Carattere che va, carattere che viene

La volta scorsa abbiamo esaminato l'interfaccia della unit COMMWIN e, tra i metodi della classe TCommWindow, *quelli dedicati alla simulazione di una interfaccia TTY. Sfruttando alcune caratteristiche di Windows, abbiamo previsto la possibilità di selezionare il carattere tra tutti i tipi non proporzionali disponibili, facendo in modo che la finestra del programma adatti ogni volta le sue dimensioni al carattere scelto. Ora vedremo come gestire la comunicazione con il computer remoto*

di Sergio Polini

L'interfaccia della classe TCommWindow comprende ventidue metodi (diciotto pubblici e quattro privati). Sette di questi sono stati esaminati nel numero di luglio/agosto, altri cinque rimangono immutati rispetto alla prima versione del programma, illustrata nel numero di giugno; si tratta dei metodi relativi alla lettura e scrittura del file di inizializzazione (CMSaveSetup, CMRestoreSetup, ReadProfile e WriteProfile) e del metodo che apre la dialog box per l'impostazione dei parametri di comunicazione (CMCommSetup). Ci rimangono i metodi mediante i quali si apre e si chiude la comunicazione, si inviano i caratteri digitati dall'utente, si visualizzano i caratteri ricevuti dal computer remoto.

Impostazione e apertura della porta

Appena inizia l'esecuzione del programma, l'utente può selezionare tutte le opzioni dei menu tranne una: non può, ovviamente, chiudere una comunicazione non ancora aperta (il comando CMClose viene disabilitato nel constructor TCommWindow.Init, visto a luglio). Piuttosto che procedere subito ad aprire il collegamento, tuttavia, può essere opportuno verificare l'impostazione della porta, mediante l'opzione *Parametri di comunicazione* del menu Opzioni; qualora lo desideri, l'utente può cambiare l'impostazione letta dal fine di inizializzazione.

La dialog box relativa ai parametri di comunicazione viene gestita dal metodo CMCommSetup, che apre un'istanza della classe TCommSetupDlg; se l'utente la chiude premendo il pulsante «Ok», i campi della variabile CommTransBuf vengono riempiti con i valori dei controlli della dialog box. Così come il metodo ReadProfile, quindi, anche CMCommSetup non fa altro che agire su una variabile nascosta nella implementation della unit, senza intervenire sulla porta.

Quando l'utente decide di attivare il collegamento, deve scegliere l'opzione Apri del menu Collegamenti; ciò provoca l'esecuzione del metodo CMOpen (figura 1), che apre la porta specificata nel campo PortSel della variabile CommTransBuf. Se l'apertura della porta ha successo, si prosegue con il metodo SetCommParams (figura 2), che rende effettiva l'impostazione della porta secondo i valori dei campi di CommTransBuf, convertendo questi in valori dei campi di un record di tipo TDCB e chiamando la funzione SetCommState.

Se va tutto bene, viene abilitato il comando Chiudi del menu Collegamenti, mentre vengono disabilitati Apri, dello stesso menu, e Parametri di comunicazione, Salva e Ripristina del menu Opzioni.

