Uno spooler di stampa

Vi ho già riferito altre volte di messaggi pervenutimi da alcuni di voi tramite Internet, via MC-link. Ora desidero raccontarvi di Massimo Penna, di Genova, e del problema che mi ha sottoposto: far comunicare un programma compilato per il modo protetto con un programma residente. In particolare, Massimo non riusciva a trasmettere al TSR l'indirizzo di una stringa. Dopo avergli indicato una soluzione e averne esaminato con lui tutti gli aspetti e le implicazioni, ho riprodotto il dialogo nell'area Pascal di MC-link. Ne è seguita una discussione interessante, in cui si sono segnalati i suggerimenti di Ottavio Risolia e la proposta di applicare la soluzione alla funzione 01h dell'INT 2Fh, che devo a Marco Cirinei: si tratta della funzione che permette di inviare comandi allo spooler di stampa installato dal comando PRINT del DOS

di Sergio Polini

Il programma di Massimo, compilato per il modo protetto, doveva comunicare con un TSR attraverso una API costituita da chiamate all'INT 65h, con una serie di valori passati nei registri. Il tutto funzionava a dovere, tranne nel caso in cui nei registri DX:BX si dovevano porre il segmento e l'offset di una stringa.

Massimo aveva individuato bene la natura del problema: usando la funzione *Seg* del Pascal per ottenere il segmento della stringa, quello che in realtà veniva reso come risultato non era un segmento, ma un selettore, che il TSR, in quanto normale programma DOS operante in modo reale, tendeva invece a interpretare come segmento.

La soluzione da lui trovata era decisamente «disperata»: dato che il Pascal 7 fornisce quattro selettori predichiarati (*Seg0040* per l'area dati del BIOS, *SegA000* per la memoria grafica delle schede EGA e VGA, *SegB000* e *SegB800* per la memoria video delle schede monocromatica e CGA), aveva copiato la stringa nell'area dati del BIOS e passato poi al TSR il segmento 0040h.

Un approccio da lui stesso giudicato «esteticamente orribile», ma che aveva in sé qualcosa di corretto: non sarebbe stato opportuno tentare di ottenere l'indirizzo fisico corrispondente al selettore reso dalla funzione *Seg*, in quanto non si sarebbe avuta alcuna garanzia circa la collocazione di tale indirizzo nell'ambito del primo megabyte di memoria: un indirizzo oltre tale limite non sarebbe risultato accessibile, infatti, da un TSR operante in modo reale. Era necessario copiare la stringa in una zona di memoria compresa nel primo megabyte; come abbiamo visto il mese scorso, la funzione 0100h dell'INT 31h consente appunto di allocare memoria nel primo megabyte, senza bisogno di usare i selettori predichiarati – per tutt'altri scopi – dal Pascal. La funzione ritorna sia il segmento che il selettore del blocco allocato, consentendo di utilizzare il selettore per la copia della stringa durante l'esecuzione in modo protetto e il segmento per passarne l'indirizzo «reale» al TSR.

L'interrupt 2Fh

Marco ha trovato il modo di generalizzare il problema.

Nel caso di Massimo, infatti, non vi

```
Chiamata:  AH     = 01h
           AL     = 00h : verifica l'intallazione dello spooler
                    01h : invia un file allo spooler
                    02h : rimuove un file dalla coda di stampa
                    03h : rimuove tutti i file dalla coda di stampa
                    04h : sospende la stampa per una lettura di stato
                    05h : riprende la stampa
           DS:DX = se AL=01h, segmento:offset di un packet, cioè
                               di una struttura di 5 byte:
                               un "livello" (che deve essere zero)
                               e segmento:offset di un nome di file
                   se AL=02h, segmento:offset di un nome di file

Risultato: se non vi sono errori:
             flag carry non settato
             se chiamato con AL=00h:
               AL = 00h : spooler non installato, ma installabile
                    01h : spooler non installato e non installabile
                    FFh : spooler installato
             se chiamato con AL=04h:
               DX    = numero di errori
               DS:SI = segmento:offset di una lista dei nomi dei file
                       presenti nella coda di stampa, rappresentati
                       mediante stringhe lunghe 64 byte; la lista
                       termina con una stringa nulla (un byte 00h)
           in caso di errore:
             flag carry settato
             AX = codice d'errore
```

Figura 1 - L'interfaccia della funzione 01h dell'INT 2Fh. Tutti i nomi di file sono stringhe ASCIIZ, cioè senza byte iniziale di lunghezza, ma terminate da un byte 00h.

erano problemi nella chiamata dell'INT 65h, ma Marco ha provato ad usare l'INT 2Fh per verificare la presenza dello spooler di stampa che si installa con il comando PRINT del DOS.

