di Sergio Polini

Il contesto di un programma

Terminiamo questo mese la presentazione della unit EXECSWAP. Negli ultimi mesi abbiamo accennato più volte alla funzione EXEC del DOS e a vari elementi di contorno. Gli argomenti che tratteremo ora ci consentiranno non solo di vedere da vicino come questa funziona e come va usata, ma anche di dare una impostazione un po' più ordinata al tutto: cercheremo infatti di capire quale è il «contesto» di un programma eseguito sotto MSDOS. Perché? Semplice: potrebbe esserci utile per quello che faremo a gennaio...

Prima di caricare un programma, il DOS crea e inizializza un'area di memoria chiamata *Program Segment Prefix* (PSP); si tratta di un residuo del vecchio CP/M ma, a differenza di tante altre caratteristiche del sistema capostipite, il PSP si è aggiornato al punto da rimanere componente essenziale del mondo MS-DOS. La figura 1 mostra in dettaglio la struttura del PSP come illustrata nella *MS-DOS Encyclopedia*, mentre il breve programma nella figura 2 ci aiuta a svelare qualche altro aspetto non documentato.

Interrupt e Environment

Abbiamo in primo luogo alcune cose di cui ormai non sapremmo cosa fare: sotto CP/M si usavano le istruzioni CALL 0000 e CALL 0005 rispettivamente per uscire da un programma e per chiamare le funzioni di sistema (il vecchio BDOS); qui non solo ritroviamo qualcosa di strettamente analogo agli offset 0 e 5 (si potrebbe usare una CALL PSP:0005 invece di INT 21h per le funzioni DOS da 0 a 24h), ma addirittura un INT 21h seguito da un RETF all'offset 50h, per chiamare anche le funzioni DOS oltre la 24h! Naturalmente nessuno si sognerebbe mai di procedere in questa maniera.

Più interessante, in teoria, l'informazione contenuta all'offset 2: l'indirizzo dell'ultimo segmento allocato al programma consentirebbe di modificare l'ampiezza dell'area allocata, ad esempio per restituire al DOS quella non necessaria. In realtà, come abbiamo visto ad ottobre, ciò si fa usando la funzione 4Ah dopo aver messo in ES l'indirizzo del PSP: si tratta, in altri termini, di una informazione a prevalente beneficio del DOS.

Noi utenti dobbiamo invece prestare particolare attenzione agli indirizzi degli interrupt 22h, 23h e 24h e a quello dell'environment. L'INT 22h è forse un caso a parte: contiene infatti l'indirizzo della routine che viene eseguita quando un programma termina, e non è frequente che venga alterato; sappiamo bene invece (ne abbiamo visto esempi anche in questa rubrica, maggio e giugno 1988) che un programma «serio» è praticamente obbligato ad intercettare gli interrupt 23h (Ctrl-C) e 24h (errori critici) se vuole costantemente mantenere il controllo degli eventi. Sappiamo anche quanto possa essere utile l'environment, ed abbiamo già accennato alla possibilità di chiamare un programma dopo avergli costruito un environment «su misura». Passare il controllo ad un programma comporta la necessità di impostare correttamente questi campi del PSP. All'environment pensa la stessa funzione EXEC (4Bh), dopo che le abbiamo passato in ES:BX l'indirizzo di un *parameter block* contenente appunto nei suoi primi due byte il segmento dell'environment (settembre scorso).

Quanto agli interrupt 22h, 23h e 24h, il DOS, prima di eseguire un programma, mette nel PSP di questo gli indirizzi delle routine a loro correntemente associati, in modo da poterli poi ripristinare quando il programma termina. Si considera infatti normale la redifinizione almeno degli interrupt 23h e 24h, ma si vuole evitare che questi rimangano «appesi» a codice che, in quanto collocato in un programma terminato, potrebbe provocare comportamenti anomali se attivato successivamente. Non ci sono problemi quando un programma è attivato dal COMMAND.COM, ma dobbiamo evitare che, nel caso di un programma attivato da un altro, quegli interrupt vadano ad eseguire codice appartenente al programma «padre», come succederebbe se questo li avesse ridefiniti; per far ciò abbiamo due soluzioni: o andiamo a scrivere direttamente nel PSP (da evitare!!!), o, più semplicemente, rimettiamo a posto gli interrupt prima di eseguire il programma «figlio». Il Turbo Pascal ci mette a disposizione una comoda procedura *SwapVectors* che vi provvede automaticamente (anche di questo avevamo parlato lo scorso settembre).

