di Sergio Polini

La procedura Exec

Ormai quasi tutti i programmi «seri» sono provvisti di una interfaccia col DOS: è possibile esaminare il contenuto di una directory, copiare o spostare dei file, anche lanciare altri programmi senza dover uscire da quello che si sta eseguendo. Un MAKE deve ovviamente interagire con il DOS in maniera diversa, in modo quasi invisibile per l'utente; tuttavia i meccanismi cui si ricorre sono gli stessi, al punto che quanto vedremo in questa puntata potrà aiutarvi a dotare di una completa «interfaccia DOS» qualsiasi vostro programma. Vedremo tuttavia che questi meccanismi non sono flessibili quanto sarebbe desiderabile; chiuderemo quindi qui l'illustrazione del MiniMake, ma a partire da ottobre esamineremo in dettaglio una tecnica che ci consentirà di liberarci da ogni impaccio

Il DOS, si sa, deriva dal buon vecchio CP/M. Si sa anche però che a partire dalla versione 2.0 ha fatto proprie alcune caratteristiche di Unix, tra le quali la struttura ad albero del file system, il reindirizzamento dell'I/O, le pipeline, un environment con variabili quali PATH o PROMPT e relativo comando SET, ecc. Accanto a queste caratteristiche «visibili», vi è stata anche l'aggiunta alle vecchie chiamate di funzione di un vasto insieme di nuove chiamate, che anche nel nome loro assegnato denunciano la derivazione da Unix, come DUP, IOCTL e EXEC.

La funzione 4Bh (EXEC) del DOS

In Unix, per dirla in estrema sintesi, vi sono due chiamate di sistema strettamente correlate: *fork* crea una copia — detta «figlio» — del processo che la esegue — detto «padre»; *exec* sostituisce al processo da cui viene eseguita un nuovo processo, che viene sovrapposto al primo nella memoria. Quando un processo ne vuole lanciare un altro, produce prima una copia di se stesso con *fork*, poi fa eseguire dalla copia la chiamata *exec*, mediante la quale un nuovo processo viene sostituito alla copia del primo. Non è il caso di entrare ora nei dettagli di un meccanismo tutt'altro che banale, né di illustrare la differenza tra «programma» e «processo» (magari possiamo limitarci a dire che, se un programma «si forka» e poi «exegue» se stesso, abbiamo un solo programma ma due processi); notiamo comunque subito che l'EXEC del DOS riassume in sé, in forma semplificata, le due chiamate di Unix: azzerando il registro AL si carica e si esegue un programma, dal quale si ritorna poi quando questo termina; ponendo un 3 in AL, invece, il nuovo programma si sovrappone a quello originario, che scompare così di scena. In realtà quest'ultima possibilità viene sfruttata piuttosto raramente, per cui possiamo limitarci al caso in cui AL vale zero.

Non basta tuttavia mettere un 4Bh in AH e uno zero in AL. Prima di chiamare la funzione 4Bh si deve assegnare alla coppia DS:DX l'indirizzo — segmento e offset — di una zona di memoria in cui sia contenuto il pathname completo del programma da eseguire, estensione COM o EXE compresa (non è possibile lanciare un file BAT se non attraverso l'esecuzione di COMMAND.COM), e alla coppia ES:BX l'indirizzo di un *parameter block* contenente a sua volta gli indirizzi di un *environment block*, di una stringa composta con gli argomenti di quella che sarebbe stata la riga comando se si fosse lanciato il programma dal prompt del DOS, di due default *File Control Block* (FCB).

