Allocazione di memoria a doppio uso

Il mese scorso abbiamo visto come modificare la unit PRNSPOOL in modo che, se eseguita in modo protetto, ci metta sempre a disposizione una valida interfaccia con lo spooler del DOS. Approntando una procedura per la chiamata dell'interrupt 2Fh dal modo protetto, siamo riusciti a verificare la presenza dello spooler, a sospenderne le operazioni per leggere la coda di stampa, ad eliminare tutti i file dalla coda, a far riprendere le stampe dopo l'accesso alla coda. Vedremo ora come inviare un file allo spooler e come eliminare un particolare file dalla coda di stampa

di Sergio Polini

Per trasmettere un file allo spooler di stampa del DOS, occorre usare la sottofunzione 01h della funzione 01h dell'INT 2Fh, ponendo in DS:DX l'indirizzo di un *packet*, cioè di una struttura avente due campi: un *livello* (che deve essere zero) e l'indirizzo di una stringa ASCIIZ (senza byte di lunghezza iniziale, ma con byte nullo finale) contenente il nome del file.

In un programma eseguito in modo protetto, sia l'indirizzo della stringa che l'indirizzo del *packet* sono espressi mediante coppie segmento:offset in cui, in realtà, il segmento è un selettore; non è, quindi, un indirizzo fisico nella memoria del computer, ma un indice in una tabella di descrittori. L'interrupt 2Fh vuole, invece, coppie segmento:offset in cui il segmento sia interpretabile come componente di un indirizzo fisico.

Si potrebbe pensare di risolvere il problema accedendo al descrittore corrispondente al selettore, per leggerne il campo *Base*. Non funzionerebbe (almeno non sempre), in quanto: a) se l'indirizzo lineare ottenibile mediante la base del descrittore indicato dal selettore e l'offset fosse interpretabile come indirizzo fisico, potrebbe risultare collocato oltre il primo megabyte di memoria e, quindi, non traducibile in una coppia segmento:offset valida per il modo reale; b) se fosse abilitata la paginazione, quell'indirizzo lineare non potrebbe essere interpretato come indirizzo fisico (vi rimando, per i dettagli, alla figura 5 dell'articolo apparso nello scorso mese di dicembre).

Occorre allocare memoria nel primo megabyte; occorre, inoltre, disporre sia del suo indirizzo «reale», espresso mediante una coppia segmento:offset, sia del suo indirizzo «protetto», cioè di un selettore cui corrisponda un descrittore con base corrispondente (non uguale) al segmento «reale».

Segmenti e selettori

Ricordiamo, per quanto ben noto, che in modo reale un indirizzo non è espresso dalla *somma* di segmento e offset, né, tanto meno, da un numero a 32 bit di cui 16 espressi dal segmento e 16 dall'offset; il segmento viene moltiplicato per 16 prima di essere sommato con l'offset per ottenere l'indirizzo, dando così luogo ad un numero a 20 bit. Ne segue tra l'altro che, dato un indirizzo *RealAddr*, questo può essere espresso mediante molteplici coppie segmento:offset; potremmo dire che si ha un'equazione *Seg * 16 + Ofs = RealAddr* che, avendo due incognite, è indeterminata. A volte infatti, per poter essere certi che due indirizzi siano uguali, occorre prima ridurre entrambi ad una forma «normalizzata», in cui l'offset è compreso tra $0000 e $000F; gli indirizzi $0040:$0063 e $0000:$0463, ad esempio, sono uguali in quanto entrambi riducibili alla forma normalizzata $0046:$0003.

Nel modo protetto, il campo *Base* di un descrittore viene sommato direttamente all'offset, senza ulteriori manipolazioni; si ottiene così un indirizzo lineare, cioè un indirizzo che, se non è attivo un meccanismo di paginazione, può essere letto come indirizzo fisico.

La funzione $0100 dell'interrupt 31h permette di allocare memoria nell'ambito del primo megabyte e, quindi, accessibile sia in modo reale che in modo protetto; è sufficiente indicare nel registro BX la dimensione del blocco da allocare, espressa in paragrafi (16 byte ognuno). Se la funzione ha successo, in AX viene reso il segmento per l'accesso in modo reale, in DX il corrispondente selettore per il modo protetto. In caso di errore, il flag carry sarà settato e AX conterrà un codice d'errore: 7 in caso di danni (per fortuna alquanto rari...) alla lista dei blocchi di memoria mantenuta dal DOS, 8 se non è disponibile tanta memoria quanta ne è stata richiesta.