Lo spooler contenuto in PRINT.EXE è disponibile fino dal DOS 2.0, ma l'interfaccia tra i programmi applicativi e lo spooler venne documentata solo con la versione 3.0.

La funzione 01h dell'INT 2Fh comprende cinque sottofunzioni, riepilogate nella figura 1.

Le sottofunzioni 01h e 02h richiedono il passaggio, nei registri DS:DX, di una coppia segmento:offset, che già sappiamo come trattare. Abbiamo visto il mese scorso, tuttavia, che quando si chiama un interrupt da un programma compilato in modo protetto l'interrupt non viene eseguito direttamente, ma emulato dal DOS Extender; quando il programma viene eseguito in una sessione DOS di Windows in modo avanzato, inoltre, l'interrupt viene intercettato ed emulato anche dal *V86 monitor* di cui abbiamo parlato nel numero di dicembre.

Ne possono risultare conseguenze inaspettate.

Piuttosto che parlare in astratto di cosa può accadere, sarà forse meglio illustrare una unit che consenta di usare lo spooler di stampa in un programma Pascal, verificarne in concreto la funzionalità sia per il modo reale che per il modo protetto, per poi trovare una soluzione ai problemi che incontreremo. Qualcuno di voi potrà trovare comunque utile la unit per programmi compilati per il modo reale.

L'interfaccia della unit (figura 2) ricalca fedelmente le sottofunzioni della funzione 01h dell'interrupt 2Fh. Il tipo principale è rappresentato dalla classe *TSpooler*; il constructor verifica la presenza dello spooler mediante la sottofunzione 00h, mentre i metodi *Submit, HoldForStatus, Remove, CancelAll* e *ReleaseOld* corrispondono, rispettivamente, alle sottofunzioni 01h, 04h, 02h, 03h e 05h.

Gli altri tipi servono a gestire il risultato della sottofunzione 04h; questa rende in DS:SI l'indirizzo di una lista di nomi di file, rappresentati da stringhe ASCIIZ (senza byte iniziale di lunghezza, ma con byte 00h finale).

Con il Pascal 7 sono disponibili anche sotto DOS, mediante la unit *Strings*, le stringhe ASCIIZ prima riservate alla compilazione di applicazioni Windows; come tale, quindi, viene dichiarato il tipo *TFileName*, mentre si prevede una classe *TFileList* in cui copiare la lista dei nomi di file presenti nella coda di stampa.

```
implementation

uses Dos;

type
  PPacket = ^TPacket;
  TPacket = record
    Level: Byte;
    Path: PFileName;
  end;

procedure TFileList.FreeItem(Item: Pointer);
begin
  StrDispose(PChar(Item));
end;

constructor TSpooler.Init;
var
  Reg: Registers;
begin
  Err := 0;
  Reg.AX := $0100;
  Intr($2F, Reg);
  if ((Reg.Flags and fCarry) <> 0) or (Reg.AL <> $FF) then
    Fail
  else begin
    Queue := New(PFileList, Init(10, 5));
    if Queue = nil then
      Err := ERRNOLIST;
  end;
end;

destructor TSpooler.Done;
begin
  if Queue <> nil then
    Dispose(Queue, Done);
end;

procedure TSpooler.Submit(FileName: TFileName);
var
  P: TPacket;
  Reg: Registers;
begin
  if Err <> 0 then Exit;
  P.Level := 0;
  P.Path := @FileName;
  Reg.AX := $0101;
  Reg.DS := Seg(P);
  Reg.DX := Ofs(P);
  Intr($2F, Reg);
  if (Reg.Flags and fCarry) <> 0 then
    Err := Reg.AX;
end;

procedure TSpooler.HoldForStatus;
var
  Reg: Registers;
  P: PChar;
  F: TFileName;
begin
  if Err <> 0 then Exit;
  Reg.AX := $0104;
  Intr($2F, Reg);
  if (Reg.Flags and fCarry) <> 0 then
    Err := Reg.AX
  else if Queue <> nil then begin
    Queue^.FreeAll;
    P := Ptr(Reg.DS, Reg.SI);
    while P^ <> #0 do begin
      Queue^.Insert(StrNew(P));
      P := P + 64;
    end;
  end;
end;
```

La classe *TFileList* viene derivata da *TCollection* invece che da *TStrCollection*, per conservare l'ordine della lista originaria (*TStrCollection* deriva a sua volta da *TSortedCollection*); va comunque ridefinito il metodo *FreeItem*, in quanto quello di *TCollection* assume che i singoli elementi siano istanze di classi derivate da *TObject* e, quindi, usa la sintassi estesa della procedura *Dispose*, quella che vuole come secondo parametro il destructor.