Per *environment block* si intende un'area di memoria in cui sono definite alcune variabili a beneficio del programma in esecuzione: il COMMAND.COM cerca ad esempio nella variabile PATH le subdirectory in cui può trovarsi un file COM o EXE o BAT, mentre il valore di COMSPEC è il pathname completo dello stesso COMMAND.COM, utile quando si voglia lanciarlo per eseguire i comandi interni del DOS (come DIR, TYPE, ecc.) oppure un file batch. Ogni variabile nell'environment è seguita da un segno di uguale, dal suo valore (una stringa di caratteri) e da uno zero; due zeri consecutivi indicano la fine del *block*. In generale, un environment è il mezzo attraverso il quale il DOS «passa parametri» al COMMAND.COM, questo ai programmi che lanciamo alla prompt

```
program DumpEnv;
uses
  Dos;
type
  Env = array[0..32767] of char;
var
  EnvPtr: ^Env;
  i: integer;
  Ch,Precedente: char;
  FineEnv: boolean;
begin
  EnvPtr := Ptr(MemW[PrefixSeg:$002C], 0);
  i := 0;
  Precedente := #0;
  FineEnv := FALSE;
  Writeln('Environment block:');
  repeat
    Ch := EnvPtr^[i];
    if Ch = #0 then begin
      if Precedente = #0 then
        FineEnv := TRUE;
      Write('<0>')
    end else
      Write(Ch);
    Precedente := Ch;
    Inc(i)
  until FineEnv;
  Writeln
end.
```

Figura 1 - Un breve programma per dare una sbirciatina all'environment.

del DOS, ogni programma ai suoi programmi «figli». Un programma che ne esegua un altro può passargli informazioni mediante un environment (si hanno fino a 32K a disposizione), cioè mediante un'area di memoria il cui segmento venga posto nella prima word del *parameter block*. Se si vuole invece passare una copia dell'environment del programma «padre», bisogna leggerne il segmento nell'offset $002C del *Program Segment Prefix*.

I FCB sono strutture di dati lunghe 37 byte, mantenute nel DOS esclusivamente per compatibilità con il CP/M, che li usava per l'accesso ai file. Nel DOS non sono quasi più usati, anche perché non è possibile accedere tramite i FCB a file che non siano nella directory corrente; la funzione 29h assume comunque che il primo argomento della riga comando sia il nome di un file, che scompone in drive, nome ed estensione per scrivere poi tali componenti nei primi 12 byte di un'area di memoria il cui indirizzo sia in ES:DI; eseguendo la funzione 29h una seconda volta si elabora in modo analogo il secondo argomento. È necessario passare gli indirizzi di ambedue i FCB così costruiti alla funzione EXEC, nel caso questa venga usata per lanciare un programma «vecchio stile».

Prima di chiamare EXEC bisogna anche pensare a salvare e poi ripristinare tutti i registri del microprocessore, compresi quelli usati per la gestione dello stack (SS e SP). In breve, un meccanismo piuttosto complesso, di cui vi avevo proposto un esempio a febbraio dello scorso anno con la funzione *Spawn*.

La procedura Exec del Turbo Pascal

Attivare la chiamata EXEC del DOS non è facilissimo, ma molto utile. La Borland ha quindi pensato bene di arricchire la libreria di procedure del suo compilatore, fin dalla versione 4.0, di una procedura *Exec* che semplifica il tutto: basta passarle due stringhe, delle quali la prima sia il pathname completo del programma da eseguire e la seconda gli argomenti che gli daremmo sulla riga comando. Non è però possibile passare al programma «figlio» un environment diverso da una semplice copia di quello del «padre»; se voleste questo dovreste scrivervi una vostra procedura, magari traendo spunto dalla funzione

```
program EnvDemo;
uses
  Environ;
var
  Variabile, NomeFile: string;
begin
  repeat
    Write('Variabile (in maiuscolo): ');
    Readln(Variabile);
    if Variabile <> '' then
      Writeln(GetEnv(Variabile))
  until Variabile = '';
  repeat
    Write('Nome file: ');
    Readln(NomeFile);
    if NomeFile <> '' then
      Writeln(FSearch(NomeFile, GetEnv('PATH')))
  until NomeFile = ''
end.
```