Abbiamo detto che in DX viene reso un selettore «corrispondente» al segmento che troviamo in AX. Nella figura 2 è illustrato un breve programma che permette di verificare tale corrispondenza.

Dopo aver assegnato un valore qualsiasi alla variabile *Num*, si alloca un paragrafo di memoria nel primo megabyte mediante la funzione 0100h dell'interrupt 31h. Se l'allocazione ha avuto successo, si assegnano alle variabili *NumSeg* e *NumSel*, rispettivamente, il segmento «reale» del blocco allocato, reso nel registro AX, ed il suo selettore, reso in DX.

Per verificare che la memoria è stata correttamente allocata, si assegna al puntatore *NumPtr* un indirizzo composto dal selettore *NumSel* e da un offset pari a zero, copiando poi nei primi due byte del blocco il valore precedentemente assegnato a *Num*. Una istruzione *Writeln* ci confermerà che l'assegnazione opera correttamente e che, quindi, *NumPtr* è un puntatore valido.

La funzione 0006h dell'interrupt 31h consente di leggere il campo *Base* del

```
Modo reale
----------

Segmento: $1234
Offset  : $5678

Indirizzo fisico:  ($1234 * $10) + $5678

         o anche:  ($1234 shl 4) + $5678

         cioè   :     $12340
                    +  $5678
                    --------
                    = $179B8


Modo protetto
-------------

Selettore: $xxxx  --->  Descrittore con campo Base = $12340
Offset   : $5678


Indirizzo lineare: $12340 + $5678 = $179B8
```

Figura 1 - Il diverso modo di calcolo di un medesimo indirizzo fisico nel modo reale e nel modo protetto (in quest'ultimo l'indirizzo lineare è anche fisico solo se non c'è paginazione).

```
Program TestSel;

uses
  DOS;

var
  Reg: Registers;
  Num: Integer;
  NumPtr: ^Integer;
  NumSeg: Word;
  NumSel: Word;
  RealAddr: Longint;
  ProtAddr: Longint;

begin
  Num := 1234;
  (* Alloca un paragrafo nel primo megabyte *)
  Reg.AX := $0100;
  Reg.BX := 1;
  Intr($31, Reg);
  if (Reg.Flags and fCarry) = 0 then begin
    NumSeg := Reg.AX;       (* segmento "reale" del blocco allocato *)
    NumSel := Reg.DX;       (* selettore del blocco allocato *)
    NumPtr := Ptr(NumSel, 0);
    NumPtr^ := Num;
    Writeln(Num, ' = ', NumPtr^);
    (* Esamina la base del segmento il cui selettore e' NumSel *)
    Reg.AX := $0006;
    Reg.BX := NumSel;
    Intr($31, Reg);
    if (Reg.Flags and fCarry) = 0 then begin
      RealAddr := NumSeg;
      RealAddr := RealAddr shl 4;            (* ind. fisico "reale" *)
      ProtAddr := Reg.CX;
      ProtAddr := ProtAddr shl 16 + Reg.DX;  (* ind. fisico protetto" *)
      Writeln(RealAddr, ' = ', ProtAddr);
    end;
    (* Rilascia la memoria allocata *)
    Reg.AX := $0101;
    Reg.DX := NumSel;
    Intr($31, Reg);
  end;
end.
```

Figura 2 - Un breve programma che verifica la corrispondenza tra segmento e settore di un'area di memoria allocata nel primo megabyte mediante la funzione 0100h dell'interrupt 31h. ▶

descrittore che abbia un indice pari ad un dato selettore, rendendone il valore nei registri CX:DX. Possiamo così verificare che il selettore *NumSel* effettivamente corrisponde al segmento *NumSeg*; ricostruiamo infatti l'indirizzo fisico *RealAddr* moltiplicando per 16 il segmento *NumSeg* e assumendo un offset pari a zero e, subito dopo, l'indirizzo fisico *ProtAddr* come *longint* la cui parte «alta» (bit da 16 a 31) sia in CX e quella «bassa» (bit da 0 a 15) in DX. Anche qui ci serviremo di un'istruzione *Writeln*, questa volta per verificare l'uguaglianza degli indirizzi così ottenuti.

