di Sergio Polini

Usando la memoria espansa

La volta scorsa abbiamo iniziato a vedere una unit EXECSWAP, che consente di sostituire la procedura Exec del Turbo Pascal con una funzione ExecWithSwap. Questa ci libera dalla necessità di determinare a priori le dimensioni dell'eventuale heap dei nostri programmi (con la direttiva $M o mediante il menu Options dell'ambiente integrato) e, come se non bastasse, ci consente di far eseguire da un programma anche altri programmi «ingombranti». Il tutto mediante temporaneo parcheggio su memoria espansa o su disco del programma da cui viene chiamata la funzione. In questo e nel prossimo articolo ne vedremo in dettaglio il funzionamento, esaminando anche alcune alternative di implementazione

Tanto per darvi subito un'idea delle prestazioni di EXECSWAP, vi propongo nella figura 1 un breve demo. Anche TESTEXEC, come la unit EXECSWAP.PAS e il file EXECSWAP.ASM, è stato scritto da Kim Kokkonen, presidente della Turbo Power Software, e qui riprodotto con la sua cortese autorizzazione. Non me la sono sentita di ricambiare tanta cortesia con modifiche al suo lavoro, che quindi vi ripropongo in versione strettamente originale. Mi sono solo concesso la libertà di aggiungere un array di 30.000 interi proprio a TESTEXEC, in modo da dare maggiore concretezza al demo: è ben probabile, infatti, che la funzione *ExecWithSwap* troverà la sua più naturale collocazione in programmi di ben altro ingombro che un breve demo.

Bene, veniamo alle cifre. Compilando TESTEXEC dopo aver disattivato le varie opzioni di debugging, si ottiene un eseguibile di appena 5.472 byte. Programmi di utilità come l'EXEMOD della Microsoft, tuttavia, ci avvertono che il programma richiederà un minimo di 82.352 byte per girare: appena partito, infatti, il programma dovrà crearsi uno stack di 16.384 byte e reclamerà altri 60.000 byte per quell'array di interi, più altri spiccioli per i restanti suoi dati e per quelli delle unit. La funzione *InitExecSwap* si incaricherà di creare un file di *swap* su EMS o su disco, sul quale parcheggerà temporaneamente ben 80.277 byte, lasciandone poco più di 2.000 in memoria. Poi, se tutto è andato bene, *ExecWithSwap* chiamerà una seconda copia di COMMAND.COM (cercandone il pathname nella variabile COMSPEC dell'environment), che potremo abbandonare digitando «exit».

Sulla mia macchina ho provato un CHKDSK subito prima di TESTEXEC, ottenendone l'informazione che avevo 596.304 byte disponibili; un secondo CHKDSK dato dopo aver fatto partire TESTEXEC (e naturalmente prima di «exit») me ne ha comunicati 589.536: una differenza di poco meno di 7K, quindi superiore ai 2.000 byte calcolati «a tavolino». In ogni caso, tuttavia, siamo parecchio sotto gli 80K che il programma si sarebbe mangiato usando la procedura *Exec* in dotazione al compilatore. Un bel risultato.

```
program TestExecSwap;
uses (* ExecSwap DEVE ESSERE L'ULTIMA UNIT NELLA LISTA *)
  Dos, ExecSwap;
const
  SwapLoc: array[boolean] of string[8] = ('su disco', 'su EMS');
var
  Status: word;
  LargeArray : array[1..30000] of integer;
begin
  if not InitExecSwap(HeapPtr, 'SWAP.$$$') then
    Writeln('Impossibile allocare spazio per swap')
  else begin
    Writeln('Allocati ',BytesSwapped,' bytes ',SwapLoc[EmsAllocated]);
    SwapVectors;
    Status := ExecWithSwap(GetEnv('COMSPEC'), '');
    SwapVectors;
    Writeln('Status: ',Status)
  end
end.
```

Figura 1 - Un breve programma per mostrare l'uso della funzione Exec WithSwap.

Memoria estesa e memoria espansa

Dicevo che il file di *swap* viene creato o su EMS o su disco. Data la maggiore velocità di accesso della memoria espansa, per prima cosa si verifica se questa è presente.