File Control Block e Disk Transfer Area

Il CP/M gestiva l'accesso ai file mediante i *File Control Block* (FCB); questi ricomparvero nell'MS-DOS soprattutto per agevolare la portabilità dal vecchio al nuovo ambiente di un gran numero di programmi, ma vennero soppiantati da nuovi meccanismi a partire dalla versione 2.0. Un FCB (prescindiamo ora dai FCB «estesi», che hanno in più un byte

Figura 1 - Struttura del Program Segment Prefix (fonte: The MS-DOS Encyclopedia, Microsoft Press. Altri testi, tra cui lo stesso Advanced MS-DOS di Ray Duncan, non offrono lo stesso grado di dettaglio; manca però anche nella Encyclopedia una informazione fondamentale: dove sono gli handle, cosa che potete scoprire con il programmino in figura 2).

Offset	Lunghezza	Contenuto
0000	2	INT 20h (CP/M: CALL 0000)
0002	2	Indirizzo dell'ultimo segmento allocato
0004	1	riservato
0005	5	Chiamate di funzione DOS (CP/M: CALL 0005)
000A	4	Indirizzo routine associata a INT 22h
000E	4	Indirizzo routine associata a INT 23h
0012	4	Indirizzo routine associata a INT 24h
0016	22	riservati
002C	2	Segmento dell'environment
002E	34	riservati
0050	3	INT 21h + RETF
0053	9	riservati
005C	16	Primo File Control Block
006C	16	Secondo File Control Block
007C	4	riservati
0080	128	Coda comando e DTA
	256	

come flag, cinque byte riservati e un byte per l'attributo del file) richiede 37 byte per contenere informazioni come il nome e la data del file, la lunghezza dei suoi record, il numero del record corrente, ecc. La principale limitazione è data dalla impossibilità di mettere in quei 37 byte un nome di file completo di path (un *pathname* può arrivare fino a 80 byte): ne segue che con i FCB si può sì lavorare su file presenti in dischi diversi, ma solo nella directory corrente di ogni disco.

C'è anche da dire che l'uso dei FCB è un po' bizzarro: nelle aree a loro destinate nel PSP vengono messi originariamente i primi due argomenti della riga comando, a condizione che questi possano essere letti come composti dalla indicazione di un drive seguita dai due punti, da un nome e da una estensione (la verifica viene effettuata dalla funzione 29h del DOS); se poi si apre un file usando il primo FCB del PSP, i 16 byte originari ovviamente non bastano più, e i 37 necessari vengono rubati al secondo FCB; se si usa invece questo, viene rubato lo spazio assegnato alla coda del comando (cioè agli argomenti della riga comando) e alla DTA. Conviene quindi riservare apposite aree del programma come FCB alternativi e passare l'indirizzo di questi alle varie funzioni del DOS «vecchio stile» (per la precisione, quelle da 0Fh a 17h, da 21h a 24h, da 27h a 29h).

Un po' macchinoso. E tuttavia quando un programma ne lancia un altro non può sapere come questo gestirà i suoi file: deve quindi preparargli due FCB «chiusi» (quelli di soli 16 byte) passando alla funzione 29h del DOS gli indirizzi sia di una stringa con gli argomenti della riga comando che di due aree di 16 byte. Gli indirizzi dei due FCB così creati vanno poi passati alla funzione EXEC (4Bh) perchè possa servirsene per creare il PSP del programma «figlio».

