Write(f, MemoriaVideo);
  (*$I-*) Close(f) (*$I+*);
end;

begin
  Assign(f, 'SNAP.IMG');
  Installa('SNAP', SalvaSchermata, $F1, 0, 6);
end.
```

Figura 1
SNAP.PAS, la versione in Turbo Pascal di SNAP.ASM, il programma residente illustrato nella MS-DOS Encyclopedia.

In primo luogo notiamo alcune direttive di compilazione. Con la direttiva $M limitiamo a 4096 byte lo spazio da riservare allo stack e non lasciamo nulla per lo heap, in modo da contenere al minimo l'occupazione di memoria; è soprattutto importante ricordare di stabilire una dimensione massima «non ingombrante» per lo heap. Con la direttiva $S disabilitiamo il controllo dello *stack overflow:* molte routine della unit vengono infatti eseguite usando lo stack del programma interrotto (quando si controlla se sussistono le condizioni per un'attivazione), poi, appena attivato il programma residente vero e proprio, viene momentaneamente ripristinato lo stack che questo aveva in origine, al momento della installazione. I meccanismi di controllo dello *stack overflow* non prevedono ovviamente che si passi repentinamente da uno stack ad un altro, e vanno quindi disabilitati.

Più giù troviamo la dichiarazione delle variabili, la prima delle quali appartiene ad un tipo «esportato» dalla unit. Ricorderete che il tipo *BuffVideo* e la variabile *MemoriaVideo* erano stati dichiarati nella interface della unit proprio per possibili applicazioni di questo tipo.

Il corpo principale del programma si limita ad assegnare alla variabile f il nome del file e ad installare il programma. I parametri passati alla procedura *Installa* sono nell'ordine: il nome del programma residente, la procedura da eseguire quando il TSR può essere attivato, il byte di identificazione, il codice di scansione del tasto da premere per l'attivazione (zero per «solo i tasti di shift»), il byte che designa la combinazione desiderata dei tasti di shift (6 sta per "4 + 2", ovvero: Ctrl + Shift sinistro). *SalvaSchermata* è un parametro procedurale e quindi, come tutti i parametri di questo tipo, deve essere il nome di una procedura da chiamare con una *far call:* di qui l'uso — obbligatorio anch'esso — della direttiva $F.

Una volta rispettate queste poche regole, si può scrivere in piena libertà il

```
(*$M 8192,0,0*)
(*$S-*)
Program TsrDemo;

uses
  Dos, Crt, Printer, TSR;

const
  Testo = White;
  Sfondo = Blue;

type
  SetChar = set of char;

var
  Nome: string;
  f: file;
  k: char;
  Errore: word;

procedure Box(x0, y0, x1, y1: integer);
var
  i: integer;
begin
  GotoXY(x0,y0); Write(#218);
  for i := x0+1 to x1-1 do
    Write(#196);
  Write(#191);
  for i := y0+1 to y1-1 do begin
    GotoXY(x0,i); Write(#179);
    GotoXY(x1,i); Write(#179)
  end;
  GotoXY(x0,y1); Write(#192);
  for i := x0+1 to x1-1 do
    Write(#196);
  Write(#217)
end;