Figura 3 - Un programma di prova per la unit Environ della figura 2.

```
unit Environ;
  (* funzioni GetEnv e FSearch per Turbo Pascal 4.0 *)
interface

function Getenv(EnvVar: string): string;

function FSearch(FileName, PathList: string): string;

implementation

type
  Environment = array[0..32767] of char;

function GetEnv(EnvVar: string): string;
var
  EnvPtr : ^Environment;
  EnvStr : string;
  FineEnv: boolean;
  i, Len : integer;
begin
  EnvPtr := Ptr(MemW[PrefixSeg:$002C], 0);
  i := 0;
  FineEnv := FALSE;
  EnvStr := '';
  EnvVar := EnvVar + '=';
  Len := Length(EnvVar);
  repeat
    if EnvPtr^[i] = #0 then begin
      if EnvPtr^[i+1] = #0 then
        FineEnv := TRUE;
      if Copy(EnvStr, 1, Len) = EnvVar then begin
        GetEnv := Copy(EnvStr, Len+1, 255);
        Exit
      end;
    EnvStr := ''
    end
    else
      EnvStr := EnvStr + EnvPtr^[i];
    Inc(i)
  until FineEnv;
  GetEnv := ''
end;

function FSearch(FileName, PathList: string): string;
var
  P: integer;
  Path, PathName, PList: string;
function Exist: boolean;
  var
    f: file;
  begin
    Assign(f, PathName);
    {$I-} Reset(f) {$I+};
    if IOResult <> 0 then
      Exist := FALSE
    else begin
      Exist := TRUE;
      Close(f)
    end
  end;
begin
  PList := PathList;
  FSearch := '';
  repeat
    P := Pos(';', PList);
    if P <> 0 then begin
      Path := Copy(PList, 1, P-1);
      PList := Copy(PList, P+1, 255)
    end
    else begin
      Path := PList;
      PList := ''
    end;
    if Path[Length(Path)] <> '' then
      Path := Path + '';
    PathName := Path + FileName;
    if Exist then begin
      FSearch := PathName;
      Exit
    end
  until PList = ''
end;

end.
```

Figura 2 - Le funzioni GetEnv e FSearch sono state introdotte con la versione 5.0 del Turbo Pascal. La unit Environ ne offre una implementazione per la versione 4.0.

Spawn (che trovate anche su MC-Link nel file SPAWN.ARC) o da quanto esporremo nelle prossime puntate. Ci si deve inoltre preoccupare della gestione della memoria del Turbo Pascal, intervenendo con la direttiva M.

Vedremo meglio il mese prossimo come un programma compilato con il Turbo Pascal usa la RAM disponibile; per ora ci è sufficiente ricordare che per default viene allocato uno stack di 16K e tutta la RAM lasciata libera dal codice, dai dati e dallo stack viene riservata per lo heap, cioè per l'allocazione dinamica con New o GetMem. Si può intervenire su queste scelte con la direttiva M, che vuole tre argomenti: dimensione dello stack e dimensioni minima e massima dello heap. Intervenendo sul primo si può ridurre o aumentare lo stack, ad esempio in funzione della presenza o meno di chiamate ricorsive; impostando una dimensione minima dello heap diversa da zero (default) si causa un errore se la RAM disponibile non è sufficiente a soddisfare tale minimo. Quello che ora ci interessa è però la dimensione massima dello heap: per default è di 640K, e ciò comporta che il programma si appropria di tutta la RAM disponibile; ne segue che non rimane spazio per altri programmi da eseguire con Exec. Dopo ogni Exec bisognerebbe esaminare la variabile DosError: se tutto è andato bene questa vale zero, altrimenti assume un valore compreso tra 2 e 18; un 8 ci direbbe che non c'è memoria sufficiente, come accadrebbe se non intervenissimo sulla direttiva M.