Analoga la situazione per la *Disk Tranfer Area* (DTA), l'area usata come buffer per le operazioni di lettura e scrittura di file attraverso le funzioni che usano i FCB. Gli ultimi 128 byte del PSP servono a due scopi: registrare gli argomenti della riga comando (che si suppone vengano letti prima di effettuare trasferimenti di dati da/a i file) e offrire un buffer per l'I/O con file aventi record non più lunghi di 128 byte. Un po' perché la DTA può venire occupata dal secondo FCB, un po' perché 128 byte non sempre sono sufficienti, un po' perché usando la DTA del PSP si perdono gli argomenti della riga comando, i programmi possono impostare un diverso buffer mediante la funzione 1Ah.

Si tratta comunque di un'area «vuota», di cui non deve preoccuparsi il programma che voglia eseguirne un altro: ciò che a quest'ultimo serve è invece, ovviamente, il contenuto iniziale di quei 128 byte, ovvero una stringa con gli argomenti della riga comando, da passare anch'essa alla funzione EXEC.

File handle

Con ciò potremmo anche aver terminato; riassumendo, la funzione EXEC (4Bh) va chiamata con l'indirizzo di una stringa contenente il pathname completo del programma da eseguire in DS:DX, ma prima bisogna rimettere a posto gli interrupt 22h, 23h e 24h, e si deve mettere in ES:BX l'indirizzo di un *parameter block* contenente a sua volta gli indirizzi di un *environment*, di una stringa con gli argomenti della riga comando, di due FCB; il contenuto del *parameter block* serve ovviamente a consentire a EXEC di costruire un PSP per il programma da eseguire. Questo è quello che l'utente può vedere.

In realtà c'è dell'altro. A partire dalla versione 2.0, la gestione dei file avviene preferibilmente con funzioni del DOS che nulla hanno a che vedere con i FCB: si usa invece una tabella contenuta dove solo COMMAND.COM sa, e la cui ampiezza è modificabile dall'utente mediante la familiare riga «FILES=xxx» nel CONFIG.SYS. Per default c'è spazio per le informazioni relative a otto file, di cui cinque riservati a *standard input* (CON), *standard output* (CON), *standard error* (CON), *standard auxiliary* (AUX) e *standard list* (PRN); ciò consente ad un programma di tenere aperti contempo-