function ReadString(var s:string; Col,Riga,Max:integer; Ammessi:SetChar): char;
var
  k: char;
  p: integer;
  Ins: boolean;
  PrevStr: string;
begin
  PrevStr := s;
  TextColor(Black);
  TextBackGround(LightGray);
  Ins := TRUE;
  CtrlBreak := FALSE;
  p := Length(s);
  repeat
    GotoXY(Col, Riga);
    Write(s,'':(Max-Length(s)));
    GotoXY(Col+p, Riga);
    k := ReadKey;
  if CtrlBreak then begin          (* per un uso 'normale':   *)
    s := 'CtrlBreak';              (* if CtrlBreak then begin *)
    k := #13;                      (*   k := #27;             *)
    CtrlBreak := FALSE             (*   CtrlBreak := FALSE    *)
  end;                             (* end;                    *)
  if k = #0 then begin
    k := ReadKey;
    case k of
      #71: p := 0;                           (* Home *)
      #79: p := Length(s);                   (* End *)
      #82: Ins := not Ins;                   (* Ins *)
      #75: if p > 0 then Dec(p);             (* Sinistra *)
      #77: if p < Length(s) then Inc(p);     (* Destra *)
      #83: if p < Length(s) then             (* Del *)
             Delete(s, p+1, 1);
    end
  end
  else begin
    case k of
       #3: begin                             (* Ctrl-C *)
             s := 'CtrlC';           (* per un uso 'normale': *)
             k := #13                (* k := #27              *)
           end;
```

(continua a pag. 240)

codice della procedura da attivare; unica avvertenza, usare la funzione *ErroreIO* invece della tradizionale *IOResult* (ricordo anche che, come abbiamo visto nell'appuntamento di febbraio, va evitato l'uso delle procedure *Intr* o *MsDos* — o di istruzioni *inline* o di moduli in assembler — per chiamare funzioni DOS comprese tra la 01h e la 0Ch). Nel caso di SNAP.PAS si è seguita una strategia molto semplice: se si è verificato un qualsiasi errore non si scrive nulla sul file SNAP.IMG. Punto.

TSRDEMO.PAS

Nel file TSRTP100.ZIP, in cui sono racchiusi i file ASM, OBJ, PAS e TPU della unit TSR a beneficio degli utenti di MC-Link, è presente anche un «demo» un po' più complesso. Una volta che si abbia a disposizione la unit, si possono scrivere infiniti tipi di programmi residenti; il punto più delicato è comunque sempre lo stesso: non si può permettere che un errore, soprattutto un «errore critico», provochi la terminazione anomala delle operazioni, con conseguenze facilmente immaginabili. Il programma TSRDEMO.PAS (figura 2) ha proprio lo scopo di collaudare la «resistenza» della nostra unit a questo tipo di inconvenienti.

La novità più appariscente rispetto a SNAP.ASM è rappresentata dalla presenza di vari strumenti per l'interattività con l'utente: si apre una finestra bordata, si mostrano messaggi, si chiede un input dalla tastiera. La funzione *ReadString* già ci consente di vedere come usare un'altra variabile esportata dalla unit, *CtrlBreak*. La variabile viene posta uguale a FALSE all'inizio, e poi dopo ogni volta che viene trovata TRUE, cosa che avviene appunto se l'utente preme Ctrl-Break, grazie alla sostituzione della routine associata all'INT 1Bh. Una «normale» funzione come *ReadString* tradurrebbe probabilmente il Ctrl-Break in un ESC, come suggerito nel commento al listato della figura 2; in TSRDEMO, tuttavia, si fa sì che l'effetto sia quello di visualizzare un messaggio che confermi la corretta intercettazione dell'INT 1Bh.

L'uso della variabile *CtrlC* è in linea di principio identico, con una importante differenza esemplificata proprio dalla funzione *ReadString*. In essa infatti i tasti digitati dall'utente vengono letti mediante la funzione predefinita *ReadKey* che, usando l'INT 16h del BIOS, è «insensibile» alla pressione del Ctrl-C (non fa scattare l'INT 23h). Quando si

(segue da pag. 239)

```
            #8: if p > 0 then begin                    (* Backspace *)
                  Delete(s, p, 1);
                  Dec(p)
                end;
            #13: begin end;                            (* Enter *)
            #27: s := '';                              (* ESC *)
            else begin
              if k in Ammessi then begin
                if Ins then begin
                  if Length(s) < Max then begin
                    Inc(p);
                    Insert(k, s, p)
                  end
                end
                else if p < Max then begin
                  Inc(p);
                  if p <= Length(s) then s[p] := k
                  else s := s + k
                end
              end
            end
          end
        end
      until (k = #13) or (k = #27);
      if k = #27 then
        s := PrevStr;
      TextColor(White);
      TextBackGround(Blue);
      GotoXY(Col,Riga);
      Write(s,'':(Max-Length(s)));
      GotoXY(Col,Riga);
      ReadString := k
    end;