Qui devo correre il rischio di annoiare chi già ha le idee chiare al riguardo, a beneficio di chi non ha mai superato la barriera dei 640K. Ricordiamo innanzitutto che i processori 80x86 (nonché il vecchio 8088) possono gestire direttamente fino ad un Mega di RAM; 384K di questa non sono però normalmente accessibili, in quanto dedicati alle ROM, alla memoria video, ecc. Rimangono appunto i nostri 640K: veramente tanti fino a pochi anni fa, ormai appena sufficienti. Ecco quindi due «prolunghe»: la memoria estesa e la memoria espansa. La prima in realtà è disponibile solo su macchine dotate di un 80286 o di un 80386, e solo se si pongono questi processori in «modo protetto», ben di-

verso dal «modo reale», quello nel quale si hanno le stesse funzionalità di un normale 8086. Per agevolare (si fa per dire...) le operazioni, si dispone di un INT $15 che consente di trasferire dati tra la memoria convenzionale e quella estesa; in realtà prima di attivare quell'interrupt si deve inizializzare una *Global Descriptor Table*, si deve cioè avere un minimo di familiarità con tecniche e strumenti tipici della programmazione di sistema in ambienti multitasking (l'argomento è stato ampiamente trattato in altra rubrica della rivista). In conclusione, l'uso più ragionevole della memoria estesa è molto più semplice: se ne abbiamo un po' sul nostro 286 o 386, ne facciamo un RAMDISK solo aggiungendo una riga al file CONFIG.SYS (ad esempio con «DEVICE=RAMDRIVE.SYS 384 512 4 /E» creiamo un disco virtuale di 384K, e quel «/E» fa sì appunto che venga posto nella memoria estesa).

La memoria espansa è invece disponibile, mediante apposite schede, anche su macchine dotate di un 8088 o 8086. Nel 1985 la Lotus e la Intel proposero la *Expanded Memory Specification* (EMS), e ben presto aderì anche la Microsoft, per dar vita a quella che venne chiamata LIM EMS. Nella sua prima versione, si trattava di usare 64K della RAM compresa tra la memoria video e lla ROM di sistema, quindi accessibili anche in modo reale: venivano lì definite quattro «pagine» di 16K ognuna attraverso le quali, mandando opportuni valori a quattro porte di I/O, si poteva avere accesso fino a quattro schede di memoria, ognuna contenente fino a 2 mega di memoria organizzata in «pagine» di 16K. In parole povere, fino a 8 mega di memoria in più a nostra disposizione con le versioni 3.0 e 3.2; con la versione 4.0 l'ampiezza di quelle pagine può anche essere diversa da 16K e si può arrivare ad un totale di 32 mega di memoria aggiuntiva. La unit *ExecSwap* vuole poter essere usata su qualsiasi versione della EMS e quindi, tra l'altro, usa pagine di 16K: di qui il valore della costante *EmsPageSize* in *InitExecSwap*.