Figura 2 - Un programma per spiare la struttura del Program Segment Prefix. Il programma mostra su video la struttura del PSP all'inizio e poi dopo ognuna di MAXFILES aperture di file; si può così scoprire dove sono gli handles: nell'area riservata di 22 byte all'offset 16h.

```
Program PSP;
uses Dos;
const
  MAXFILES = 3;
type
  Str2 = string[2];
var
  f: array[1..MAXFILES] of file;
  i: integer;
function Hex(b: byte): Str2;
  const
    HexDigit : string[16] = '0123456789ABCDEF';
  var
    s: Str2;
  begin
    s[2] := HexDigit[b mod 16 + 1];
    s[1] := HexDigit[b div 16 + 1];
    Hex := s
  end;
procedure DumpPSP;
  type
    PSP = array[0..255] of byte;
  var
    PSPPtr : ^PSP;
    i      : integer;
    CmdTail: string[127];
  begin
    PSPPtr := Ptr(PrefixSeg,0);
    for i := 0 to $5B do begin
      if i in [$0,$2,$4,$5,$A,$E,$12,$16,$2C,$2E,$50,$53] then begin
        Writeln;
        Write(Hex(i),': ')
      end;
      Write(Hex(PSPPtr^[i]))
    end;
    Writeln;
    Write('5C: ');
    if PSPPtr^[$5C] > 0 then begin
      Write(Chr(PSPPtr^[$5C] + Ord('@')),':');
      for i := $5D to $6B do
        Write(Chr(PSPPtr^[i]))
    end
    else
      Write('Primo FCB vuoto');
    Writeln;
    Write('6C: ');
    if PSPPtr^[$6C] > 0 then begin
      Write(Chr(PSPPtr^[$6C] + Ord('@')),':');
      for i := $6D to $7B do
        Write(Chr(PSPPtr^[i]))
    end
    else
      Write('Secondo FCB vuoto');
    Writeln;
    Write('7C: ');
    for i := $7D to $7F do
      Write(Hex(PSPPtr^[i]));
    Writeln;
    Write('80: ');
        if PSPPtr^[$80] > 0 then begin
          Move(PSPPtr^[128], CmdTail, 128);
          Writeln(CmdTail)
        end
        else
          Writeln('Coda comando vuota')
    end;
  begin
    DumpPSP;
    Readln;
    for i := 1 to MAXFILES do begin
      Assign(f[i], Hex(i));
      Rewrite(f[i]);
      DumpPSP;
      Readln
    end;
    for i := 1 to MAXFILES do begin
      Close(f[i]);
      DumpPSP;
      Readln;
      Erase(f[i])
    end
  end.
```

```
; function ExecWithSwap(Path, CmdLine: string): word;
ExecWithSwap    PROC FAR
        PUSH    BP
        MOV     BP, SP
        MOV     Status, 1               ; Assumi un insuccesso
; Copia le variabili nel Code Segment
        LES     DI, [BP+6]              ; ES:DI -> CmdLine
        MOV     CmdPtr.ofst, DI
        MOV     CmdPtr.segm, ES         ; CmdPtr -> CmdLine
        LES     DI, [BP+10]             ; ES:DI -> Path
        MOV     PathPtr.ofst, DI
        MOV     PathPtr.segm, ES        ; PathPtr -> Path
        MOV     SaveSP, SP
        MOV     SaveSS, SS
        MovMem  BytesSwappedCS.lo, BytesSwapped.lo
        MovMem  BytesSwappedCS.hi, BytesSwapped.hi
        MovMem  EmsHandleCS, EmsHandle
        MovMem  FrameSegCS, FrameSeg
        MovMem  FileHandleCS, FileHandle
        MovMem  PrefixSegCS, PrefixSeg
        InitSwapCount

; Vedi se devi 'swappare' su disco o su EMS
        CMP     EmsAllocated, 0
        JZ      NotEms
        JMP     WriteE
NotEms:
        CMP     FileAllocated, 0
        JNZ     WriteF
        JMP     ESDone

; Usa un file su disco
WriteF:
        MovSeg  DS, CS
        InitSwapFile
        JNC     EF0                     ; Salta se niente errori
        JMP     ESDone                  ; .. altrimenti esci
EF0:
        SetSwapCount FileBlockSize      ; CX := byte da salvare su disco
        MOV     DX, OFFSET FirstToSave  ; DS:DX -> inizio dell'area di
        DosCallAH 40h                   ; .. memoria da salvare
        JC      EF1                     ; Se errori, salta
        CMP     AX, CX                  ; Salvato tutto?
        JZ      EF2                     ; Se si', salta
EF1:
        JMP     ESDone
EF2:
        NextBlock DS, FileBlockSize     ; DS -> prossimo blocco da salvare
        JNZ     EF0                     ; Continua se ne rimangono
        MOV     UsedEms, 0              ; Flag := non usata l'EMS
        JMP     SwapDone

; Usa la memoria espansa
WriteE:
        MOV     ES, FrameSeg
        MOV     DX, EmsHandle
        XOR     BX, BX                  ; BX := 0 (prima pagina logica)
        MovSeg  DS, CS
EE0:
        XOR     AL, AL                  ; AL := 0 (pagina fisica 0)
        EmsCall 44h                     ; Mappa da pag. logica a fisica
        JZ      EE1                     ; Se tutto bene, salta
        JMP     ESDone                  ; .. altrimenti esci
EE1:
        SetSwapCount EmsPageSize        ; CX := byte da salvare
        XOR     DI, DI                  ; ES:DI -> base della pagina EMS
        MOV     SI, OFFSET FirstToSave  ; DS:SI -> inizio dell'area di
        MoveFast                        ; .. memoria da salvare
        INC     BX                      ; Prossima pagina logica
        NextBlock DS, EmsPageSize       ; DS -> prossimo blocco da salvare
        JNZ     EE0                     ; continua se ne rimangono
        MOV     UsedEms, 1              ; Flag := usata EMS

; Dealloca la memoria occupata dai dati 'swappati'
SwapDone:
        MOV     AX, PrefixSegCS
        MOV     ES, AX                  ; ES := Seg del PSP
        DEC     AX
        MOV     DS, AX                  ; DS := Seg del memory control block
        MOV     CX, DS:[0003h]          ; num. di paragrafi attuali
        MOV     ParasWeHave, CX         ; .. ricordato in ParasWeHave
        SetTempStack                    ; NB: assegna anche CS a BX
        MOV     AX, OFFSET FirstToSave + 15
        MOV     CL, 4
        SHR     AX, CL                  ; Converte l'OFFSET in paragrafi
        ADD     BX, AX
        SUB     BX, PrefixSegCS         ; BX := paragrafi da mantenere
        DosCallAH 4Ah
        JNC     EX0                     ; Se tutto bene, prosegui con EXEC
        JMP     EX5                     ; .. altrimenti salta

; Chiama la funzione EXEC del DOS
EX0:
        MOV     AX, ES:[002Ch]          ; AX := Seg dell'environment
        MOV     EnvironSeg, AX
        MovSeg  ES, CS
        LDS     SI, PathPtr             ; DS:SI -> Pathname da eseguire
        MOV     DI, OFFSET Path         ; ES:DI -> sua copia in ASCIIZ
        CLD
        LODSB                           ; AL := sua lunghezza
        CMP     AL, 63
        JB      EX1
        MOV     AL, 63                  ; .. troncata se > 63
EX1:
        MOV     CL, AL
        XOR     CH, CH                  ; CX := byte da copiare
        REP     MOVSB
        XOR     AL, AL
        STOSB                           ; Aggiungi uno zero
        LDS     SI, CmdPtr              ; DS:SI -> riga comando
        MOV     DI, OFFSET CmdLine      ; ES:DI -> sua copia
        LODSB                           ; AL := sua lunghezza
        CMP     AL, 126
        JB      EX2
        MOV     AL, 126                 ; .. troncata se > 126
EX2:
        STOSB
        MOV     CL, AL
        XOR     CH, CH
```

