di Sergio Polini

Ancora sugli errori critici

La volta scorsa abbiamo visto un esempio di una routine incorporata in un programma Turbo Pascal ma associata ad un interrupt; la routine non viene eseguita mediante la normale chiamata di una funzione o procedura, ma quando «scatta» l'interrupt. Essendo ben diverso il modo di attivazione, non bastano i meccanismi consueti cui il compilatore ricorre per preservare il contesto in cui la routine viene eseguita (BP salvato nello stack, SP parcheggiato in BP): è necessario tener conto del fatto che l'interrupt può partire in qualsiasi momento e quindi salvare tutti i registri del microprocessore.
Le indicazioni contenute a pag. 214 del manuale offrono una guida sufficiente per la maggior parte dei casi, e costituiscono comunque un buon punto di partenza; ora vedremo come partire di lì per disegnare una routine che venga eseguita quando viene generato un INT 24H

Perché sostituire la routine che il DOS ci propone per gestire gli «errori critici»? Abbiamo già accennato a qualche motivo la volta scorsa; possiamo aggiungere altre considerazioni.

Uno spiacevole effetto è di tipo meramente esteriore: quel messaggio «Annulla, Riprova, Ignora?» può comparire in qualsiasi punto dello schermo, magari distruggendo una maschera per l'input di dati che avevamo costruito con tanta pazienza. Un ben più grave danno deriva dal modo in cui il programma in esecuzione termina se si sceglie «Annulla»: viene infatti attivato in INT 23H (quello del Ctrl-C), che provoca un ritorno al DOS senza la chiusura dei file aperti e l'aggiornamento della directory (come invece fa la funzione 4CH dell'INT 21H). Il motivo è facilmente individuabile: se l'errore consistesse nella impossibilità di scrivere sul disco, tentare di chiudere i file non servirebbe a niente; si potrebbe, è vero, rinunciare a chiudere i file sul disco danneggiato e chiudere invece quelli che si trovassero su un altro disco in un altro drive, ma il DOS non può farsi carico di possibili dettagli come questo, dipendenti esclusivamente da una particolare configurazione hardware e da una particolare organizzazione dei file. Ancora: si può prevedere che un programma invii dati alla stampante solo se presente; se questa fosse spenta o senza carta l'utente potrebbe rimediare, altrimenti il programma potrebbe prendere atto della mancanza della periferica e limitarsi all'output su video. Ma questo può farlo solo un programma «ad hoc», un programma che possa permettersi di rinunciare eventualmente alla stampante. Non potrebbe fare lo stesso un programma di comunicazione che riscontrasse problemi sulla porta seriale.

L'unica soluzione è una routine «personalizzata», diversa per ogni programma. Quello di cui abbiamo bisogno, quindi, è uno schema generale e flessibile, adattabile a singole esigenze.

Scambio di informazioni

Il nostro obiettivo deve essere quello di scrivere il meno possibile in Assembler. Abbiamo visto il mese scorso che sono necessari degli «inline statement» per salvare e poi ripristinare i registri del microprocessore, per riabilitare gli interrupt e per far terminare la routine con un IRET invece che con il normale RET. È però possibile che la routine che assoceremo all'INT 24H chiami una normale procedura Pascal, in modo da poter gestire con maggiore comodità e flessibilità le diverse situazioni possibili. Quello che non possiamo evitare è la decodifica delle informazioni circa il tipo di errore, trasmesse dal DOS alla routine associata all'INT 24H mediante alcuni registri.

Se l'errore è stato causato da un disco, il bit più a sinistra di AH è zero; in questo caso in AL c'è il numero del drive (0 per A, 1 per B, e così via) e altri bit di AH contengono informazioni sul tipo d'errore (lettura o scrittura, area del disco interessata). Se quel bit è invece 1, bisogna leggere la word puntata da BP:SI+4: se è settato il bit più a sinistra di questa l'errore è stato causato da un dispositivo a caratteri (la stampante, la tastiera, il video, una porta seriale) il cui nome si trova in BP:SP+10. Se quel bit è 0 è danneggiata la copia in memoria della File Allocation Table. Il codice d'errore è in DI (negli otto bit di destra).