Si chiama poi la funzione EnableCommNotification. Come visto già nel numero di maggio, mediante questa si abilita l'invio ad una data finestra (il secondo parametro) di un messaggio WM_COMMNOTIFY generato da una porta (il primo parametro); quando si riceve un messaggio WM_COMMNOTIFY, si deve esaminare Msg.LParamLo, che può essere una combinazione di uno o più dei flag CN_EVENT, CN_RECEIVE o CN_TRANSMIT. Il terzo e il quarto parametro di EnableCommNotification indicano, rispettivamente, quanti caratteri devono trovarsi nella coda di input prima che venga inviato il messaggio, e il numero di caratteri nella coda di output al di sotto del quale viene segnalata col messaggio l'opportu-

```
procedure TCommWindow.CMOpen(var Msg: TMessage);
const
  CRLF: array[0..2] of Char = #13#10;
begin
  ComPort := OpenComm(CommTransBuf.PortSel, 4096, 4096);
  if ComPort < 0 then
    MessageBox(HWindow, 'Errore apertura porta.',
               CommTransBuf.PortSel, mb_Ok or mb_IconExclamation)
  else if SetCommParams then begin
    EnableMenuItem(Attr.Menu, cm_Open,
                   mf_ByCommand + mf_Grayed + mf_Disabled);
    EnableMenuItem(Attr.Menu, cm_Close, mf_ByCommand + mf_Enabled);
    EnableMenuItem(Attr.Menu, cm_CommSetup,
                   mf_ByCommand + mf_Grayed + mf_Disabled);
    EnableMenuItem(Attr.Menu, cm_SaveSetup,
                   mf_ByCommand + mf_Grayed + mf_Disabled);
    EnableMenuItem(Attr.Menu, cm_RestoreSetup,
                   mf_ByCommand + mf_Grayed + mf_Disabled);
    EnableCommNotification(ComPort, HWindow, MAXCOLS, -1);
    EscapeCommFunction(ComPort, SETDTR);
    if (xCursor > 0) then
      EchoChars(CRLF, 2);
    MoveCaret;
  end
  else begin
    CloseComm(ComPort);
    ComPort := -1;
    MessageBox(HWindow, 'Errore impostazione porta',
               CommTransBuf.PortSel, mb_Ok or mb_IconExclamation);
  end;
end;
```

Figura 1 – Il metodo che viene eseguito quando l'utente sceglie l'opzione Apri *del menu* Collegamenti.

nità di scriverne altri; se si usa –1 per il terzo parametro, non viene usato il flag CN_RECEIVE; se si usa –1 per il quarto, non viene usato il flag CN_TRANSMIT; nel nostro caso, non si terrà conto del flag CN_TRANSMIT, ma ci si avvarrà di un metodo per intercettare il messaggio WM_COMMNOTIFY con il flag CN_RECEIVE settato; si usano quindi il numero dei caratteri in una riga dello schermo TTY come terzo parametro e –1 per il quarto.

Dopo aver «alzato» il segnale DTR mediante la funzione EscapeCommFunction, si sistema, se necessario, la posizione del cursore: qualora si stia aprendo un collegamento dopo averne condotto e chiuso un precedente, il dialogo relativo a quest'ultimo sarà ancora visibile nella finestra *e xCursor* potrà essere maggiore di zero; in questo caso, si invia una sequenza CR/LF mediante il metodo EchoChars.

Connessione con il computer remoto

Normalmente, un modem va inizializzato prima di chiamare il computer remoto. Quando si usa un normale programma di comunicazione, ciò avviene automaticamente, sulla base di alcune indicazioni fornite dall'utente una volta per tutte. W-LINK, tuttavia, è un programma ridotto all'osso e, quindi, occorre provvedere manualmente; con il mio modem, ad esempio, per prima cosa digito la stringa "AT&F0L2S0=0". Fatto questo, indico il numero di telefono di MC-link, con "ATDT4180440" (userei "ATDP4180440" se non fossi collegato ad una centralina SIP digitale).

I caratteri digitati vanno intercettati e trasmessi al modem; a ciò provvede il metodo WMChar (figura 3). Viene usata la funzione WriteComm per inviare il carattere alla porta; in caso di errore (risultato della funzione diverso dal numero di caratteri inviati – uno in questo caso), la comunicazione viene bloccata e può

```
function TCommWindow.SetCommParams: Boolean;
var
  DCB: TDCB;
  Code: Integer;
begin
  with DCB, CommTransBuf do begin
    Id := ComPort;
    Val(BaudSel, BaudRate, Code);
    Val(DataSel, ByteSize, Code);
    case ParitySel[0] of
      'N': Parity := NOPARITY;
      'D': Parity := ODDPARITY;
      'P': Parity := EVENPARITY;
      'M': Parity := MARKPARITY;
      'S': Parity := SPACEPARITY;
    end;
    if StopSel[0] = '2' then
      StopBits := TWOSTOPBITS
    else if StrLen(StopSel) = 1 then
      StopBits := ONESTOPBIT
    else
      StopBits := ONE5STOPBITS;
    RlsTimeout   := 0;
    if HShakeSel[0] = 'H' then begin
      CtsTimeout := 30;
      Flags      := fBinary or fOutxCtsFlow or fRtsFlow;
    end
    else begin
      CtsTimeout := 0;
      Flags      := fBinary or fOutX or fInX;
    end;
    DsrTimeout   := 0;
    XonChar      := #17;
    XoffChar     := #19;
    XonLim       := 4096 div 4;
    XoffLim      := DCB.XonLim;
    EofChar      := #26;
    EvtChar      := #0;
    TxDelay      := 0;
  end;
  SetCommParams := SetCommState(DCB) = 0;
end;
```