A questo punto viene spontaneo chiedersi che valore dare ai tre argomenti di questa. Quanto alla dimensione dello stack, la cosa più ragionevole è testare il programma con la direttiva S attiva, che provoca l'interruzione del programma se si verifica un overflow; in questo caso non rimane altro che dare valori via via più alti fino a che non si elimina l'errore, poi eventualmente disattivare la direttiva (se ne guadagna un pochino in velocità di esecuzione). Quanto alle dimensioni dello heap, due zeri vanno ovviamente benissimo se il programma non usa l'allocazione dinamica, altrimenti siamo in una situazione un po' scomoda: si usano infatti strutture dinamiche di dati proprio quando non si può calcolare a priori quanta memoria occorre, ed essere costretti a stabilire un limite non è gradevole. In teoria ci si potrebbe regolare sulla quantità di memoria richiesta dai programmi che si vuole chiamare con Exec, almeno quelli assolutamente indispensabili: si potrebbe ad esempio voler assicurare solo l'esecuzione del COMMAND.COM per consentire all'utente l'accesso ai comandi interni del DOS. Il problema è però che occorre anche fare ipotesi sulla RAM fisicamente disponibile, e non è detto che tutte le macchine su cui il programma girerà avranno 512 o 640K. Si tratta di un problema generale,

```
{$M 16384, 0, 16384}              (* Necessario per Exec *)
program MiniMake;
{$L UPSTR}
uses
  Dos, MMAlex, MMSim, MMParser;

var
  Target: TPtr;

function UpStr(S: ComStr): ComStr; external;

procedure DelTarget(Target: PathStr);
var
  f: file;
begin
  Assign(f, Target);
  {$I-} Reset(f); {$I+}
  if IOResult = 0 then begin      (* se il target e' un file esistente *)
    Close(f);
    Erase(f);
    Writeln(UpStr(Target), ' cancellato')
  end
end;

procedure ErroreDos(Target,Comando:PathStr; Argomenti:ComStr; Codice:integer);
begin
  Write(Comando,' ',Argomenti,': ');
  case Codice of
    2: Writeln('file non trovato');
    3: Writeln('path non trovato');
    5: Writeln('accesso negato');
    6: Writeln('"handle" non valido');
    8: Writeln('memoria insufficiente');
   10: Writeln('"environment" non valido');
   11: Writeln('formato non valido');
   18: Writeln('troppi file aperti')
  end;
  DelTarget(Target);
  Halt(1)
end;

procedure Esegui(Target, Comando: PathStr; Argomenti: ComStr);
const
  Ext: array[1..3] of string[4] = ('.EXE', '.COM', '.BAT');
var
  i       : integer;
  ExitCode: word;
  Env     : string;
  Path    : PathStr;

begin
  writeln(#9,Comando,' ',Argomenti);
  Env := GetEnv('PATH');
  i := 0;
  if Pos('.', Comando) = 0 then begin
    Path := '';
    repeat
      Inc(i);
      Path := FSearch(Comando+Ext[i], Env)
    until (Path <> '') or (i > 3)
  end
  else
    Path := Comando;
  SwapVectors;
  if i >= 3 then
    Exec(GetEnv('COMSPEC'), '/C '+Comando+' '+Argomenti)
  else
    Exec(Path, Argomenti);
  SwapVectors;
  if DosError <> 0 then
    ErroreDos(Target, Comando, Argomenti, DosError);
  ExitCode := DosExitCode;
  if ExitCode <> 0 then begin
    case Hi(ExitCode) of
      1: Writeln(Comando,': interrotto dall''utente');
      2: Writeln(Comando,': errore su device');
      3: Writeln(Comando,': programma residente');
    end;
    if Lo(ExitCode) <> 0 then
      Writeln(UpStr(Comando),': errore di esecuzione');
    DelTarget(Target);
    Halt(1);
  end
end;

procedure AssegnaDataOra;
begin end;

procedure Aggiorna(Nome: PathStr);
begin end;

begin
  Writeln('MiniMAKE Versione 1.0 - Sergio Polini (MC1166)');
  Init;
  Parse;
  AssegnaDataOra;
  if ParamCount > 0 then
    Aggiorna(ParamStr(1))
  else
    Aggiorna(PrimoTarget^.Id^.Nome)
end.
```