La mia proposta è di passare tutte queste informazioni ad una funzione ErroreCritico di tipo integer, che potrà fare (quasi) tutto quello che volete. L'importante è che il risultato sia poi 0, 1, o 2 (o anche 3 se si usa una delle versioni più recenti del DOS). Già sappiamo che una funzione di tipo integer ritorna il suo risultato in AX, e in effetti una routine associata all'INT 24H dovrebbe terminare dopo aver messo in AL uno di quei valori: 0 se l'utente ha scelto «Ignora», 1 per «Riprova», 2 per «Annulare», 3

Listato 1

```
{ INT24H.INC }

const
  DatSeg: integer = 0;
procedure Int24h;
begin
  inline(
    $53/$51/$52/$56/$57/$1E/$06/$FB/
    { come manuale p. 214, ma senza PUSH AX ($50)}
    $2E/$8E/$1E/DatSeg/       { mov     ds,cs:DatSeg  }
    $89/$C3/                  { mov     bx,ax     :info su errore disco in bx  }
    $25/$00/$80/              { and     ax,8000h  :se settato primo bit a sin. }
    $74/$0D/                  { jz      disk      :  non e' un errore di disco }
    $8E/$46/$00/              { mov     es,[bp]   :seg. device driver header}
    $26/$F7/$44/$04/$00/$80/  { test    word ptr es:[si+4],8000h }
    $75/$01/                  { jnz     char          }
    $40/                 { fat:  inc     ax        :se errore FAT  ax = 3 }
    $40/                 { char: inc     ax        :se errore char ax = 2 }
    $40/                 { disk: inc     ax        :se errore disk ax = 1 }
    $25/$FF/$7F/              { and     ax,7FFFh  :azzera il primo bit a sin.}
    $83/$EC/$02/              { sub     sp,2      :chiama ErroreCritico }
    $50/                      { push    ax            }
    $53/                      { push    bx            }
    $81/$E7/$FF/$00/          { and     di,00FFh  :azzera il byte a sin.}
    $57/                      { push    di            }
    $06/                      { push    es            }
    $83/$C6/$0A/              { add     si,10         }
    $56/                      { push    si            }
    $E8/ErroreCritico-*-2/    { call    ErroreCritico }
    $07/$1F/$5F/$5E/$5A/$59/$5B/$8B/$E5/$5D/$CF);
    { come manuale p. 214, ma senza POP AX ($58)}
end;
procedure Install24;
begin
  DatSeg := DSeg;
  reg.ds := CSeg;  reg.dx := ofs(Int24h);  { DS:DX := seg:ofs di Int24h }
  reg.ax := $2524; MsDos(reg)
end;
```

per «Tralascia» (se la versione del DOS lo riconosce).

Il file INT24H.INC (listato n. 1) contiene la definizione di una costante tipizzata DatSeg e di due procedure. Install24 assegna a DatSeg il valore di DS (fornito dalla funzione predefinita DSeg) e quindi associa all'INT 24H la procedura Int24h. Potrebbe anche salvare l'indirizzo della routine originaria per poi ripristinarla alla fine, come abbiamo fatto il mese scorso per l'INT 5. Fate voi: non è strettamente necessario perché il DOS rimette le cose a posto da solo alla fine del programma; la routine originariamente associata all'INT 24H andrebbe tuttavia ripristinata se si intendesse far girare il programma rimanendo nel Turbo Pascal, perché in tal caso alla fine il controllo ritornerebbe al compilatore invece che al DOS. In altri termini, se ne può fare a meno solo se il programma viene compilato su disco.

La procedura Int24h

Il corpo della procedura è tutto costituito da un «inline statement». All'inizio si salvano i registri e si riassegna a DS il valore del data segment del programma, come avevamo già visto la volta scorsa. L'unica differenza è che non si salva nello stack anche AX, mancano cioè sia il PUSH AX all'inizio che il POP AX alla fine. Quando l'interrupt ritorna, infatti, AL deve contenere un numero da 0 a 2 (o 3, secondo la versione del DOS): questo non potrebbe accadere se si ripristinasse il valore che AX aveva prima dell'esecuzione della routine.