    (*$F+*)
    procedure Demo;
    begin
      TextColor(Testo);
      TextBackGround(Sfondo);
      Box(9,9,71,21);
      Window(10,10,70,20);
      ClrScr;
      Nome := '';
      GotoXY(15,6);
      Write('Come ti chiami?');
      k := ReadString(Nome, 31, 6, 12, [' ','A'..'Z','a'..'z']);
      if k <> #27 then begin
        ClrScr;
        GotoXY(1,1);
        Writeln('Ciao ',Nome,'!');
        GotoXY(1,4);
        Writeln('Apri il drive A:, poi premi un tasto.');
        repeat until KeyPressed;
        Assign(f,'a:pippo');
        (*$I-*) Rewrite(f) (*$I+*);
        Errore := ErroreIO;
        Writeln('Tentativo apertura file. Codice d''errore: ',Errore:3);
        (*$I-*) Close(f) (*$I+*);
        Errore := ErroreIO;
        Writeln('Tentativo chiusura file. Codice d''errore: ',Errore:3);
        GotoXY(1,11);
        Write('Premi ESC.');
        repeat k := ReadKey until k = #27;
        ClrScr;
        GotoXY(1,4);
        Writeln('Ora spegni la stampante (se collegata)');
        Writeln('poi premi un tasto.');
        repeat until KeyPressed;
        (*$I-*) Writeln(Lst,'pippo') (*$I+*);
        Errore := ErroreIO;
        Writeln('Tentativo uso stampante. Codice d''errore: ',Errore:3);
        GotoXY(1,11);
        Write('Premi ESC per tornare al programma interrotto.');
        repeat k := ReadKey until k = #27
      end;
      Window(1,1,80,25);
    end;
    (*$F-*)

    begin
      Installa('TSRDEMO',Demo,$F2,0,6)
    end.
```

Figura 2 - Il programma TSRDEMO, che mette alla prova il corretto funzionamento della unit TSR in situazioni di Ctrl-C, Ctrl-Break ed errori critici.

usa *ReadKey* quindi, il Ctrl-C va riconosciuto verificando se il carattere ritornato dalla funzione non abbia codice ASCII 3. Anche in questo caso un programma «normale» tradurrebbe probabilmente il carattere in ESC, ma TSRDEMO preferisce dare conferma che il Ctrl-C è stato correttamente gestito.

TSRDEMO apre una finestra sul video, vi chiede il vostro nome (fase durante la quale potete provare a premere Ctrl-Break o Ctrl-C), quindi vi invita ad aprire il drive A:. Lo scopo è naturalmente quello di provare a fare qualcosa (apertura di un file) su un disco inaccessibile. Seguendo le istruzioni, potrete vedere accendersi la spia del drive e poi apparire subito dopo una messaggio con il codice di errore 152: unità non pronta. Dopo aver premuto ESC, verrete invitati a spegnere la stampante; dopo il tentativo di stampa, vedrete un altro messaggio che vi comunicherà il codice d'errore 160: errore di scrittura. Questo è quello che succede se avete una versione del DOS pari o successiva alla 3.1; con versioni precedenti può capitare che il tentativo d'accesso ad una stampante spenta o assente provochi una lunga misteriosa attesa, seguita dalla generazione del codice d'errore 152 invece che 160. I motivi di questo comportamento sono da ricercare tutti nella storia del DOS, come avevamo già visto nella chiacchierata dello scorso aprile.

Quello che importa, comunque, è che TSRDEMO consente di verificare che la unit TSR offre strumenti sufficienti per mettere i vostri programmi residenti al riparo anche dagli errori critici.

Estensioni e modifiche

Il programma SNAP ha come byte di identificazione un $F1; a TSRDEMO è toccato un $F2. Non a caso diversi. Ma cosa succederebbe se i due byte fossero uguali? Se SNAP fosse già residente in memoria, l'installazione di TSRDEMO fallirebbe perché il programma, a causa dell'uso che fa dell'INT 2Fh, crederebbe di essere già stato installato. Una situazione del genere può ben capitare: nulla vieta che, sulla macchina su cui vogliamo usare un programma residente realizzato con la unit TSR, sia già residente un altro programma che faccia un analogo uso dell'INT 2Fh (in particolare, riconoscimento «di se stesso» mediante un byte di identificazione). A ciò si può ovviare con poco sforzo: basta infatti prevedere che il byte non venga inserito nel codice sorgente, ma venga invece dato dall'utente sulla riga comando. In questo modo, se l'installazione fallisse perché apparentemente già avvenuta, basterebbe ripetere provando con un byte diverso.