Figura 4 - Il file MMAKE.PAS. Le procedure AssegnaDataOra e Aggiorna sono state illustrate nel numero di luglio.

nel senso che, a meno di optare per soluzioni che rendono poco efficiente tutto un programma, si presenta anche con compilatori di altri produttori e per altri linguaggi. Vedremo il mese prossimo come risolvere radicalmente il problema; per ora ci limiteremo a dare ai tre argomenti della direttiva M del MiniMake valori «arbitrari», ma sufficienti per la maggior parte dei casi (si inciamperebbe solo su un makefile eccezionalmente lungo).

Un'ultima considerazione. Ogni programma compilato con il Turbo Pascal assegna proprie routine a diversi interrupt (ben 18 nel TP 5.x), e ovviamente tali routine risiedono nel codice dello stesso programma. Se tali interrupt «scattassero» quando è in esecuzione il programma «figlio» si potrebbe generare un bel po' di confusione; per evitare misteriosi comportamenti, a partire dalla versione 5.0 si dispone di una procedura SwapVectors, che scambia le routine associate a quegli interrupt con le routine salvate in apposite variabili SaveIntXX (al posto di «XX» ci sono i numeri in esadecimale dei 18 interrupt). Il manuale consiglia quindi di chiamare SwapVectors sia prima che dopo Exec: la prima volta vengono riassociate agli interrupt le routine attive prima che il programma le cambiasse, la seconda volta viene ripetuta l'assegnazione che il programma aveva operato appena partito. Vale la pena di seguire il consiglio; vale anche la pena di dichiarare ulteriori variabili SaveIntXX per eventuali altri interrupt che il vostro programma ridefinisse e poi usare le procedure GetIntVec e SetIntVec pèr «swapparli così» come fa SwapVectors.

```
        DOSSEG
CODE    SEGMENT WORD PUBLIC
        ASSUME  CS:CODE
        PUBLIC  UpStr
;---------------------------------------
; function UpStr(S: string): string;
UpStr   PROC    NEAR
        PUSH    BP
        MOV     BP, SP
        PUSH    DS
        LDS     SI, [BP+4]
        LES     DI, [BP+8]
        LODSB
        MOV     CL, AL  ; byte di lunghezza
        INC     CL
Ciclo:
        STOSB
        DEC     CL
        JZ      Fine
        LODSB
        CMP     AL, 'a'
        JB      Ciclo
        CMP     AL, 'z'
        JA      Ciclo
        SUB     AL, 20h
        JMP     SHORT Ciclo
Fine:
        POP     DS
        POP     BP
        RET     4
UpStr   ENDP
;---------------------------------------
CODE    ENDS
        END
```

Figura 5 - Il file UPSTR.ASM, contenente una funzione che ritorna una stringa uguale a quella passatale come parametro, ma con tutte le lettere minuscole convertite in maiuscole.

Cercando nell'environment

Abbiamo visto che la procedura Exec richiede che le si indichi il pathname completo del programma da eseguire, proprio come accade con la funzione EXEC del DOS. Scomodo: pensate ad un programma che cerchi COMMAND .COM nel drive C su una macchina a due floppy. L'environment, d'altra parte, serve proprio ad evitare problemi del genere: prima di chiamare COMMAND .COM devo cercare il valore della variabile COMSPEC dell'environment e poi passare questo a Exec. Le versioni 5.x offrono una funzione GetEnv che ritorna appunto il valore della variabile dell'environment passatale come argomento (attenzione: tutta in maiuscolo!); ad esempio sulla mia macchina GetEnv('COMSPEC') ritorna 'C:COMMAND.COM'.