Dopo il «prologo», vengono salvate in BX tutte le informazioni contenute in AX (se non è un errore di disco in BX andranno a finire valori senza senso, ma non importa) e si determina il tipo di errore, esaminando eventualmente la word all'indirizzo BP:SI+4 (notate che, a causa delle ormai familiari istruzioni PUSH BP e MOV BP, SP con cui comincia ogni procedura Turbo Pascal, il BP che ci serve è nello stack all'indirizzo contenuto in BP: è per questo che usiamo MOV ES,[BP] invece di MOV ES,BP). Se si tratta di un errore di disco si assegna ad AX il valore 1, 2 se si tratta di un dispositivo a caratteri, 3 se è danneggiata la copia interna della FAT.

Quindi si chiama la funzione ErroreCritico: si fa spazio nello stack per il risultato con SUB SP,2, e poi si «pushano» i tre parametri: la fonte dell'errore (in AX), le informazioni relative all'eventuale errore di disco (in BX), il codice d'errore (in DI) e l'indirizzo BP:SI+10, dove ErroreCritico potrà trovare il nome del dispositivo a caratteri che ha dato problemi. Per effettuare la «chiamata» vera e propria si possono seguire varie strade. Io ho messo un "EXTRN ErroreCritico:NEAR" e poi "CALL ErroreCritico" nel file ASM, ottenendo così un "E8 0000 E" nel file LST. In ogni caso nell'«inline statement» bisogna usare il codice esadecimale E8 (CALL) seguito dall'indirizzo *relativo* della funzione da chiamare. Indirizzo relativo vuol dire «differenza tra la locazione nel code segment della funzione da chiamare e la locazione in cui si trova la CALL»; il Turbo Pascal traduce il nome della funzione nel primo indirizzo e un asterisco nel secondo (cfr. pag. 212 del manuale), ma perché tutto funzioni occorre sottrarre un 2. Di qui la codifica:

$ E8/ ErroreCritico - * - 2 /

La funzione ErroreCritico

Volendo giocare un po' con le parole, potremmo dire che la procedura Int24h rappresenta la parte «costante» del nostro interrupt handler, la funzione ErroreCritico quella «variabile»: è in questa infatti che possiamo prenderci cura dei dettagli operativi del nostro programma; potremmo ad esempio settare una variabile globale di tipo boolean per rinunciare all'uso della stampante se questa risulta mancante, secondo quanto suggerivamo.

Quello che non possiamo fare è chiamare (direttamente o tramite funzioni o procedure del Turbo Pascal) funzioni del DOS con numero superiore a OCH e poi tornare a Int24h: con ogni probabilità scopriremmo di aver alterato le zone di memoria che il DOS usa per i suoi fini; in altri termini, ci si può bloccare tutto.

Ecco quindi che definiamo una procedura WriteStr, che scrive una stringa a partire da un punto dato (GotoXY usa il BIOS) servendosi della funzione 9. Ciò può sembrare non necessario, in quanto la Write del Turbo Pascal 3.01 usa la funzione 5, ma un'altra versione potrebbe usarne una diversa. La funzione ErroreCritico chiama WriteStr per scrivere un messaggio su video, e poi usa le funzioni 7 e 6 del DOS per leggere un carattere da tastiera. Nel listato numero 2 il messaggio descrive il tipo di errore; in particolare, se il problema è stato causato da un dispositivo a caratteri, riceve nel parametro variabile Nome l'indirizzo di un array di 8 byte in cui il DOS ha scritto il nome del dispositivo e aggiunge questi 8 byte alla stringa «Errore su» con la procedura Move. Se poi si tratta di un errore di disco o di un dispositivo a caratteri, fornisce anche una descrizione più analitica dell'errore e propone quattro scelte. «Abbandona» «Ignora» e «Riprova» equivalgono alle «Annulla» «Ignora» e «Riprova» del DOS, e infatti provocano il ritorno a Int24h con il corrispondente valore numerico in AX.

«Esci» è la vera novità, in quanto provoca la fine del programma non con un INT 23H (come l'«Annulla» del DOS) ma o con un Halt (che usa la funzione 4CH) o con un reset generale (mediante INT 19H). La funzione 4CH provvede a trasferire sui file aperti il contenuto dei rispettivi

buffer, come tutte le funzioni del DOS che provocano la fine di un programma, ma causa anche la chiusura dei file e l'aggiornamento delle directory: esito al quale non vale proprio la pena di rinunciare se l'errore non è stato causato da un disco ma, ad esempio, dalla stampante.