raneamente solo tre file, a meno di non intervenire appunto con il comando FILES, come ben sappiamo, per istruire il DOS affinché crei una tabella più ampia.

Nel PSP di un programma c'è poi una tabella di *handle*, cioè di indici per la tabella di sistema: ad ogni «indice» diverso da 0FFh (-1) corrisponde un componente di questa, e quindi un file aperto. Usare questa tabella, cioè usare le funzioni per la gestione dei file che lavorano con gli *handle*, vuol dire poter lavorare su file anche se non presenti nella directory corrente, ma anche disporre di ridirezione dell'I/O, poter operare in reti locali, ecc. In altri termini, questa tabella è elemento essenziale del «contesto» di un programma.

Domande: dove sta? come è fatta? La Microsoft, per quanto a me risulta, dice solo che risiede nel PSP, ovviamente in una delle aree riservate. Dato che sono curioso almeno quanto voi, vi ho preparato il programmino della figura 2: per prima cosa mostra il contenuto del PSP, diviso nei suoi diversi «campi»; i FCB e l'area destinata ad ospitare gli argomenti della riga comando e la DTA sono se possibile mostrati in chiaro, come stringhe. Ad esempio, se date il comando "PSP A:PIPPO B:PLUTO PAPERINO", potrete vedere "A:PIPPO" e "B:PLUTO" rispettivamente nel primo e nel secondo FCB, e tutti e tre gli argomenti nell'ultimo «campo».

Poi viene il bello. Premete RETURN e il programma prima apre un file poi mostra di nuovo il PSP, per tre volte; quindi torna indietro, chiudendo i file e mostrando ogni volta il PSP. In questo modo è facile accorgersi che la tabella degli handle si trova all'offset 16h. Qui si nota infatti una sequenza del tipo «0101010002» (evidentemente corrispondente a CON-CON-CON-AUX-PRN), seguita da 15 byte con valore 0FFh; ogni volta che si apre un file si vede un byte passare dal valore 0FFh ad un intero positivo, a partire da 03 (salvo configurazioni particolari, come ad esempio quella della mia macchina...); ogni volta che si chiude un file si vede un byte positivo ridiventare 0FFh. All'offset 16h corrisponde però un'area di 22 byte, gli ultimi 20 dei quali servono appunto ad ospitare i 20 *handle*, tanti quanti sono i file che un programma può tenere aperti contemporaneamente (se c'è almeno FILES=20 in CONFIG.SYS; il limite di 20 può essere superato con la funzione 67h del DOS, introdotta con la versione 3.3, che alloca dinamicamente una nuova tabella). I primi due byte contengono invece il segmento del PSP del programma che ha chiamato quello in esecuzione; ciò consente al DOS di risalire la catena dei PSP dei programmi attivi, fino ad arrivare ad un PSP in cui i due byte all'offset 16h sono uguali al segmento del PSP stesso: fino cioè ad arrivare al COMMAND.COM e quindi alla tabella di sistema.