Una nota. Molti programmi riconoscono la presenza della memoria espansa e sanno sfruttarla: l'editor del Turbo Pascal può ad esempio usarla per il proprio buffer di 64K, mentre non può fare altrettanto con la memoria estesa. Sono

```
;EXECSWAP.ASM
;-------------------------------------------------------------------------
DATA    SEGMENT BYTE PUBLIC
        EXTRN   BytesSwapped:DWORD      ; dichiarate in EXECSWAP.PAS
        EXTRN   EmsAllocated:BYTE
        EXTRN   FileAllocated:BYTE
        EXTRN   EmsHandle:WORD
        EXTRN   FrameSeg:WORD
        EXTRN   FileHandle:WORD
        EXTRN   SwapName:BYTE
        EXTRN   PrefixSeg:WORD          ; dalla unit System
DATA    ENDS
;-------------------------------------------------------------------------
CODE    SEGMENT WORD PUBLIC
        ASSUME  CS:CODE,DS:DATA
        PUBLIC  ExecWithSwap,FirstToSave
        PUBLIC  AllocateSwapFile,DeallocateSwapFile
        PUBLIC  DefaultDrive,DiskFree
        PUBLIC  EmsInstalled,EmsPageFrame
        PUBLIC  AllocateEmsPages,DeallocateEmsHandle
;-------------------------------------------------------------------------
FileAttr        EQU     6               ; hidden + system
EmsPageSize     EQU     16384
FileBlockSize   EQU     32768
StkSize         EQU     128
lo              EQU     (WORD PTR 0)
hi              EQU     (WORD PTR 2)
ofst            EQU     (WORD PTR 0)
segm            EQU     (WORD PTR 2)
;-------------------------------------------------------------------------
; Variabili nel Code Segment
EmsDevice       DB      'EMMXXXX0',0    ; Device driver per l'EMS
UsedEms         DB      0               ; 1 se su EMS, 0 se su disco
BytesSwappedCS  DD      0               ; Byte da 'swappare'
EmsHandleCS     DW      0               ; handle per l'EMS
FrameSegCS      DW      0               ; Seg della page frame dell'EMS
FileHandleCS    DW      0               ; Per il swap su disco
PrefixSegCS     DW      0               ; Seg del Prog. Segment Prefix
Status          DW      0               ; Risultato di ExecSwap
LeftToSwap      DD      0               ; Byte ancora da 'swappare'
SaveSP          DW      0               ; Copia del registro SP
SaveSS          DW      0               ; Copia del registro SS
PathPtr         DD      0               ; Ptr al programma da eseguire
CmdPtr          DD      0               ; Ptr alla sua riga comando
ParasWeHave     DW      0               ; Paragrafi prima del swap
CmdLine         DB      128 DUP (0)     ; Riga comando da passare al DOS
Path            DB      64 DUP (0)      ; Pathname da passare al DOS
FileBlock1      DB      16 DUP (0)      ; Primo FCB da passare al DOS
FileBlock2      DB      16 DUP (0)      ; Secondo FCB da passare al DOS
; Le seguenti 4 variabili devono essere contigue e nell'ordine indicato
EnvironSeg      DW      0               ; Seg dell'environment
CmdLinePtr      DD      0               ; Ptr a CmdLine
FilePtr1        DD      0               ; Ptr a FileBlock1
FilePtr2        DD      0               ; Ptr a FileBlock2
TempStack       DB      StkSize DUP (0) ; Stack temporaneo
StackTop        LABEL   WORD            ; Marca l'inizio di TempStack
```

```
;-------------------------------------------------------------------------
; Macro
MovSeg          MACRO   Dest,Src        ; Assegna ad un segment register
        PUSH    Src                     ; .. il valore di un altro
        POP     Dest
                ENDM

MovMem          MACRO   Dest,Src        ; Muove da memoria a memoria
        MOV     AX, Src                 ; .. via AX
        MOV     Dest, AX
                ENDM

InitSwapCount   MACRO                   ; Inizializza il contatore per
                                        ; .. i byte da 'swappare'
        MovMem  LeftToSwap.lo, BytesSwappedCS.lo
        MovMem  LeftToSwap.hi, BytesSwappedCS.hi
                ENDM

SetSwapCount    MACRO BlkSize           ; Ritorna CX = byte da muovere
        LOCAL   FullBlk                 ; .. e riduce LeftToSwap
        MOV     CX, BlkSize             ; Assumi tutto un blocco
        CMP     LeftToSwap.hi, 0        ; Se la word alta <> 0
        JNZ     FullBlk                 ; .. salta a FullBlk
        CMP     LeftToSwap.lo, BlkSize  ; Se resta un blocco o piu'
        JAE     FullBlk                 ; .. salta a FullBlk
        MOV     CX, LeftToSwap.lo       ; Altrimenti muovi quanto resta
FullBlk:
        SUB     LeftToSwap.lo, CX       ; Decrementa LeftToSwap
        SBB     LeftToSwap.hi, 0
                ENDM

NextBlock       MACRO SegReg, BlkSize   ; SegReg := ptr al prossimo
        MOV     AX, SegReg              ; .. blocco da muovere
        ADD     AX, BlkSize/16          ; Aggiunge il numero di paragrafi
        MOV     SegReg, AX
        MOV     AX, LeftToSwap.lo       ; Rimangono byte da muovere?
        OR      AX, LeftToSwap.hi       ; Se si', Zero flag = 0
                ENDM

EmsCall         MACRO FuncAH            ; Chiama EMM e prepara l'esame
        MOV     AH, FuncAH              ; .. del risultato
        INT     67h
        OR      AH, AH                  ; Se errori, Zero flag = 0
                ENDM

DosCallAH       MACRO FuncAH            ; Chiama una funzione del DOS
        MOV     AH, FuncAH
        INT     21h
                ENDM

DosCallAX       MACRO FuncAX            ; Chiama una funzione del DOS
        MOV     AX, FuncAX
        INT     21h
                ENDM

InitSwapFile    MACRO                   ; Inizializza il file di swap
        MOV     BX, FileHandleCS
        XOR     CX, CX                  ; 0 byte
        XOR     DX, DX                  ; dall'inizio del file
        DosCallAX 4200h                 ; funzione Seek del DOS
```