Possiamo chiamare la procedura Halt, e quindi una funzione del DOS di numero superiore a OCH, solo perché subito dopo il programma termina; non possiamo invece chiamare una funzione «proibita» e poi tornare alla procedura Int24h e di qui alla funzione del DOS che era incappata nell'errore. Non possiamo, ad esempio, chiudere i file aperti e poi tornare a Int24h assegnando a Errore-Critico un qualche valore. Può capitare, tuttavia, che l'errore consista nella impossibilità di scrivere sul disco nel drive A mentre abbiamo altri file aperti sul drive C, come nel programma del listato numero 3: in casi come questo si possono sì chiudere i file su C, ma poi, se si prova a uscire dal programma con Halt, la funzione 4CH proverà nuovamente ad aggiornare i file sul drive A. Niente di grave: poiché la funzione 4CH rimette a posto l'INT 24H, comparirà il solito messaggio «Annulla, Riprova, Ignora?», si risponderà «A» e tutto finirà lì. Se si vuole comunque evitare a tutti i costi un epilogo del genere, si può sempre provocare un reset, come appunto nel listato numero 2.

Il programma ERRCRIT.PAS propone anche un esempio di possibile rinuncia all'output su stampante: invece di usare il consueto «writeln(lst,...)», si usa «writeln(OutDev,...)», dove OutDev è una variabile di tipo Text che viene assegnata al file «lst:». La funzione ErroreCritico controlla se l'errore è stato causato dalla stampante e, se l'utente ha risposto con «I» (per convincere il DOS che si vuole davvero ignorare l'errore bisogna premere «I» con una certa insistenza), assenga FALSE ad una variabile boolean StampanteOK.

Il programma controlla il valore di questa variabile subito prima del *secondo* invio alla stampante e, se trova FALSE, assegna OutDev a «con:»; in questo modo, tutte le istruzioni «writeln(OutDev,...)» provocheranno un out-put su video invece che alla stampante che non c'è.

Questi naturalmente sono solo esempi: ho cercato di tracciare un quadro quanto più completo possibile delle diverse soluzioni e dei trabocchetti in cui si può cadere, ma una «vera» funzione ErroreCritico può essere scritta solo in un vero programma.

