di Sergio Polini

Come il DOS comunica con i programmi residenti

Chiunque abbia sfogliato con «normale» curiosità la documentazione tecnica del DOS, o testi come quello di Duncan, ha capito perfettamente il tema della puntata di questo mese. Qualcun altro sarà magari rimasto un po' perplesso: ma i programmi residenti non erano una sorta di peccaminoso prodotto dell'impudenza di arditi programmatori? Non erano altezzosamente ignorati dalla Microsoft? Ebbene no: basta ricordare l'ormai onnipresente PRINT, comando del DOS con evidenza realizzato come programma residente, o anche i vari ASSIGN, APPEND, SHARE, ecc., anch'essi prodotto della stessa tecnica. Mi auguro che la discussione di questo mese aiuti i «perplessi» (e magari anche gli altri...) a trarre maggiore beneficio dalle estensioni che il DOS si è dato in tema di TSR

La volta scorsa avevamo quasi ultimato l'illustrazione della procedura EseguiTSR, rimandando solo la discussione degli ultimi dettagli. Prima di trattare del «canale di comunicazione» annunciato dal titolo, completeremo quindi quel discorso. Sarà l'occasione per vedere a quali trucchi si debba a volte ricorrere per far fare al DOS quello che vogliamo (sarà bene precisare, magari, che si tratta di trucchi consigliati dalla *MS-DOS Encyclopedia*, non dalla mia smanettomania; in altri termini: potete stare tranquilli).

PSP e DTA

Nell'appuntamento dello scorso dicembre avevamo visto come il Program Segment Prefix (PSP), pur essendo un residuo del vecchio CP/M, si sia evoluto al punto da rimanere componente essenziale del mondo MS-DOS in generale e del «contesto» di un programma in particolare. Ricorderete, tra l'altro, l'importanza dei File Control Block (FCB) che, pur essendo anch'essi di provenienza CP/M, sono ancora essenziali per alcune funzioni del DOS. Ricorderete anche la tabella degli handle, richiesti dalle funzioni introdotte con la versione 2.0 del sistema operativo, e la Disk Transfer Area (DTA).

Se un programma residente non rimanesse costantemente «associato» al proprio PSP, potrebbe trovarsi ad usare arbitrariamente il «contesto» del programma interrotto; ciò può comportare problemi, ovviamente, solo nel caso di operazioni su disco, ma se non si volessero eliminare alla radice tali possibili problemi, vi sarebbero solo due alternative: chiudere gli occhi e sperare che vada tutto bene, oppure limitare drasticamente le possibili applicazioni della nostra unit TSR. Ecco perché nella procedura *Installa* (figura 3) viene subito salvato nella variabile *NuovoPSP* il PSP del programma residente, in modo che la procedura *EseguiTSR* possa, al momento dell'attivazione, registrare in PrevPSP il PSP del programma interrotto e restituire al TSR quello che questo aveva al momento della installazione, per poi rimettere tutto a posto dopo l'esecuzione della procedura *TSRProg*.

Ci si serve a questo scopo delle procedure illustrate nella figura 1, che ricor-

```
procedure GetPSP(var SegPSP: word);
var
  Reg: registers;
begin
  if VersioneDOS < $300 then        (* forza uso di AuxStack *)
    Inc(AddrFlagErrCrit^);
  Reg.AH := $51;
  MsDos(Reg);
  SegPSP := Reg.BX;
  if VersioneDOS < $300 then
    Dec(AddrFlagErrCrit^)
end;

procedure SetPSP(SegPSP: word);
var
  Reg: registers;
begin
  if VersioneDOS < $300 then        (* forza uso di AuxStack *)
    Inc(AddrFlagErrCrit^);
  Reg.AH := $50;
  Reg.BX := SegPSP;
  MsDos(Reg);
  if VersioneDOS < $300 then
    Dec(AddrFlagErrCrit^)
end;
```

Figura 1 - Le procedure che leggono e impostano il PSP.