Il programma termina rilasciando la memoria allocata.

Rimozione di un file dalla coda di stampa

La figura 3 illustra il metodo *Remove* della classe *TSpooler* come implementato nella unit PROSPOOL, versione per il modo protetto della unit PRNSPOOL.

Come negli altri metodi, si verifica in primo luogo che non si sia precedentemente incorsi in errori, mediante lettura del valore della variabile *Err*; se questa vale zero, si procede allocando un blocco di cinque paragrafi (80 byte) nel primo megabyte, mediante la funzione 0100h dell'INT 31h.

Si può così assegnare al puntatore *F* l'indirizzo del primo byte in tale blocco, per poi copiarvi il nome del file, passato nel parametro *FileName*. Ricordo, al riguardo, che *FileName* appartiene al tipo *TFileName*, definito come array di caratteri con indice iniziale zero e, quindi, come stringa ASCIIZ; la sintassi estesa del Borland Pascal prevede compatibilità tra array di caratteri con indice iniziale zero ed il tipo *PChar* (puntatore a carattere); la funzione *StrCopy* copia la stringa ASCIIZ puntata dal secondo parametro in quella puntata dal secondo e, grazie a tale compatibilità, possiamo usare come secondo parametro direttamente *FileName*, senza bisogno di ricavarne esplicitamente l'indirizzo (secondo uno stile di programmazione ben noto a chi lavori in C o C++).

In virtù della corrispondenza tra il segmento che troviamo in *Reg.AX* ed il selettore reso in *Reg.DX*, appena verificata, possiamo ora chiamare l'interrupt 2Fh del DOS mediante la procedura *RealModeInt2F*, assegnando 0102h al campo *EAX* del record *RMCS* (funzione 01h, sottofunzione 02h dell'interrupt

```
procedure TSpooler.Remove(FileName: TFileName);
var
  Reg: Registers;
  F: PChar;
  FSeg, FSel: Word;
begin
  if Err <> 0 then Exit;
  Reg.AX := $0100;          (* alloca dal primo megabyte ... *)
  Reg.BX := 5;              (* ... 5 paragrafi = 80 byte     *)
  Intr($31, Reg);
  if (Reg.Flags and fCarry) = 0 then begin
    FSeg := Reg.AX;
    FSel := Reg.DX;
    F := Ptr(FSel, 0);
    StrCopy(F, FileName);
    FillChar(RMCS, SizeOf(RMCS), 0);
    RMCS.EAX := $0102;
    RMCS.DS := FSeg;
    RealModeInt2F;
    Err := ErrInt31;
    if Err = 0 then
      if (RMCS.Flags and fCarry) <> 0 then
        Err := RMCS.EAX and $0000FFFF;
    Reg.AX := $0101;
    Reg.DX := FSel;
    Intr($31, Reg);
  end
  else
    Err := Reg.AX;
end;
```

Figura 3 - Il metodo TSpooler.Remove *della unit PROSPOOL.*

2Fh) e il segmento *FSeg* al campo *DS*. Occorrerebbe anche assegnare al campo *EDX* l'offset nullo, ma ciò non è necessario in quanto, come abbiamo visto il mese scorso, il record *RMCS* viene tutto azzerato prima di procedere all'avvaloramento di alcuni dei suoi campi.

Otterremo così di eliminare il file *FileName* dalla coda di stampa, oppure un codice d'errore nel 16 bit «bassi» del campo *EAX* (in cui è stato copiato il valore del registro AX dopo l'esecuzione dell'INT 2Fh).

In ogni caso, si può provvedere a rilasciare la memoria allocata.

Aggiunta di un file alla coda di stampa

Per inviare un file allo spooler si procede in modo analogo, con qualche piccola complicazione in più, a causa della necessità di gestire due indirizzi: quello del nome del file e quello del *packet*.

La figura 4 illustra il metodo *TSpooler.Submit*, in cui si inizia come già visto sia nel programma *TestSel* che nel metodo *TSpooler.Remove*, per copiare il nome del file da stampare in un blocco di memoria compreso nel primo megabyte. Se l'allocazione non ha successo, si esce subito con un codice di errore nella variabile *Err*.