Variazioni sul tema

La Borland e il Turbo User Group distribuiscono un programma di pubbli-

Listato 2

```
{ ERRCRIT.INC }

type
  string80 = string[80];
var
  reg: record case integer of
       1: (ax,bx,cx,dx,bp,si,di,ds,es,flags: integer);
       2: (al,ah,bl,bh,cl,ch,dl,dh: byte)
       end;
procedure writestr( st: string80; col,row: integer );
begin
  gotoxy(col,row);
  st := st + '$';        { carattere di fine stringa per la funzione 9 }
  reg.ah := 9;
  reg.ds := seg(st);
  reg.dx := ofs(st[1]); { st[1] per saltare il primo byte, dove e' }
  msdos(reg)            { registrata la lunghezza della stringa    }
end;
function ErroreCritico(Tipo,DskInfo,Codice:integer; var Nome): integer;
const
  ERRDSK = 1;  ERRCAR = 2;  ERRFAT = 3;
  IGNORE = 0;  RETRY  = 1;  ABORT  = 2;
  msg: string80 = '';
  Dispositivo: string[8] = '        '; { 8 spazi }
var
  Area, Drive: integer; ch: char;
begin
  case Tipo of
    ERRDSK: begin
              Drive := lo(DskInfo);
              Area  := (hi(DskInfo) and 6) shr 1;
              if (DskInfo and $0100) = 0 then msg := 'Errore di lettura'
              else msg := 'Errore di scrittura';
              msg := msg + ' sul drive ' + chr(Drive+65) + ' (';
              case Area of
                0: msg := msg + 'file di sistema):';
                1: msg := msg + 'FAT):';
                2: msg := msg + 'directory):';
                3: msg := msg + 'file utente):';
              end
            end;
    ERRCAR: begin
              move(Nome,Dispositivo[1],8);
              msg := 'Errore su ' + Dispositivo + ': ';
            end;
    ERRFAT: msg := 'Errore nella copia interna della FAT';
  end;
  writestr(msg,1,23);
  if Tipo <> ERRFAT then begin
    case Codice of
      $00: msg := 'protetto dalla scrittura';
      $01: msg := 'unita'' sconosciuta';
      $02: msg := 'non pronto';
      $03: msg := 'comando sconosciuto';
      $04: msg := 'errore nei dati (CRC)';
      $05: msg := 'lunghezza della struttura di richiesta errata';
      $06: msg := 'errore posizionamento testina';
      $07: msg := 'dispositivo sconosciuto';
      $08: msg := 'settore non trovato';
      $09: msg := 'senza carta';
      $0A: msg := 'errore di scrittura';
      $0B: msg := 'errore di lettura';
      $0C: msg := 'errore generico';
    end;
    writestr(msg,1,24);
    writestr('Abbandona, Esci, Ignora, Riprova (A/E/I/R)? ',1,25);
    repeat
      reg.ah := 7; MsDos(reg);                    { legge un carattere }
      ch := upcase(chr(reg.al))
    until ch in ['A','E','I','R'];
    reg.ah := 6; reg.dl := byte(ch); MsDos(reg);  { lo mostra su video }
    if ch = 'A' then ErroreCritico := ABORT
    else if ch = 'E' then begin
      if Tipo = ERRCAR then Halt(1)
      else if Drive = 0 then begin   { se non va il drive A: }
        close(f1);                   { chiudi il file sul drive C: }
        inline($CD/$19)              { reset }
      end
    end
    else if (ch = 'I') then begin
      if copy(Dispositivo,1,3) = 'PRN' then StampanteOK := FALSE;
      ErroreCritico := IGNORE
    end
    else ErroreCritico := RETRY;
  end
  else
    ErroreCritico := ABORT;
end;
```

Listato 3

```
{ ERRCRIT.PAS }

Program EsempioGestioneErroriCritici;
var
  f1,f2,OutDev: text;
  StampanteOK : boolean;
{$I ERRCRIT.INC}
{$I INT24H.INC}
begin
  Install24;
  clrscr; writeln('Prova apertura file');
  writeln('Apri il drive A: e poi premi <RETURN>'); readln;
  writeln('Quando compare il messaggio d''errore');
  writeln('richiudi il drive A: e premi "R"');
  assign(f1,'c:pluto'); rewrite(f1);
  assign(f2,'a:pippo'); rewrite(f2);
  close(f1); close(f2);
  clrscr; writeln('Prova scrittura su stampante');
  StampanteOK := TRUE;
  assign(OutDev,'lst:'); reset(OutDev);
  writeln('Spegni la stampante e poi premi <RETURN>'); readln;
  writeln('Quando compare il messaggio d''errore premi ripetutamente "I"');
  writeln(OutDev,'Invio alla stampante');
  writeln;
  if not StampanteOK then begin
    assign(OutDev,'con:'); reset(OutDev)
  end;
  writeln(OutDev,'Invio al video, visto che la stampante non e'' pronta');
  writeln('Premi <RETURN> per uscire dal programma'); readln
end.
```

co dominio elaborato per la prima volta da Marshall Brain (noto a chi legga le TUG Lines o il Micro/Systems Journal) in cui si segue un approccio completamente diverso. La routine che intercetta l'INT 24H provoca il ritorno al programma Turbo Pascal con un adeguato codice d'errore (alcune funzioni del DOS usano OFFH, altre 01, altre settano il carry flag...).

Se si disabilita la direttiva «I», dopo ogni istruzione di I/O si può chiamare una funzione INT24Result al posto della funzione predefinita IOResult e ottenerne un codice «composito», in cui cioè sono incorporati sia il codice che si sarebbe ottenuto chiamando IOResult che quello messo dal DOS nel registro DI nel caso di «errore critico». Naturalmente INT24Result ritorna zero se non vi sono stati errori. Chi fosse interessato, può trovare su MC-Link sia un programma ERRCRIT.PAS (un unico file che comprende i tre listati qui pubblicati) che il programma di Marshall Brain col nome INTINT24.PAS (ovvero: INTercettazione dell'INTerrupt 24h). MC