Figura 2 - Le procedure che leggono e impostano la DTA.

```
procedure GetDTA(var Segmento, Offset: word);
var
  Reg: registers;
begin
  Reg.AH := $2F;
  MsDos(Reg);
  Segmento := Reg.ES;
  Offset := Reg.BX;
end;

procedure SetDTA(Segmento, Offset: word);
var
  Reg: registers;
begin
  Reg.DS := Segmento;
  Reg.DX := Offset;
  Reg.AH := $1A;
  MsDos(Reg);
end;
```

rono, a loro volta, alle funzioni 50h e 51h del DOS. Queste richiedono qualche cautela in quanto, nelle versioni 2.x del DOS, usano l'area di memoria chiamata *IOStack,* di cui avevamo discusso a febbraio, cioè la stessa area usata dalle funzioni DOS da 01h a 0Ch, quelle che, mentre aspettano che l'utente faccia qualcosa, chiamano ripetutamente l'INT 28h. Sappiamo ormai bene che questa è una situazione in cui è possibile che venga attivato un programma residente, il quale quindi deve evitare di chiamare funzioni DOS che usino *IOStack* al fine di non corrompere le informazioni parcheggiate in quest'area (che, ricordiamo, non è un vero e proprio stack) dalla funzione interrotta. Ciò impedirebbe l'uso delle funzioni 50h e 51h, ma per fortuna possiamo ricorrere ad un banale «trucco»: perché un TSR possa essere attivato, il flag *ErrorMode* (anch'esso visto a febbraio) deve essere zero; se lo incrementiamo mediante il puntatore *AddrFlagErrCrit,* facciamo credere al DOS che si sia verificato un errore critico: in tali situazioni tutte le funzioni che usano normalmente *IOStack* si avvalgono invece dell'area di memoria *AuxStack*. Ciò avviene per consentire alla routine eventualmente associata all'INT 24h di chiamare senza danni le funzioni DOS da 01h a 0Ch, ma può ovviamente anche aiutarci a risolvere il nostro problema. Tutto liscio, invece, nelle versioni successive del DOS, in quanto in esse le funzioni 50h e 51h usano lo stack del programma chiamante.

Sono più semplici le procedure che leggono e impostano l'indirizzo della DTA (figura 2). Merita qualche parola solo la scelta di dare per scontato che la DTA del programma residente sia quella di default (all'offset 80h del PSP), mentre, in generale, può ben capitare che un programma sostituisca a questa una *Disk Transfer Area* diversa e più ampia. In realtà, tuttavia, «restituire» al TSR la DTA di default ha il solo scopo di assicurare che in nessun caso venga usata quella del programma interrotto; per il resto, un programma in Turbo Pascal

Figura 3 - La procedura che installa il programma residente.

```
procedure Installa(Nome:string; Prog:Proc; ID:byte; Scan:byte; Shift:byte);
var
  Reg: registers;
begin
  Tasto := Scan;
  StatoShift := Shift;
  TSRProg := Prog;
  MultiplexID := ID;
  if MultiplexID < $C0 then begin
    Writeln('MultiplexID non valido.');
    Writeln('Valori consentiti: C0h..FFh.');
    Exit
  end;
  NuovoSS  := SSeg;
  NuovoSP  := SPtr;
  NuovoPSP := PrefixSeg;
  VersioneDOS := Swap(DosVersion);
  if VersioneDOS < $200 then begin
    Writeln('Necessario DOS 2.x o versioni successive.');
    Exit
  end;
  if not CercaFlag(AddrFlagInDOS, AddrFlagErrCrit) then begin
    Writeln('Flag errori critici non trovato.');
    Exit
  end;
  if VersioneDOS < $300 then begin
    if GetIntVec2F = FALSE then
      SetInt2FVuoto
    else begin
      Reg.AX := $FF00;
      Intr($2F,Reg);
      if Reg.AH <> $FF then begin
        Writeln('PRINT.COM gia'' installato.');
        Writeln('Impossibile installare ',Nome,'.');
        Exit
      end
    end
  end;
  Reg.AH := MultiplexID;
  Reg.AL := 0;
  Intr($2F,Reg);
  if Reg.AL <> 0 then begin
    if Reg.AL = 1 then
      Writeln('Impossibile installare ',Nome,'.')
    else if Reg.AL = $FF then
      Writeln(Nome,' gia'' installato.');
    Exit
  end;
  GetIntVec(5, Addr(PrevInt5));
  SetIntVec(5, Addr(NuovoInt5));
  GetIntVec(8, Addr(PrevInt8));
  SetIntVec(8, Addr(NuovoInt8));
  GetIntVec(9, Addr(PrevInt9));
  SetIntVec(9, Addr(NuovoInt9));
  GetIntVec10;
  SetIntVec10;
  GetIntVec13;
  SetIntVec13;
  GetIntVec($28, Addr(PrevInt28));
  SetIntVec($28, Addr(NuovoInt28));
  if GetIntVec2F then                (* se era nil, ora e' Int2FVuoto *)
    ;                                (*   cfr. TSRINT.ASM             *)
  SetIntVec2F;
  SwapVectors;
  Writeln(Nome,' installato.');
  Keep(0)
end;
```