Figura 2 - Il file EXECSWAP.ASM. La funzione ExecWithSwap verrà illustrata nel prossimo numero.

stati quindi approntati dei simulatori di EMS, che rendono possibile usare pare della memoria estesa o del disco rigido come se fosse EMS. Potete trovarne anche uno su MC-Link, col nome VEMM.ZIP.

La funzione InitExecSwap

Prima di verificare se c'è spazio sufficiente sulla memoria espansa eventualmente presente, bisogna sapere ... quanto spazio ci occorre. Si chiama a questo proposito una funzione *PtrDiff* (compresa nei listati del mese scorso), che calcola quanta memoria c'è tra *LastToSave* e *@FirstToSave*. Il primo è uno dei parametri passati a *InitExecSwap*, descritto la volta scorsa (in breve: l'estremo superiore della zona di memoria occupata dal programma, contenuto nelle variabili predefinite *HeapOrg* o *HeapPtr*, oppure calcolato con la formula *Ptr(Seg(FreePtr^)+$1000,0)*, secondo che si usi o no memoria dinamica); l'altro, come possiamo vedere nella figura 2, è l'indirizzo di una «label» posta subito dopo il codice della funzione *ExecWithSwap*: questa infatti deve restare in memoria, mentre tutto quanto viene dopo può essere temporaneamente eliminato; quella «label» indica quindi l'inizio della zona di memoria da sottoporre a *swap*.

La verifica della presenza della memoria EMS viene condotta chiamando la funzione *EmsInstalled*, anch'essa nella figura 2: si tratta di provare ad aprire un file dal nome EMMXXXX0 come se si trattasse di un file normale. Problema: e se non ci fosse memoria espansa ma esistesse un «vero» file con quel nome? In verità è piuttosto improbabile, ma non sarebbe una cattiva idea controllare. Vi propongo quindi nella figura 4 una variante di *EmsInstalled* in cui si usa la funzione $44 del DOS, sottofunzione $00; lasciando in BX l'*handle* del file, basta poi controllare il bit 7 del registro DX: se è zero, il file aperto è un file «vero»; se abbiamo «aperto» il file ma quel bit è uno possiamo quindi dedurne che abbiamo effettivamente memoria espansa sulla nostra macchina.