La unit per il modo reale

L'implementazione della unit PRNSPOOL (figura 3) inizia con la dichiarazione di un tipo *TPacket* e del metodo *FreeItem* della classe *TFileList*, cui seguono i metodi di *TSpooler*.

```
function TSpooler.FirstInQueue: PChar;
begin
  if (Err = 0) and (Queue <> nil) and (Queue^.Count >= 1) then begin
    FirstInQueue := Queue^.At(0);
    Next := 1;
  end
  else
    FirstInQueue := nil
end;

function TSpooler.NextInQueue: PChar;
begin
  if (Err = 0) and (Queue <> nil) and (Next < Queue^.Count) then begin
    NextInQueue := Queue^.At(Next);
    Inc(Next);
  end
  else
    NextInQueue := nil
end;

procedure TSpooler.Remove(FileName: TFileName);
var
  Reg: Registers;
begin
  if Err <> 0 then Exit;
  Reg.AX := $0102;
  Reg.DS := Seg(FileName);
  Reg.DX := Ofs(FileName);
  Intr($2F, Reg);
  if (Reg.Flags and fCarry) <> 0 then
    Err := Reg.AX;
end;
procedure TSpooler.CancelAll;
var
  Reg: Registers;
begin
  if Err <> 0 then Exit;
  Reg.AX := $0103;
  Intr($2F, Reg);
  if (Reg.Flags and fCarry) <> 0 then
    Err := Reg.AX;
end;

procedure TSpooler.ReleaseHold;
var
  Reg: Registers;
begin
  if Err <> 0 then Exit;
  Reg.AX := $0105;
  Intr($2F, Reg);
  if (Reg.Flags and fCarry) <> 0 then
    Err := Reg.AX;
end;

function TSpooler.Error: Integer;
begin
  Error := Err;
  Err := 0;
end;

function HeapFunc(Size: Word): Integer; far;
begin
  HeapFunc := 1;
end;

begin
  HeapError := @HeapFunc;
end.
```

Figura 3 - L'implementazione della unit PRNSPOOL.

rata una funzione che gestisca gli errori di memoria insufficiente, come si fa in coda alla unit), il puntatore ad essa avrà valore **nil**; altrimenti (come vedremo nel programma di test), il constructor potrà essere utilizzato come una funzione *Boolean* e ritornerà FALSE.

Se si accerta la presenza dello spooler, si procede ad inizializzare la variabile d'istanza privata *Queue*, destinata a ricevere la lista dei file presenti nella coda di stampa.

Se non vi è memoria sufficiente, la variabile d'istanza *Err* avrà valore -1, rappresentato dalla costante ERRNOLIST; in questo modo sarà agevole riconoscere la situazione e si potrà comunque usare lo spooler, rinunciando, però alla possibilità di esaminare la coda di stampa.

Per gli errori che possono verificarsi, ho scelto una strategia analoga a quella che il Pascal adotta per gli errori di I/O: un errore impedisce la prosecuzione delle operazioni (altre chiamate dei metodi di *TSpooler* semplicemente non fanno nulla: tutti iniziano con una istruzione che chiama *Exit* se *Err* è diversa da zero), fino a che non si chiami una funzione (il metodo *Error*) che rende il codice dell'errore e azzera di nuovo la variabile *Err*.