usa per l'accesso ai file solo le funzioni DOS che lavorano con gli *handle,* mai quelle che presuppongono FCB e DTA. Le uniche eccezioni sono rappresentate dalle procedure *FindFirst* e *FindNext,* che però provvedono autonomamente, ogni volta che vengono chiamate, ad impostare una DTA coincidente con il loro parametro di tipo *SearchRec.* Se il vostro programma (e perché mai?) dovesse usare direttamente le funzioni DOS «vecchio stile», quelle che vogliono FCB e DTA, sarà vostra cura impostare correttamente una eventuale DTA diversa da quella di default.

Il Multiplex Interrupt

Voltiamo pagina. Con il DOS 2.0 ha fatto la sua apparizione PRINT.COM, un programma residente che, una volta caricato in memoria, consente di inviare alla stampante uno o più file senza impedire la contemporanea esecuzione di altri programmi. Tecnicamente si dice che si tratta di uno *spooler;* in pratica, rappresenta uno strumento tanto utile quanto sottoutilizzato. Mi spiego meglio. L'utente esperto non ha ovviamente alcun problema: è a conoscenza della esistenza di PRINT.COM, sa dove ritrovare, se non la ricorda, la sintassi del comando, lo userà ogni volta che ne avrà bisogno. Altrettanto non può dirsi del «normale» utente di programmi applicativi, quasi costretto a chiedere alla sua macchina solo quello che l'autore del programma ha previsto e messo a sua disposizione. E qui devo dire che mi è capitato di vedere troppe volte programmi che impongono «tempi morti» lunghi e non necessari, mentre una scritta del tipo «Attendere» domina lo schermo: l'esecuzione della stampa blocca l'esecuzione anche per alcuni minuti, senza che l'utente possa fare altro che girare i pollici. Vi sono sì programmi che consentono la stampa «in background» di un file, ma si tratta quasi sempre di programmi protagonisti dei cataloghi di case illustri, a partire dal glorioso WordStar della MicroPro. Il normale programma applicativo sembra ignorare le possibilità di cui la Microsoft ha da tempo dotato il DOS.

La colpa magari non è solo degli autori, ma anche della stessa Microsoft, che ha documentato l'interfaccia con il PRINT.COM solo a partire dal DOS 3.0: si tratta dell'INT 2Fh, ovvero del cosiddetto *Multiplex Interrupt,* che mette a disposizione appunto una sorta di canale di comunicazione con PRINT, ASSIGN, SHARE e APPEND, ed in generale con programmi residenti. Perché la comunicazione possa avere luogo, occorre che ad ogni programma venga assegnato un byte di identificazione: 01h per PRINT, 02h per ASSIGN, 10h per SHARE, B7h per APPEND; in generale, si intendono riservati per il DOS i valori fino a BFh, mentre sono a disposizione di noi programmatori tutti i valori compresi tra C0h e FFh.