Non è questo il solo metodo: si può anche andare a spulciare nella routine associata all'INT $67 (quello attraverso il quale, come vedremo, si comunica con la EMS), per cercarvi la stringa

```
                ENDM

HaltWithError   MACRO Level             ; Se errore non recuperabile
        MOV     AL, Level               ; .. imposta Errorlevel
        DosCallAH 4Ch                   ; .. e torna al DOS
                ENDM

MoveFast        MACRO                   ; Muove CX byte da DS:SI a ES:DI
        CLD
        SHR     CX, 1                   ; Converte in numero di word
        REP     MOVSW
        RCL     CX, 1                   ; Se e' avanzato un byte 'dispari'
        REP     MOVSB                   ; .. muovi pure quello
                ENDM

SetTempStack    MACRO                   ; Imposta lo stack temporaneo
        MOV     AX, OFFSET StackTop
        MOV     BX, CS
        CLI
        MOV     SS, BX
        MOV     SP, AX
        STI
                ENDM
;-------------------------------------------------------------------------
; function ExecWithSwap(Path, CmdLine: string): word;
ExecWithSwap    PROC FAR
; verrà illustrata nel prossimo numero
ExecWithSwap    ENDP
;-------------------------------------------------------------------------
; La LABEL FirstToSave marca l'inizio della memoria da 'swappare'
FirstToSave:
;-------------------------------------------------------------------------
; function AllocateSwapFile: boolean;
AllocateSwapFile PROC NEAR
        MOV     CX, FileAttr
        MOV     DX, OFFSET SwapName + 1 ; Salta il byte di lunghezza
        DosCallAH 3Ch                   ; Crea il file
        MOV     FileHandle, AX
        MOV     AL, 0                   ; Ritorna FALSE se errori
        JC      ASDone
        INC     AL                      ; .. altrimenti TRUE
ASDone:
        RET
AllocateSwapFile ENDP
;-------------------------------------------------------------------------
; procedure DeallocateSwapFile;
DeallocateSwapFile PROC NEAR
        MOV     BX, FileHandle
        DosCallAH 3Eh                   ; Chiudi il file
        XOR     CX, CX                  ; Attributo := normale
        MOV     DX, OFFSET SwapName + 1 ; DS:DX -> nome senza byte di lungh.
        DosCallAX 4301h                 ; Imposta l'attributo
        DosCallAH 41h                   ; Cancella il file
        RET
DeallocateSwapFile ENDP
;-------------------------------------------------------------------------
; function EmsInstalled: boolean
EmsInstalled    PROC FAR
        PUSH    DS
        MovSeg  DS, CS
        MOV     DX, OFFSET EmsDevice    ; DS:DX -> nome del driver EMS
        DosCallAX 3D02h                 ; Prova ad aprirlo
        POP     DS
        MOV     BX, AX                  ; BX := handle (se aperto)
        MOV     AL, 0                   ; Ritorna FALSE se non aperto
        JC      EIDone
        DosCallAH 3Eh                   ; Chiudi
        MOV     AL, 1                   ; Ritorna TRUE
EIDone:
        RET
EmsInstalled    ENDP
;-------------------------------------------------------------------------
; function EmsPageFrame: word;
EmsPageFrame    PROC FAR
        EmsCall 41h                     ; Imposta la page frame
        MOV     AX, BX                  ; AX := segmento
        JZ      EPDone                  ; Se tutto bene, ritorna il seg.
        XOR     AX, AX                  ; .. altrimenti ritorna 0
EPDone:
        RET
EmsPageFrame    ENDP
;-------------------------------------------------------------------------
; function AllocateEmsPages(NumPages: word): word;
AllocateEmsPages PROC FAR
        MOV     BX, SP
        MOV     BX, SS:[BX+4]           ; BX := NumPages
        EmsCall 43h                     ; 'Apri' la EMS
        MOV     AX, DX                  ; AX := handle
        JZ      APDone                  ; Se tutto bene, ritorna l'handle
        MOV     AX, 0FFFFh              ; .. altrimenti ritorna $FFFF
APDone:
        RET     2
AllocateEmsPages ENDP
;-------------------------------------------------------------------------
; procedure DeallocateEmsHandle(Handle: word);
DeallocateEmsHandle PROC FAR
        MOV     BX, SP
        MOV     DX, SS:[BX+4]           ; DX := Handle
        EmsCall 45h                     ; 'Chiudi' la EMS
        RET     2
DeallocateEmsHandle ENDP
;-------------------------------------------------------------------------
; function DefaultDrive: char;
DefaultDrive    PROC FAR
        DosCallAH 19h
        ADD     AL, 'A'
        RET
DefaultDrive    ENDP
;-------------------------------------------------------------------------
; function DiskFree(Drive: byte): longint;
DiskFree        PROC FAR
        MOV     BX, SP
        MOV     DL, SS:[BX+4]           ; DL := Drive
        DosCallAH 36h
        MOV     DX, AX                  ; DX := settori per cluster
        CMP     AX, 0FFFFh              ; Numero del drive non valido?
        JZ      DFDone                  ; .. se cosi', ritorna $FFFFFFFF
        MUL     CX                      ; CX = byte per settore
        MUL     BX                      ; BX = num. settori liberi
DFDone:

        RET     2
DiskFree        ENDP
;-------------------------------------------------------------------------
CODE    ENDS
        END
```

«EMMXXXX0»; se la si trova, è praticamente certo che la memoria espansa è presente e pronta per essere usata. Si tratta di un metodo un po' garibaldino e, a mio parere, un po' meno sicuro di quello proposto nella figura 4: qualche spiritoso potrebbe magari essersi divertito a mettere quella stringa nel codice di una diversa routine associata all'interrupt. Se lo preferite, comunque, ne trovate un esempio nel file EMS.PAS, compreso nel file DEMOS.ARC distribuito insieme al Turbo Pascal. Se due metodi non vi bastano, ne potete trovare ancora altri nella prima e nella seconda edizione dell'impagabile *Advanced MSDOS* di Ray Duncan, Microsoft Press.