Non basta. Se passo a Exec il risultato di GetEnv('COMSPEC') lancio una copia del COMMAND.COM, con la quale posso poi eseguire qualsiasi cosa; unica accortezza sarà di mettere un '/C' all'inizio del secondo parametro, ad esempio: Exec(GetEnv('COMSPEC'), '/C DIR'). Tra l'altro il COMMAND.COM si incarica di cercare un file eseguibile, COM o EXE o BAT, in tutte le directory assegnate alla variabile PATH. Per quanto ciò sia comodo, c'è un inconveniente: la variabile DosError contiene l'eventuale errore che ha causato la mancata esecuzione del programma passato a Exec, ma nulla ci dice circa eventuali errori incontrati da quel programma durante la sua esecuzione. Meglio: ogni programma dovrebbe terminare con un codice d'errore, uguale a zero se tutto è andato bene o diverso da zero se qualcosa è andato storto; serve proprio a questo il parametro opzionale della procedura Halt, che si avvale della funzione 4Ch del DOS. Un codice diverso da zero può poi essere intercettato da un file batch, con i test del tipo IF ERRORLEVEL .., o anche dal programma «padre», il quale — se realizzato in Turbo Pascal — può interrogare la funzione DosExitCode. Il risultato di questa è una word in cui il byte «alto» contiene 0 se è andato tutto bene, 1 se il programma è stato interrotto da un Ctrl-C, 2 se è intervenuto un errore su qualche device, 3 se si trattava di un programma che è rimasto residente in memoria; il byte «basso» contiene invece il codice ritornato dalla funzione 4Ch. Se eseguiamo tramite COMMAND.COM i comandi esterni del DOS o altri programmi, quel codice viene fagocitato dal COMMAND.COM e non possiamo quindi leggerlo con Dos ExitCode; ne segue che conviene chiamare direttamente i programmi COM o EXE.

Per far questo, tuttavia, dobbiamo conoscerne il pathname completo che, se vogliamo realizzare programmi sufficientemente flessibili, deve essere ricercato scorrendo tutte le directory del PATH. Anche qui ci viene in aiuto il compilatore, con la funzione FSearch: questa cerca il file passatole come primo argomento in tutte le directory elencate nel secondo argomento. Nell'elenco le varie directory devono essere separate da un punto e virgola, che è lo stesso formato usato per la variabile PATH dell'environment. In sostanza si tratta di procedere come segue: ricavo il valore del PATH, passo questo e il nome del comando da eseguire a FSearch; se ne ottengo una stringa nulla vuol dire che neppure COMMAND .COM potrebbe eseguirlo, altrimenti ottengo il suo pathname completo, che passo a Exec. Se il comando non contiene l'estensione, provo FSearch con tutte e tre le estensioni possibili (COM, EXE e BAT); se trovo un file COM o EXE lo eseguo direttamente, se trovo un file BAT lo eseguo passandolo come argomento a COMMAND.COM; se non trovo nulla potrebbe trattarsi di un comando interno del DOS, che provo quindi a passare al COMMAND.COM. La procedura Esegui del MiniMake si regola proprio così.

Conclusione

Con questa puntata termina l'esposizione del MiniMake: la figura 4 contiene le parti non ancora pubblicate del modulo principale, MMAKE.PAS, mentre la figura 5 vi propone il sorgente in assembler di una funzione UpStr che ritorna una stringa uguale a quella passatale come argomento, ma con tutte le lettere maiuscole. Si tratta più di un semplice esempio di interfacciamento tra Pascal e Assembler che di qualcosa di essenziale: viene infatti usata solo dalla routine DelTarget che, se vi sono stati errori durante l'aggiornamento del Target, lo cancella dal disco, in modo da non lasciarne in giro una versione non aggiornata e magari non correttamente funzionante.

Il mese prossimo cominceremo a vedere non solo come svincolare la procedura Exec dalla necessità di dare non si sa bene quali valori alla direttiva M, ma anche come rendere possibile l'esecuzione di programmi «ingombranti», tali che la somma della loro dimensione e di quella del programma che li esegue supera i 640K. Non mancate!