Una prima funzione dell'INT 2Fh è quella di dirci se il programma residente cui corrisponde un certo byte di identificazione è stato installato o meno. Per far questo basta mettere quel byte in AH e uno zero in AL; se ne otterrà un valore in AL da interpretare come segue: se è zero, il programma non è stato installato ma può essere installato; se è 01h non è stato installato e non può essere installato, se è FFh il programma è già residente in memoria. In alcuni casi, come con PRINT o con APPEND, sono presenti anche altre funzionalità.

Avrete già intuito che uso fa la nostra unit dell'INT 2Fh. Prima di vederne i dettagli, tuttavia, vi propongo nella figura 4 una unit PRINT.PAS che potrebbe aiutarvi a rendere più efficienti i vostri programmi che fanno uso della stampante. L'idea di base è piuttosto semplice: invece che mandare lunghi testi in stampa, potete farne un file su disco (magari un *RAMdisk*) da stampare poi in background mediante il PRINT del DOS, se già installato, senza bisogno di ricorrere alla procedura *Exec* per dare il comando.

A questo scopo, la unit PRINT propone una *interface* mediante la quale si ha

```
unit Print;

interface

function  PrintInstallato: boolean;
procedure StampaInBackground(Path: string; var CodErr: integer);
procedure RimuoviFileDaCoda(Path: string; var CodErr: integer);
procedure CancellaTuttiDaCoda(var CodErr: integer);
procedure PrimoFileInCoda(var Path: string; var CodErr: integer);
procedure NextFileInCoda(var Path: string; var CodErr: integer);
procedure ContinuaStampa(var CodErr: integer);

implementation

uses Dos;

type
  FunzioniDiPrint =
    (Installazione, Stampa, RimuoviFile, CancellaTutto, Coda, Sblocca);
  FileInCoda = array[1..64] of char;

var
  CodaPtr: ^FileInCoda;
  i      : integer;

procedure Int2F(Fun: FunzioniDiPrint; var Segm, Offs: word; var Err: integer);
var
  Reg: registers;
begin
  Reg.AH := $01;                    (* "id" del print spooler *)
  Reg.AL := byte(Fun);
  Reg.DS := Segm;                   (* DS:DX significativi solo se Fun *)
  Reg.DX := Offs;                   (*  pari a Stampa o a RimuoviFile  *)
  Intr($2F, Reg);
  if (Reg.Flags and 1) <> 0 then    (* se settato il flag carry ... *)
    Err := Reg.AX
  else begin
    if Fun = Installazione then
      if Reg.AL = $FF then Err := 0       (* PRINT e' installato *)
      else Err := -1
    else begin
      Err := 0;
      if Fun = Coda then begin
        Segm := Reg.DS;
        Offs := Reg.SI;
      end;
    end;
  end;
end;

function PrintInstallato: boolean;
var
  S, O  : word;
  CodErr: integer;
begin
  Int2F(Installazione, S, O, CodErr);
  PrintInstallato := CodErr = 0;
end;
```

Figura 4 - La unit PRINT, per l'uso dello spooler di stampa del DOS da parte di un qualsiasi programma applicativo.

```
procedure StampaInBackGround(Path: string; var CodErr: integer);
var
  S, O      : word;
  Pacchetto: record
    Livello : byte;
    Offset  : word;
    Segmento: word;
  end;
begin
  Path := Path + #0;                    (* stringa ASCIIZ con #0 finale *)
  Pacchetto.Livello := 0;               (* unico valore consentito *)
  Pacchetto.Offset := Ofs(Path) + 1;    (* salta il byte di lunghezza *)
  Pacchetto.Segmento := Seg(Path);
  S := Seg(Pacchetto);
  O := Ofs(Pacchetto);
  Int2F(Stampa, S, O, CodErr);
end;

procedure RimuoviFileDaCoda(Path: string; var CodErr: integer);
var
  S, O: word;
begin
  Path := Path + #0;                    (* stringa ASCIIZ con #0 finale *)
  S := Seg(Path);
  O := Ofs(Path) + 1;                   (* salta il byte di lunghezza *)
  Int2F(RimuoviFile, S, O, CodErr);
end;

procedure CancellaTuttiDaCoda(var CodErr: integer);
var
  S, O: word;
begin
  Int2F(CancellaTutto, S, O, CodErr);
end;

procedure PrimoFileInCoda(var Path: string; var CodErr: integer);
var
  S, O    : word;
  j       : integer;
begin
  Int2F(Coda, S, O, CodErr);
  if CodErr = 0 then begin
    i := 1;
    j := 1;
    CodaPtr := Ptr(S, O);
    Path := '';
    while CodaPtr^[j] <> #0 do begin
      Path := Path + CodaPtr^[j];
      Inc(j);
    end;
    i := i + 64;
  end;
end;


  procedure NextFileInCoda(var Path: string; var CodErr: integer);
  var
    j: integer;
  begin
    j := i;
    Path := '';
    while CodaPtr^[j] <> #0 do begin
      Path := Path + CodaPtr^[j];
      Inc(j);
    end;
    i := i + 64;
  end;

  procedure ContinuaStampa(var CodErr: integer);
  var
    S, O: word;
  begin
    Int2F(Sblocca, S, O, CodErr);
  end;

  end.
```