In ogni caso, se la EMS c'è bisogna calcolare quante «pagine» di 16K ci occorrono e provare ad allocarle (nel caso degli 80.277 byte di TESTEXEC occorrono cinque pagine). Ciò viene fatto chiamando la funzione *AllocateEmsPages*, che usa una delle funzioni di quell'INT $67 attraverso il quale un programma può comunicare con l'EMM *(Expanded Memory Manager)*. Se ne ottiene come risultato un registro AH azzerato se tutto è andato bene, più eventuali altri valori nel registri AL, BX e/o DX. La funzione $43, in particolare, è analoga alla funzione $3D dell'INT $21 (apertura di file): va chiamata con il numero delle pagine richieste in BX e, se l'operazione ha avuto successo, ritorna in DX un *handle* che potrà essere usato per le funzioni di lettura, scrittura e chiusura. Se AH è diverso da zero si è invece verificato un errore: AH vale ad esempio $88 se sulla EMS non c'è spazio disponibile per le pagine che abbiamo richiesto, o $87 se addirittura la richiesta è superiore alle dimensioni stesse della EMS installata; la nostra *AllocateEmsPages* ritorna comunque un generico $FFFF quale che sia stato l'errore. Anche qui abbiamo un'alternativa, anche questa proposta nel file EMS.PAS: potremmo infatti usare la funzione $42 dell'INT $67, che ritorna in DX il numero totale di pagine presenti nella memoria espansa e in BX il numero di quelle non ancora usate. Basterebbe confrontare quest'ultimo valore con quello della variabile *PagesInEms* e, se BX è almeno uguale, allocare poi le pagine che ci servono con la funzione $43.

Una volta portata a termine con successo l'allocazione, possiamo finalmente completare i preliminari chiamando la funzione *EmsPageFrame*. Questa ci dà l'indirizzo delle quattro pagine di 16K che, come dicevamo prima, fanno da tramite tra la memoria convenzionale e quella espansa: usa la funzione $41 dell'INT $67, che ritorna appunto in BX il segmento della zona di memoria che ospita quelle pagine. Vedremo il mese prossimo come servirsi di questa informazione per le operazioni di lettura/scrittura su EMS.

```
function InitExecSwap(LastToSave: pointer; SwapFileName: string): boolean;
const
  EmsPageSize = 16384;
var
  PagesInEms: word;
  BytesFree : longint;
  DriveChar : char;
begin
  InitExecSwap := FALSE;
  if EmsAllocated or FileAllocated then
    Exit;
  BytesSwapped := PtrDiff(LastToSave, @FirstToSave);
  if BytesSwapped <= 0 then
    Exit;
  SaveExit := ExitProc;
  ExitProc := @ExecSwapExit;
  if EmsInstalled then begin
    PagesInEms := (BytesSwapped + EmsPageSize - 1) div EmsPageSize;
    EmsHandle := AllocateEmsPages(PagesInEms);
    if EmsHandle <> $FFFF then begin
      EmsAllocated := TRUE;
      FrameSeg := EmsPageFrame;
      if FrameSeg <> 0 then begin
        InitExecSwap := TRUE;
        Exit
      end
    end
  end;
  if Length(SwapFileName) <> 0 then begin
    SwapName := SwapFileName + #0;
    if Pos(':', SwapFileName) = 2 then
      DriveChar := UpCase(SwapFileName[1])
    else
      DriveChar := DefaultDrive;
    BytesFree := DiskFree(byte(DriveChar) - $40);
    FileAllocated := (BytesFree > BytesSwapped) and AllocateSwapFile;
    if FileAllocated then
      InitExecSwap := TRUE;
  end
end;
```

Figura 3 - La funzione InitExecSwap della unit EXECSWAP (pubblicata il mese scorso).

Swap su disco

Se qualcosa non ha funzionato, se cioè la memoria espansa non è presente, non vi è spazio sufficiente, o qualcos'altro è andato storto (ad esempio problemi hardware sulle schede EMS), *InitExecSwap* prova ad aprire un file di *swap* su disco (il nome del file viene passato alla funzione nel suo secondo parametro). Si procede in modo analogo a quanto già visto per la EMS, in particolare si controlla che vi sia spazio sufficiente su disco.

Vi risparmio i dettagli delle routine interessate, sia perché il sorgente è abbondantemente commentato, sia perché «... che diamine! su disco sappiamo lavorare tutti!»