Si procede poi con l'allocazione di un paragrafo per assegnarne l'indirizzo ad un puntatore *P* (puntatore al tipo *TPacket*); se l'allocazione non ha successo, si esce con un codice d'errore dopo aver rilasciato la memoria allocata precedentemente per il nome del file, altrimenti si prosegue avvalorando i campi del *packet*: *Level* avrà un valore pari a zero, *Path* sarà un puntatore costruito con il segmento «reale» del nome del file, *FSeg*, ed un offset nullo.

Si chiama quindi l'interrupt 2Fh mediante la procedura *RealModeInt2F*, dopo aver assegnato ai campi *EAX* e *DS* di *RMCS*, rispettivamente, il valore 0101h (funzione 01h, sottofunzione 01h) e il segmento «reale» del packet, *PSeg*. Anche in questo caso, ovviamente, la procedura *FillChar*, chiamata prima dell'avvaloramento dei campi di *RMCS*, assicura che il campo *EDX* abbia valore nullo, quindi uguale all'offset del *packet*.

Se non vi sono errori, il file viene inviato allo spooler per la stampa, altrimenti verrà reso un codice d'errore in *Err*; in ogni caso, si può rilasciare la memoria allocata per il *packet* prima di uscire.

E se si usasse la API di Windows?

Il Borland Pascal consente di sviluppare applicazioni che, eseguite in modo protetto, possono avvalersi di librerie a linking dinamico (DLL). Si possono così creare librerie di funzioni e procedure utilizzabili da diverse applicazione senza riprodurne il codice nei file EXE di ognu-

Il mondo a portata di modem

Come illustrato in altra parte della rivista, MC-link è diventato grande; non solo nel senso che, dopo anni di sperimentazione e poi di pratica, si presenta ormai come un ambiente consolidato e per molti indispensabile, ma anche e soprattutto perché, grazie ai cosiddetti «servizi supplementari», consente ora di avere pieno accesso ad uno scenario telematico internazionale, popolato da centinaia di migliaia di computer e da centinaia di milioni di utenti.

Non è questa la sede per illustrare le potenzialità dell'accesso a Internet, Usenet o Telnet; vi potrà tuttavia interessare qualche esempio di quanto può trovarsi curiosando qua e là per il mondo.

Ho scelto due esempi carichi per me di un significato particolare, in quanto vi riconosco progetti che avevo intrapreso tempo fa e poi abbandonato per mancanza di tempo. Per fortuna, un russo e due australiani li hanno portati avanti per me.

È bastato connettersi al sito *vtucs.cc.vt.edu* e chiedere il contenuto della directory */turbo-vision/pascal* per notare i file TVG103_0.ZIP e OODB.ZIP.

Grafica in finestra con Turbo Vision

Sappiamo tutti che Turbo Vision è una interfaccia a carattere; se si desidera fare grafica, occorre abbandonare il modo testo per ritrovarsi sul tradizionale sfondo nero del DOS: si possono tracciare punti, linee, ecc., ma si deve rinunciare al desktop, ai menu, alla linea di stato, alla possibilità di aprire diverse finestre.

È ovvio che, se ciò non piace, si può rimediare: basta sostituire in qualche modo le procedure *InitVideo*, *SetVideoMode* e *DoneVideo* della unit DRIVERS in modo che il video venga posto in modo grafico. Ma non è lavoro da poco.

Gli australiani Christopher Burke e Richard Morris lo hanno portato a termine in modo molto efficace. La loro unit TVGRAPH è molto facile da usare: basta menzionarla in una clausola **uses** prima di qualsiasi altra unit di Turbo Vision. La sezione di inizializzazione di TVGRAPH sostituisce i primi 5 byte di quelle procedure di DRIVERS con un JMP a procedure equivalenti che, però, predispongono un video EGA o VGA in modo grafico.

Le altre unit di Turbo Vision possono essere usate così come sono, senza bisogno di alcuna modifica. Al tempo stesso, però, è possibile tracciare punti e linee entro *rettangoli grafici*, cioè rettangoli con coordinate espresse in punti; un'apposita procedura *TextToGraphics* converte le coordinate di un *TRect* (25x80) in quelle di un GRect (7500x10000). Si possono così creare finestre che, pur apparendo su un desktop identico a quello delle normali applicazioni Turbo Vision, accetteranno un output grafico.