accesso a tutte le possibilità di uso di un PRINT già installato: si può in primo luogo verificare che l'installazione sia avvenuta, con la funzione *PrintInstallato*, mentre la procedura *StampaInBackground* aggiunge un file alla coda di stampa; *RimuoviFileDaCoda* e *CancellaTuttiDaCoda* tolgono un dato file, o tutti i file, dalla coda di stampa, mentre *PrimoFileInCoda* e *NextFileInCoda* consentono di verificare quali file siano in essa; poiché per fornire quest'ultimo tipo di informazioni la stampa viene congelata, con *ContinuaStampa* si può provocare la normale prosecuzione delle operazioni.

Il tutto è brevemente esemplificato nel «demo» della figura 5.

La procedura Installa

Possiamo a questo punto tornare alla nostra unit TSR, per vedere come avviene l'installazione di un programma residente. Abbiamo già anticipato diversi aspetti: l'assegnazione ad apposite variabili dei codici dei tasti che dovranno attivare il TSR, del numero di versione del DOS, dei valori dei registri SS e SP, oppure la chiamata della funzione *CercaFlag*. Accanto a queste istruzioni vi è anche quella che assegna il valore del parametro *ID* alla variabile *MultiplexID*, il cui nome vi fa indovinare che quel valore altro non è che il byte di identificazione che bisogna attribuire ad ogni programma residente. Viene quindi subito controllato che tale valore sia valido (cioè compreso tra C0h e FFh), così come si controlla che non si stia operando sotto una versione del DOS antecedente la 2.0.

A questo punto, tutto dipende da quale versione del DOS ospita il nostro TSR. Se si tratta di una versione 2.x può capitare che manchi del tutto un INT 2Fh; per questo motivo *GetIntVec2F* non è una procedura, ma una funzione che ritorna FALSE se all'INT 2Fh non è associata alcuna routine: in questo caso si usa *SetInt2FVuoto* per associare all'interrupt un semplice IRET (i relativi sorgenti sono nella figura 6, che contiene la seconda e ultima parte del file TSRINT.ASM, la cui prima parte è stata pubblicata a marzo). Le versioni 2.x del DOS comportano inoltre che programmi residenti scritti mediante la unit TSR, a causa dell'uso che questa fa dell'INT 2Fh, devono essere installati prima di PRINT.COM; si controlla quindi che PRINT.COM non sia già residente in memoria.

Completati questi preliminari, si chiama l'INT 2Fh per verificare che il nostro programma non sia già stato installato: si pone in AH il byte di identificazione e in AL uno zero; se in AL rimane zero si può procedere all'installazione. Per far questo si salvano nelle variabili che abbiamo già visto gli indirizzi delle routine associate agli interrupt 5h, 8h, 9h, 10h, 13h e 28h e si sostituiscono a queste le rispettive procedure contenute in TSR.PAS e TSRINT.ASM; si procede analogamente per l'INT 2Fh (la funzione *GetIntVec2F* ritorna ora sicuramente TRUE), associandovi una routine che, se il byte di identificazione non è quello del TSR salta alla routine originaria, *PrevInt2F*, altrimenti se AL è zero mette in AL FFh per confermare l'avvenuta installazione.