Il metodo *Submit* aggiunge un file alla coda di stampa, mentre il metodo *HoldForStatus*, utilizzando la sottofunzione 04h, sospende le operazioni di stampa per permettere di esaminare il

```
unit PrnSpool;

interface

uses Objects, Strings;

const
  ERRNOLIST = -1;

type

  PFileName = ^TFileName;
  TFileName = array[0..66] of Char;

  PFileList = ^TFileList;
  TFileList = object(TCollection)
    procedure FreeItem(Item: Pointer); virtual;
  end;

  PSpooler = ^TSpooler;
  TSpooler = object
    constructor Init;
    destructor Done; virtual;
    procedure Submit(FileName: TFileName);
    procedure HoldForStatus;
    function FirstInQueue: PChar;
    function NextInQueue: PChar;
    procedure Remove(FileName: TFileName);
    procedure CancelAll;
    procedure ReleaseHold;
    function  Error: Integer;
  private
    Err: Integer;
    Queue: PFileList;
    Next: Integer;
  end;
```

Figura 2
L'interfaccia della unit PRNSPOOL.

Il constructor utilizza la sottofunzione 00h della funzione 01h dell'interrupt 2Fh. Viene accettato come risultato utile solo FFh (spooler installato); in ogni altro caso l'esecuzione termina con una chiamata della procedura *Fail*, che permette di riconoscere la situazione di inizializzazione fallita in due modi: se l'istanza della classe viene allocata dinamicamente (e se si è dichia-

```
Program TestSpooler;
uses PrnSpool;
const
  MaxFiles = 3;
var
  Spooler: TSpooler;
  Error, i: Integer;
  P: PChar;
  FileName: array[1..MaxFiles] of TFileName;
begin
  if Spooler.Init then begin
    Writeln('Spooler installato.');
    for i := 1 to MaxFiles do begin
      Write(i, '^ file da stampare: '); Readln(FileName[i]);
    end;
    Writeln;
    Error := Spooler.Error;
    i := 1;
    while (i <= MaxFiles) and (Error = 0) do begin
      Spooler.Submit(FileName[i]);
      Error := Spooler.Error;
      if Error = 0 then
        Writeln(FileName[i], ' aggiunto a coda di stampa')
      else
        Writeln(FileName[i], ': errore');
      Inc(i);
    end;
    Writeln;
    if Error = 0 then begin
      Spooler.HoldForStatus;
      Error := Spooler.Error;
      if Error = 0 then begin
        P := Spooler.FirstInQueue;
        while P <> nil do begin
          Writeln(P, ' in coda');
          P := Spooler.NextInQueue;
        end;
      end;
      Writeln;
      Write('File da rimuovere: '); Readln(FileName[1]);
      Spooler.Remove(FileName[1]);
      Error := Spooler.Error;
      if Error = 0 then begin
        Write('File rimosso. Premi Invio per proseguire.');
        Readln;
      end
      else
        Writeln('File non rimosso: errore');

    Spooler.ReleaseHold;
    if Spooler.Error = 0 then
      Writeln('Stampa in corso...');
  end;
  Spooler.Done;
end
else
  Writeln('Spooler non installato.');
end.
```

▲
Figura 4 - Un breve programma di test per la unit PRNSPOOL.

Figura 5 - Un esempio di esecuzione del programma TSTSPOOL.EXE.
▼

```
C:\> tstspool
Spooler installato.
1^ file da stampare: PROVA1.TXT
2^ file da stampare: PROVA2.TXT
3^ file da stampare: PROVA3.TXT

PROVA1.TXT aggiunto a coda di stampa
PROVA2.TXT aggiunto a coda di stampa
PROVA3.TXT aggiunto a coda di stampa

PROVA1.TXT in coda
PROVA2.TXT in coda
PROVA3.TXT in coda

File da rimuovere: PROVA2.TXT
File rimosso. Premi invio per proseguire.
Stampa in corso...
```

contenuto della coda e di intervenire, eventualmente, su questa: dopo aver vuotato la lista contenuta nella variabile *Queue*, infatti, le si aggiungono i nomi di file – ognuno rappresentato mediante una stringa di 64 byte – il primo dei quali si trova all'indirizzo reso nei registri DS:SI.

I metodi *FirstInQueue* e *NextInQueue* permettono di scorrere l'elenco di nomi di file salvato in *Queue*, mediante la variabile d'istanza *Next*, inizializzata a 1 dal primo metodo, poi incrementata ad ogni chiamata del secondo.

I metodi *Remove* e *CancelAll* possono essere utilizzati, rispettivamente, per eliminare un solo file o tutti i file dalla coda di stampa; è evidente che andrebbero chiamati dopo *HoldForStatus* e prima di *ReleaseHold*, che annulla la sospensione delle operazioni e provoca così la ripresa delle stampe (potrebbe essere opportuno vuotare la lista *Queue* in *ReleaseHold*, invece che – o oltre che – in *HoldForStatus*, in quanto l'elenco dei file accodati non è più attendibile dopo un *ReleaseHold*; fate voi).