Due soli i punti deboli. TVGRAPH, scritta per il Pascal 6.0, presuppone una implementazione della unit VIEWS che potrebbe variare in diverse versioni di Turbo Vision; a quanto ho potuto verificare, tuttavia, non vi sono problemi con il Pascal 7.0. Il sorgente, inoltre, rende chiaro l'assunto e mette il programmatore in grado di far fronte ad eventuali modifiche apportate in versioni successive.

L'output grafico, inoltre, può essere disturbato dal cursore del mouse (grafico anch'esso); è stato però sufficiente aggiungere nella interfaccia della unit le due procedure *MouseCursorOff* e *MouseCursorOn*, dichiarate solo nella sezione di implementazione, per risolvere il problema.

Un database orientato all'oggetto

Ho sempre pensato che la OOP potrà affermarsi come stile di programmazione dominante solo quando si disporrà anche di database orientati all'oggetto.

Cosa sia, o dovrebbe essere, un OODB non è facile a dirsi in breve; vi rimando (almeno per il momento) a testi come *Relational Databases and Knowledge Bases* di G. Gardarin e P. Valduriez (Addison-Wesley) o al classico *Principles of Database and Knowledge-Base Systems* di J. Ullman (Computer Science Press).

OODB.ZIP contiene un elementare OODB utilizzabile in applicazioni Turbo Vision, realizzato presso l'Istituto di Matematica Applicata M.V.Keldysh di Mosca dagli studenti del Prof. Vitaly Shmatikov, ora trasferitosi all'Università di Washington a Seattle.

Il database è elementare, ma contiene alcuni elementi fondamentali: la possibilità di aggiungere ad esso oggetti qualsiasi, purché istanze di classi derivate da *TObject*, un indice derivato da *TCollection*, l'attribuzione ad ogni oggetto di una sua *identità* distinta dal suo *valore*, il supporto di *transazioni*, un meccanismo di *garbage collection*.

Non è possibile discutere ora di identità o transazioni, ma non è escluso che si torni sugli OODB in futuro.

```
procedure TSpooler.Submit(FileName: TFileName);
var
  Reg: Registers;
  F: PChar;
  FSeg, FSel: Word;
  P: PPacket;
  PSeg, PSel: Word;
begin
  if Err <> 0 then Exit;
  Reg.AX := $0100;          (* alloca dal primo megabyte ... *)
  Reg.BX := 5;              (* ... 5 paragrafi = 80 byte     *)
  Intr($31, Reg);
  if (Reg.Flags and fCarry) = 0 then begin
    FSeg := Reg.AX;
    FSel := Reg.DX;
    F := Ptr(FSel, 0);
    StrCopy(F, FileName);
    Reg.AX := $0100;        (* alloca nel primo megabyte ... *)
    Reg.BX := 1;            (* ... un paragrafo per il packet *)
    Intr($31, Reg);
    if (Reg.Flags and fCarry) = 0 then begin
      PSeg := Reg.AX;
      PSel := Reg.DX;
      P := Ptr(PSel, 0);
      P^.Level := 0;
      P^.Path := Ptr(FSeg, 0);
      FillChar(RMCS, SizeOf(RMCS), 0);
      RMCS.EAX := $0101;
      RMCS.DS := PSeg;
      RealModeInt2F;
      Err := ErrInt31;
      if Err = 0 then
        if (RMCS.Flags and fCarry) <> 0 then
          Err := RMCS.EAX and $0000FFFF;
      Reg.AX := $0101;
      Reg.DX := PSel;
      Intr($31, Reg);
    end
    else
      Err := Reg.AX;
    Reg.AX := $0101;
    Reg.DX := FSel;
    Intr($31, Reg);
  end
  else
    Err := Reg.AX;
end;
```

◀ *Figura 4 - Il metodo* TSpooler.Submit *della unit PROSPOOL.*

Figura 5 - Il programma TESTSEL (figura 2) riscritto utilizzando le funzioni della API di Windows.