L'ultima cosa da non dimenticare è la chiamata della procedura *SwapVectors*. Ne abbiamo già parlato a proposito della unit *ExecSwap;* qui la situazione, pur se analoga, è un po' diversa. Un programma residente «non termina mai»: risiede permanentemente in memoria, pronto ad essere attivato. Un programma scritto in Turbo Pascal inizia sempre con alcune istruzioni (il codice di «start up» cui abbiamo accennato la volta scorsa), che, tra altre cose, associano ad alcuni interrupt particolari routine. Un esempio... a caso: ogni volta che viene eseguita (con le istruzioni Assembler DIV o IDIV) una divisione per zero, scatta l'INT 0. In un programma Turbo Pascal questo è associato ad una routine che provoca l'arresto dell'esecuzione e la comparsa di un messaggio di errore. Chiaramente non vogliamo che, mentre il nostro TSR dorme in sottofondo, un DIV di un altro programma provochi... l'arresto del TSR! O, cosa ben più probabile, l'impantanamento della macchina.

Per questo motivo, quando un programma Turbo Pascal termina quegli interrupt vengono rimessi a posto. Nel caso di un TSR, però, dobbiamo pensarci noi, appunto chiamando la procedura SwapVectors. Dopo di ciò, possiamo finalmente chiamare la procedura Keep per lasciare residente in memoria il nostro programma.

```
Program PrintDemo;
uses Print;
var
  NomeFile: string;
  CodiceErrore: integer;       (* ignorati per semplificare *)
begin
  if not PrintInstallato then Halt(1);
  StampaInBackground('PRINDEMO.PAS', CodiceErrore);
  StampaInBackground('PRINT.PAS', CodiceErrore);
  StampaInBackground('TSR.PAS', CodiceErrore);
  Writeln('Coda di stampa:');
  PrimoFileInCoda(NomeFile, CodiceErrore);
  while NomeFile <> '' do begin
    Write(NomeFile, ' ');
    NextFileInCoda(NomeFile, CodiceErrore);
  end;
  Writeln;
  ContinuaStampa(CodiceErrore);
  RimuoviFileDaCoda('PRINT.PAS', CodiceErrore);
  RimuoviFileDaCoda('TSR.PAS', CodiceErrore);
end.
```

Figura 5 - Un esempio di uso della unit PRINT.

```
;-------------------------------------------------
; procedure NuovoInt2F; interrupt;
NuovoInt2F      PROC    FAR
                push    ds
                push    ax
                mov     ax,SEG DATA
                mov     ds,ax
                pop     ax
                cmp     ah,MultiplexID
                pop     ds
                je      NI2F_01
                jmp     cs:PrevInt2F
NI2F_01:        or      al,al
                jnz     NI2F_02
                mov     al,0FFH
NI2F_02:        iret
NuovoInt2F      ENDP
;-------------------------------------------------
; function GetIntVec2F: boolean; external;
GetIntVec2F     PROC    NEAR
                mov     ax,352Fh
                int     21h
                mov     word ptr cs:PrevInt2F,bx
                mov     word ptr cs:PrevInt2F+2,es
                mov     ax,es
                or      ax,bx
                jz      GI2F_01
                mov     ax,1
GI2F_01:        ret
GetIntVec2F     ENDP
;-------------------------------------------------
; procedure Int2FVuoto; interrupt;
Int2FVuoto      PROC    FAR
                iret
Int2FVuoto      ENDP
;-------------------------------------------------
; procedure SetInt2FVuoto; external;
SetInt2FVuoto   PROC    NEAR
                push    ds
                push    cs
                pop     ds
                mov     dx,OFFSET Int2FVuoto
                mov     ax,252Fh
                int     21h
                pop     ds
                ret
SetInt2FVuoto   ENDP
;-------------------------------------------------
; procedure SetIntVec2F; external;
SetIntVec2F     PROC    NEAR
                push    ds
                push    cs
                pop     ds
                mov     dx,OFFSET NuovoInt2F
                mov     ax,252Fh
                int     21h
                pop     ds
                ret
SetIntVec2F     ENDP
;-------------------------------------------------
CODE            ENDS
;=================================================
                END
```

Figura 6 - La seconda e ultima parte del file TSRINT.ASM (la prima parte è stata pubblicata nel numero di marzo).