Anche qui, però, mi sento in dovere di proporvi delle varianti. Proviamo ad immaginare di non avere memoria espansa e di lanciare il nostro TESTEXEC. Verrà creato un file SWAP.$$$, che non potremo vedere con un semplice DIR perché aperto con gli attributi *hydden* e *system* (costante *FileAttr* all'inizio della figura 2). Comparirà un messaggio del tipo «Allocati 80277 byte su disco», e subito dopo ci ritroveremo la prompt del DOS. Domanda: che succede se ora chiamiamo nuovamente TESTEXEC? Risposta: verrà creato un nuovo file SWAP.$$$ che ... si sovrapporrà al primo! Dati poi i due «exit», torneremo al primo TESTEXEC dopo avergli restituito, invece del suo codice e dei suoi dati, quelli del secondo! Facile immaginare le conseguenze, potenzialmente catastrofiche, di giochetti del genere: basta pensare a chiamare non due volte TESTEXEC, ma due programmi diversi che usino lo stesso nome per il file di *swap*!

Una prima soluzione consiste nel cambiare il valore della costante *FileAttr* da 6 a 7, aggiungendo così l'attributo *readonly* (sola lettura); il secondo TESTEXEC non riuscirà a sovrapporre il suo SWAP.$$$ a quello del primo e terminerà con un messaggio d'errore. Forse non è quanto di meglio si possa

```
EmsInstalled    PROC FAR
        PUSH    DS
        MovSeg  DS, CS
        MOV     DX, OFFSET EmsDevice    ; DS:DX -> nome del driver EMS
        DosCallAX 3D02h                 ; Prova ad aprirlo
        POP     DS
        MOV     BX, AX                  ; BX := handle (se aperto)
        MOV     AL, 0                   ; Ritorna FALSE se non aperto
        JC      EIDone
        MOV     AX, 4400h               ; Chiedi informazioni sul device
        INT     21h                     ;   il cui handle è in BX
        JC      EIDone                  ; Esci se errori
        DosCallAH 3Eh                   ; Chiudi (NB: non cambia DX)
        MOV     AL, 0                   ; Assumi FALSE
        TEST    DX, 80h                 ; Se il bit 7 di DX è 0 è un file
        JZ      EIDone                  ;   quindi lascia 0 in AL
        MOV     AL, 1                   ; Altrimenti ritorna TRUE
EIDone:
        RET
EmsInstalled    ENDP
```

Figura 4 - Una possibile variante di EmsInstalled: la funzione $44 del DOS consente di verificare se EMMXXXX0 è un «vero» file.

desiderare. Un'altra, più efficace, soluzione potrebbe essere più o meno la seguente: definiamo un loop che incrementa una variabile di tipo longint da 1 in poi, provando ad ogni ciclo se esiste un file avente come nome quel numero; se esiste si prosegue, altrimenti si è trovato il nome di un file inesistente e come tale candidato ideale per l'operazione di *swap*.

Se il nome del file non contiene l'indicazione del drive (come sarebbe invece un "C:WAPDIRWAP.$$$"), viene ovviamene utilizzato il drive corrente. Si sa però che i dischi sono molto più lenti della RAM, tanto che sarebbe masochistico non usare l'eventuale RAMDISK presente. Il problema è però che non si può sapere a priori che nome il DOS gli ha assegnato, in quanto questo nome dipende dalla configurazione della macchina: sarà un drive D: se la macchina ha tre dischi «veri», E: se ne ha quattro, ecc.

In situazioni come queste mi piace usare l'environment: il nome del file di *swap* può essere costruito concatenando il valore di una variabile dell'environment e una stringa; ad esempio, si può passare a *InitExecSwap* il parametro:

GetEnv('RAMDISK')+'SWAP.$$$';

Si tenterà così di aprire il file D:WAP.$$$ se RAMDISK=D:. È possibile tuttavia che non vi sia spazio nel RAMDISK mentre ve ne sarebbe invece sul disco rigido. Si può quindi modificare *InitExecSwap* in modo che, se non c'è spazio sul RAMDISK, provi ad aprirlo sul disco corrente (basta eliminare i primi due caratteri del parametro).

Per ora ci fermiamo qui. Il mese prossimo completeremo l'esame di EXECSWAP prestando particolare attenzione al «contesto» di un programma.