```
Reg.AH := $49;
Reg.ES := Word(Ptr(PrefixSeg, $2C)^);
MsDos(Reg);
```

Figura 3 - Il codice da aggiungere alla procedura Installa per sbarazzarsi dell'environment.

SNAP e TSRDEMO hanno però in comune la combinazione di tasti prescelta per l'attivazione. Se provate ad installare prima l'uno poi l'altro vedrete che, premendo i tasti Ctrl e Shift sinistro, viene sempre attivato solo quello che è stato installato per primo, mentre un beep avverte che l'altro non può venire attivato. La unit TSR, così come è ora, non offre rimedi a questa situazione; ho infatti preferito lasciare a voi la scelta della soluzione.

In primo luogo, infatti, si potrebbe procedere in modo analogo a quanto già accade per il byte di identificazione: la routine che viene associata all'INT 2Fh (vista il mese scorso) potrebbe controllare anche i codici dei tasti invece che limitarsi al byte di identificazione. Ciò però sarebbe d'aiuto solo nel caso di programmi residenti realizzati con la unit TSR, mentre non offrirebbe alcuna garanzia nel caso di programmi realizzati in altro modo. Si potrebbero però anche modificare le routine da associare agli interrupt 08h e 28h; riguardando i listati pubblicati a marzo, potrete notare che alla variabile *InTSRKey* (che vale TRUE quando viene riconosciuta la combinazione di tasti che si vuole attivi il programma residente) viene assegnato FALSE nel caso non siano verificate tutte le condizioni che consentono l'attivazione. Sia in *NuovoInt8* che in *NuovoInt28* ciò accade nell'ambito di una istruzione **if** *TSRAttivabile* **then ... else ...**; l'unica differenza è rappresentata dalla chiamata della procedura *Beep*, che è presente solo in *NuovoInt8*. In entrambi i casi, basta eliminare il ramo **else** di quella istruzione per ottenere un comportamento diverso: assegnare FALSE a *InTSRKey* equivale ad annullare a tutti gli effetti la pressione dei tasti attivatori; eliminando l'assegnazione si otterrebbe invece solamente di rimandare l'attivazione del programma residente, in modo tale che diventerebbe possibile attivare due programmi l'uno dopo l'altro con una sola pressione degli stessi tasti.

Quanto ciò sia desiderabile, lascio a voi giudicare. Non mi sembra esagerato pretendere che l'utente sia consapevole di quanti e quali programmi residenti sono installati sulla sua macchina e delle combinazioni di tasti che li attivano. Da tale punto di vista, sarebbe sufficiente prevedere anche in questo caso la possibilità di indicare i codici di scansione e di shift nella riga comando.

C'è infine da considerare l'environment. Questa area di memoria occupa 160 byte per default, ma le più recenti versioni del DOS consentono di espanderla fino a 32K. Ogni programma si porta dietro una copia dell'environment, e ciò potrebbe rappresentare un problema in quanto si rischia di tenere inutilmente occupata memoria preziosa. Il rimedio è molto semplice: basta aggiungere alla procedura *Installa* codice come quello in figura 3, che restituisce al DOS la memoria occupata dall'environment. C'è però da notare che così ci si precluderebbe la possibilità di «comunicare» con il TSR mediante le variabili dell'environment; al più si potrebbe quindi prevedere un ulteriore parametro di *Installa*, di tipo *boolean*, attraverso il quale comunicare alla procedura se si desidera o meno che venga eseguito il codice della figura 3.

Strumenti ulteriori

Non voglio lasciare l'argomento dei programmi residenti senza segnalarvi un'altra perla del «solito» Kim Kokkonen. È disponibile su MC-Link un file TSRSRC.ZIP contenente alcuni programmi di utilità da lui realizzati; sono certo che troverete interessante in particolare una coppia di programmi chiamati MARK e RELEASE, che consente la disinstallazione di programmi residenti senza bisogno di resettare la macchina: va prima eseguito MARK, che rimane residente; tutti i TSR installati successivamente possono essere rimossi dalla memoria (con tanto di ripristino degli interrupt sostituiti) con RELEASE. Ho usato diverse volte MARK e RELEASE in questi ultimi mesi, e ho avuto così modo di constatarne la validità e l'utilità. Se poi pensate che in TSRSRC.ZIP ci sono anche i sorgenti...

Con ciò abbiamo terminato. Vi do quindi appuntamento tra trenta giorni per una nuova escursione: *exception handling* in Turbo Pascal. A presto.