ExecWithSwap

Ora siamo in grado di comprendere appieno il sorgente della funzione *ExecWithSwap*, ultima e fondamentale tessera del nostro mosaico. Scorrendo il listato nella figura 3 possiamo vedere che, dopo aver salvato nel *code segment* alcune variabili (in quanto il *data segment* verrà «swappato»), si verifica se si deve usare la memoria espansa o

```
        REP     MOVSB
        MOV     AL, 0Dh
        STOSB                           ; Aggiungi ^M
        MovSeg  DS, CS
        MOV     SI, OFFSET CmdLine
        MOV     CmdLinePtr.ofst, SI
        MOV     CmdLinePtr.segm, DS     ; CmdLinePtr := ptr a CmdLine
        INC     SI
        MOV     DI, OFFSET FileBlock1
        MOV     FilePtr1.ofst, DI
        MOV     FilePtr1.segm, ES       ; FilePtr1 := ptr al primo FCB
        DosCallAX 2901h
        MOV     DI, OFFSET FileBlock2
        MOV     FilePtr2.ofst, DI
        MOV     FilePtr2.segm, ES       ; FilePtr2 := ptr al secondo FCB
        DosCallAX 2901h
        MOV     DX, OFFSET Path
        MOV     BX, OFFSET EnvironSeg
        DosCallAX 4B00h                 ; Exec
        JC      EX3                     ; Se errori, salta
        XOR     AX, AX                  ; .. altrimenti azzera AX
EX3:
        MOV     Status, AX              ; Status := AX

; Imposta lo stack temporaneo e rialloca la memoria
        SetTempStack
        MOV     ES, PrefixSegCS
        MOV     BX, ParasWeHave
        DosCallAH 4Ah
        JNC     EX4                     ; Se tutto bene, salta
        HaltWithError 0FFh              ; .. altrimenti esci
EX4:
        InitSwapCount                   ; Inizializza LeftToSwap

; Vedi quale metodo di swap era stato usato
EX5:
        CMP     UsedEms, 0
        JZ      ReadF
        JMP     ReadE

; Riprendi i dati dal file di swap
ReadF:
        MovSeg  DS, CS
        InitSwapFile
        JNC     EF3
        HaltWithError 0FEh              ; Se errori, esci
EF3:
        SetSwapCount FileBlockSize      ; CX := byte da ripristinare
        MOV     DX, OFFSET FirstToSave  ; DS:DX -> inizio dell'area di
        DosCallAH 3Fh                   ; .. memoria da ripristinare
        JNC     EF4                     ; Se tutto bene, salta
        HaltWithError 0FEh              ; .. altrimenti esci
EF4:
        CMP     AX, CX                  ; AX := byte letti
        JZ      EF5                     ; Se letti quanti in CX, salta
        HaltWithError 0FEh              ; .. altrimenti errore: esci
EF5:
        NextBlock DS, FileBlockSize     ; DS -> Prossimo blocco da leggere
        JNZ     EF3                     ; Continua se ne rimangono
        JMP     ESDone                  ; .. altrimenti abbiamo finito
```

```
; Riprendi i dati dalla memoria espansa
ReadE:
        MOV     DS, FrameSegCS          ; DS -> page frame
        MOV     DX, EmsHandleCS         ; DX := handle del blocco EMS
        XOR     BX, BX                  ; BX := prima pagina logica
        MovSeg  ES, CS
EE3:
        XOR     AL, AL                  ; AL := pagina fisica 0
        EmsCall 44h                     ; Mappa da pag. logica a fisica
        JZ      EE4                     ; Se tutto bene, salta
        HaltWithError 0FDh              ; .. altrimenti esci
EE4:
        SetSwapCount EmsPageSize        ; CX := byte da ripristinare
        XOR     SI, SI                  ; DS:SI -> base della pagina EMS
        MOV     DI, OFFSET FirstToSave  ; ES:DI -> inizio dell'area di
        MoveFast                        ; .. memoria da ripristinare
        INC     BX                      ; Prossima pagina logica
        NextBlock ES, EmsPageSize       ; ES -> prossimo blocco da muovere
        JNZ     EE3                     ; Continua se ne rimangono

ESDone:
        CLI
        MOV     SS, SaveSS
        MOV     SP, SaveSP
        STI
        MOV     AX, SEG DATA
        MOV     DS, AX
        MOV     AX, Status
        POP     BP
        RET     8

ExecWithSwap    ENDP
```