Un breve programma di test

Trovate in figura 4 un breve programma di test.

La variabile *Spooler* non viene allocata dinamicamente e, quindi, si può usare il constructor – grazie alla procedura *Fail* – come una funzione *Boolean* che ritorna TRUE se risulta correttamente installato lo spooler di stampa contenuto nel comando PRINT del DOS.

In questo caso, vengono chiesti i nomi di alcuni file da stampare, che vengono poi passati allo spooler mediante il metodo *Submit*.

Le operazioni vengono subito sospese chiamando *HoldForStatus*, al fine di proporre all'utente di esaminare l'elenco dei file in coda di stampa e di scegliere un file da eliminare dalla coda (da non stampare).

Il file da eliminare va digitato esattamente come si era fatto all'inizio: è importante, in particolare, non usare prima le minuscole e poi le maiuscole (a rigore, basterebbe convertire in caratteri tutti maiuscoli i nomi di file digitati dall'utente prima di usarli come parametri dei metodi di *TSpooler*, o meglio prevedere la conversione in seno agli stessi metodi).

Il programma conferma – se tutto è andato bene – la rimozione del file dalla coda, per poi dare inizio alle stampe mediante il metodo *ReleaseHold*.

La figura 5 mostra cosa può accadere eseguendo il programma, quando tutto va bene...

Se il programma viene compilato per il modo reale, funziona correttamente sia se eseguito sotto DOS «normale», sia se eseguito in una sessione DOS di Windows; in quest'ultimo caso, funziona correttamente sia se lo spooler è stato installato prima di Windows, sia se viene installato nella sessione DOS subito prima di eseguire il programma (non funziona se lo spooler è stato installato in un'altra sessione DOS, ma non si può pretendere troppo!).

Se il programma viene compilato

L'angolo della posta

Un lettore – Lucio Serra, di Pognano di Brisighella (RA) – mi ha mandato uno script per la modifica dei comandi dell'editor della IDE del Borland Pascal, mediante il compilatore di macro TEMC.

L'aspetto più interessante mi pare l'implementazione di comandi di tipo «sintattico»: con Ctrl-Q-I, ad esempio, si genera automaticamente lo scheletro di una istruzione **if**, con Ctrl-Q-W quello di una istruzione **with**, e così via.

Il tutto è anche ben documentato.

Non solo: Luciano, giustamente soddisfatto del suo lavoro, ha ritenuto simpatico mettere il suo script a disposizione di tutti. Ho quindi raccolto i suoi file in BPEDIT.ZIP, che potete trovare su MC-link, nell'area PAS-FILES.

Paolo Bonzini di Turate (CO), invece, mi chiede: «Si può chiamare un metodo virtuale da dentro un constructor? In altre parole, la VMT è inizializzata appena questo è chiamato o alla fine?».

A rigore, potrei limitarmi a ricordare che la risposta è già nel manuale. Si tratta, tuttavia, di sole due righe in centinaia e centinaia di pagine; il dubbio è quindi comprensibile e può essere utile anche ad altri precisare che un constructor *può* chiamare anche metodi virtuali.

Il constructor, infatti, provvede a salvare l'offset della VMT della classe cui appartiene l'istanza che inizializza, nell'ambito del codice che, generato automaticamente dal compilatore, viene eseguito prima del blocco di istruzioni scritte dal programmatore. La chiamata di un metodo virtuale, quindi, può anche essere la prima istruzione «visibile» di un constructor.

per il modo protetto, funziona correttamene se eseguito in una sessione DOS di Windows; se eseguito sotto DOS «normale», tuttavia, si limita a segnalare che lo spooler non è installato, anche se il comando PRINT del DOS è stato dato subito prima dell'esecuzione.

Se ne deve dedurre che l'interrupt 2Fh viene emulato da Windows meglio di quanto non sia dal DOS Extender fornito con il Pascal 7.0.

Per fortuna, come abbiamo visto il mese scorso, un DOS Extender conforme alle specifiche DPMI 0.9 deve comunque offrire la possibilità di emulare un interrupt previsto per il modo reale, mediante la funzione 0300h dell'interrupt 31h.

Il mese prossimo, quindi, riscriveremo l'implementazione della unit PRNSPOOL in modo da renderla usabile da programmi compilati per il modo protetto. MC

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.