▼

```
Program TestSel;

uses
  WinAPI;

type
  TDoubleSeg = record
    case Integer of
      1: (Prot: Word; Real: Word);
      2: (Both: Longint);
  end;

var
  Num: Integer;
  NumPtr: ^Integer;
  NumSeg: TDoubleSeg;
  RealAddr: Longint;
  ProtAddr: Longint;

begin
  Num := 1234;
  (* Alloca un paragrafo nel primo megabyte *)
  NumSeg.Both := GlobalDosAlloc(SizeOf(Integer));
  if NumSeg.Both <> 0 then begin
    NumPtr := Ptr(NumSeg.Prot, 0);
    NumPtr^ := Num;
    Writeln(Num, ' = ', NumPtr^);
    (* Esamina la base del segmento il cui selettore e' NumSel *)
    ProtAddr := GetSelectorBase(NumSeg.Prot);
    RealAddr := NumSeg.Real;
    RealAddr := RealAddr shl 4;
    Writeln(RealAddr, ' = ', ProtAddr);
    (* Rilascia la memoria allocata *)
    GlobalDOSFree(NumSeg.Prot);
  end;
end.
```

na, ma semplicemente salvandone una copia in un file con estensione DLL. Ogni applicazione potrà utilizzare le funzioni e procedure della libreria, stabilendo un collegamento dinamico (cioè al momento dell'esecuzione) con esse.

Non solo è possibile l'uso di DLL sviluppate con il compilatore, ma anche di quelle sviluppate con altri compilatori, in altri linguaggi. Tutto ciò che occorre è conoscere le funzione esportate dalla libreria e il loro *indice* in essa. Ne segue che è anche possibile utilizzare le DLL di Windows o create per Windows, alla sola condizione che non si tenti di chiamare funzioni e procedure che presuppongono l'ambiente grafico.

Questo vuol dire che è possibile utilizzare, in un programma compilato per il DOS in modo protetto, anche funzioni della API di Windows, comprese quelle che forniscono servizi equivalenti a quelli accessibili mediante le funzioni dell'interrupt 31h. Tra queste troviamo, in particolare, *GlobalDOSAlloc* e *GlobalDOSFree*, che corrispondono alle funzioni 0100h e 0101h dell'INT 31h e con le quali, quindi, potremmo allocare e rilasciare memoria nel primo megabyte, o *GetSelectorBase*, che rende il valore del campo *Base* del descrittore con indice pari al selettore passato come parametro.

Per chiamare le funzioni della API di Windows è sufficiente usare la unit WINAPI. A titolo di esempio, la figura 5 propone una versione del programma TESTSEL della figura 2, riscritta in modo da utilizzare le funzioni della API di Windows.

Su MC-link si è discusso della opportunità di usare direttamente l'interrupt 31h in luogo delle corrispondenti funzioni di Windows, grazie alle quali si ottengono (come è facile verificare) sorgenti più «puliti». Personalmente, preferisco usare la API di Windows quando programmo sotto Windows, l'INT 31h quando programmo sotto DOS, se non altro perché la API di Windows non comprende funzioni equivalenti a quelle che l'INT 31h offre per eseguire codice in modo reale e non posso essere sicuro che un'applicazione DOS verrà sempre eseguita su macchine su cui sia stato installato anche Windows. Ma non pretendo di avere ragione. MC

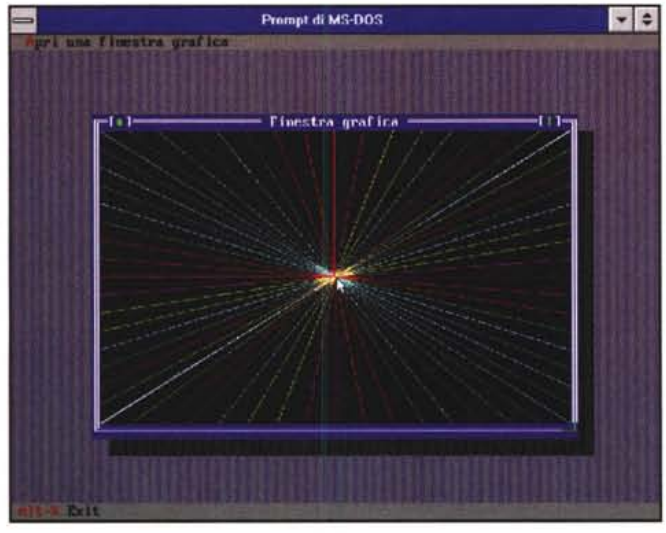

Figura 6 - Una unit TVGRAPH disponibile tramite Internet rende grafico il Turbo Vision; si può così fare grafica in una finestra, invece che su tradizionale sfondo nero del DOS, mantenendo tutte le funzionalità dell'interfaccia utente.

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