Figura 3
La procedura ExecWithSwap.

un file su disco. Quindi, dopo aver parcheggiato nel luogo opportuno tutta la parte del programma successiva alla label *FirstToSave*, si usa la funzione 4Ah per liberare la memoria che ne era occupata. A questo punto intervengono tutti i preliminari di cui abbiamo parlato e finalmente la chiamata della funzione EXEC.

Terminata l'esecuzione del programma figlio si imposta uno stack temporaneo per poter ripristinare la situazione precedente: si rialloca la memoria prima occupata dal programma «padre» e se ne ripristina il contenuto traendolo, secondo il caso, dalla memoria espansa o dal file di *swap*.

Per i movimenti di dati da memoria convenzionale a memoria estesa e viceversa viene prima chiamata la funzione 44h dell'INT 67h, che associa un'area della memoria espansa a una delle quattro «pagine» di 16K della *page frame* descritta il mese scorso. Ciò consente di copiare i dati, ma solo 16K per volta. Matteo Codazzi, un attento utente di MC-Link, ha subito rilevato che si potrebbe utilizzare una soluzione più comoda: la funzione 57h dell'INT 67h (illustrata nella figura 4) consente infatti di spostare in una volta sola fino ad un massimo di un mega. Si tratta però di una funzione introdotta con la versione 4.0 della LIM EMS, e quindi, in teoria, occorrerebbe prima verificare la versione della EMS presente (si può farlo con la funzione 46h dell'INT 67h, che ritorna in AL il numero di versione in formato BCD: cioè un 40h per la 4.0).

AX = 5700h
DS:SI = indirizzo di un blocco così strutturato:

Offset	Lunghezza	Contenuto
00	4	lunghezza in byte del blocco da muovere
04	1	tipo origine (1 per EMS, altrimenti 0)
05	2	handle della EMS (altrimenti 00)
07	2	offset dell'origine
09	2	segmento dell'origine o numero pagina logica
0B	1	tipo destinazione (1 per EMS, altrimenti 0)
0C	2	handle della EMS (altrimenti 00)
0E	2	offset della destinazione
10	2	segmento destinazione o numero pagina logica

Figura 4 - Come si chiama la funzione 57h dell'INT 67h per spostare un blocco di memoria ampio fino a un mega da un'area in memoria convenzionale o espansa ad un'altra area in memoria convenzionale o espansa.

Conclusione

Con ciò termina l'illustrazione della unit EXECSWAP di Kim Kokkonen: una soluzione efficiente dei problemi della procedura Exec del Turbo Pascal, un gioiellino da includere in ogni programma che voglia consentire ai propri utenti una intelligente interfaccia con il DOS.

Per poterne illustrare compiutamente il funzionamento abbiamo dovuto abbandonare il puro Pascal per scendere fino ai più nascosti meandri del DOS. Ne abbiamo guadagnato la comprensione di meccanismi che, grazie alla versatilità e alla potenza delle ultime versioni del compilatore, ci permetteranno di realizzare programmi ancora più arditi.