Ultimo appuntamento

Nella figura 7 trovate l'ultima parte del file TSR.PAS, la sezione di inizializzazione della unit: è talmente semplice che non credo servano ulteriori commenti.

```
begin
  CtrlBreak      := FALSE;
  CtrlC          := FALSE;
  ErroreCritico  := 0;
  InInt5         := 0;
  InInt8         := 0;
  InInt9         := 0;
  InInt10        := 0;
  InInt13        := 0;
  InInt28        := 0;
  InTSRKey       := FALSE;
  InTSRProg      := FALSE
end.
```

Figura 7 - La sezione di inizializzazione della unit TSR.

Vi do quindi direttamente appuntamento al mese prossimo. Tra trenta giorni vedremo alcuni esempi di programmi residenti realizzati con la unit, il cui scopo sarà soprattutto quello di mostrare come vanno gestite le situazioni «non facili», quelle in cui non va tutto per il verso giusto: in primo luogo, come affrontare l'eventuale verificarsi di «errori critici» (un drive aperto, una stampante senza carta, ecc.). Discuteremo anche alcune possibili estensioni e modifiche della unit.

A presto.

A500 + ESPAN L. 758.000

ies COMPUTER

SERVIZIO ASSISTENZA

XT 8088, 10MHz, 1 FD 360, 1 HD 20 MB, 512 KB TASTIERA, SK MONOCR.	L. **945.000**
AT 286, 16 MHz, 1 FD 1.2 HD 20 MB, 1024KB, TASTIERA, SK MONOCR	L. **1.390.000**
386 25 MHz, 1 FD 1.2, 1 HD 40 MB, 1024KB, TASTIERA, SK VGA	L. **2.730.000**
HD 20 MB	L. **310.000**
HD 40 MB (veloce 19 m/s)	L. **750.000**
HD 80 MB (veloce 19 m/s)	L. **1.370.000**
DRIVE 1.2 MB	L. **158.000**
DRIVE 720KB	L. **124.000**
DRIVE 1.44	L. **158.000**
CONTR. HD XT	L. **89.000**
CONTR. AT	L. **150.000**
SK HERCULES	L. **70.000**
SK CGA	L. **70.000**
SK DUAL	L. **85.000**
SK EGA	L. **145.000**
SK VGA (256 KB)	L. **260.000**
SK SUPER VGA (512 KB, 16 bit)	L. **310.000**
MONITOR	
MONOCROMATICO BIFREQUENZA	L. **190.000**
VGA MICROVITEC	L. **750.000**
MULTISINC	L. **850.000**
FAX CANON	
FAX 80	L. **1.200.000**
FAX 120	L. **1.550.000**

PREZZI IVA ESCLUSA

18 MESI DI GARANZIA SU TUTTI I PC

STAMPANTI EPSON

LX 800	L. **390.000**
LQ 500	L. **590.000**
FX 1050	L. **970.000**
LQ 1050	L. **1.350.000**
GQ 5000	L. **2.990.000**

STAMPANTI CITIZEN

120 D PLUS	L. **430.000**
SWIFT 9	L. **490.000**
15 E	L. **530.000**
SWIFT 24	**Telefonare**
PRODOT 9	**Telefonare**
PRODOT 9X	**Telefonare**

EPSON

CITIZEN

06/3651588 - 06/3651688

ATTENZIONE: per un errore tipografico, nell'inserzione pubblicitaria apparsa nel numero di marzo i prezzi sono stati indicati comprensivi di IVA. Rettifichiamo l'informazione chiedendo scusa ai lettori. Tutti i prezzi sono da intendersi IVA ESCLUSA.