Figura 2 – Il metodo mediante il quale le scelte operate attraverso i campi della variabile CommTransBuf *vengono tradotte in effettiva impostazione della porta seriale.*

```
procedure TCommWindow.WMChar(var Msg: TMessage);
var
  NULL: TComStat absolute 0000:0000;
begin
  if (ComPort >= 0) and (WriteComm(ComPort, @Msg.WParamLo, 1) <> 1) then
    ReportError(GetCommError(ComPort, NULL));
end;
```

Figura 3 – Il metodo che provvede ad inviare alla porta i caratteri digitati dall'utente.

```
procedure TCommWindow.ReportError(ErrorCode: Word);
const
  ErrorMsg: array[1..16] of array[0..8] of Char =
    ('RXOVER','OVERRUN','RXPARITY','FRAME','BREAK','CTSTO','DSRTO',
     'RLSDTO','TXFULL','PTO','IOE','DNS','OOP','','','MODE');
  ErrorMsgLen: array[1..16] of Integer =
    (6, 7, 8, 5, 5, 5, 5, 6, 6, 3, 3, 3, 0, 0, 4);
  CRLF: array[0..2] of Char = #13#10;
var
  i: Integer;
begin
  for i := 1 to 16 do begin
    if ErrorCode and 1 = 1 then begin
      EchoChars(CRLF, 2);
      EchoChars(ErrorMsg[i], ErrorMsgLen[i]);
      EchoChars(CRLF, 2);
    end;
    ErrorCode := ErrorCode shr 1;
  end;
end;
```

Figura 4 – Il metodo con il quale si avverte l'utente di eventuali errori di comunicazione.

```
procedure TCommWindow.WMCommNotify(var Msg: TMessage);
var
  Count: Integer;
  Buf: array[0..MAXCOLS] of Char;
  ComStat: TComStat;
  Message: TMsg;
begin
  if LoWord(Msg.LParam) and cn_Receive <> cn_Receive then
    Exit;
  repeat
    Count := Abs(ReadComm(ComPort, Buf, MAXCOLS));
    if Count > 0 then
      EchoChars(Buf, Count);
    ReportError(GetCommError(ComPort, ComStat));
  until PeekMessage(Message, 0, 0, 0, PM_NOREMOVE)
        and (ComStat.cbInQue < MAXCOLS);
end;
```

Figura 5 – Il metodo WMCommNotify, *che intercetta i messaggi generati dalla ricezione di caratteri.*

essere riattivata solo chiamando la funzione GetCommError; ciò viene fatto passando il risultato di questa al metodo *ReportError* (figura 4).

Nulla da dire riguardo a quest'ultima, che utilizza il metodo *EchoChars*, sui cui torneremo tra breve. Quanto a GetCommError, invece, ricordo che a questa può essere passato un parametro variabile di tipo TComStat per ottenere informazioni aggiuntive; se interessa solo il risultato della funzione, tale parametro può essere «nullo», cosa che non può farsi con un nil, che non può essere usato quando si richiede un parametro variabile; ricorro quindi al «trucco» già illustrato a maggio: una variabile locale dichiarata come residente all'indirizzo 0:0 mediante la clausola absolute.

Il dialogo in finestra

Il metodo WMChar non si cura di visualizzare i caratteri digitati dall'utente, in quanto questi «tornano indietro» e, quindi, possono essere trattati come i caratteri inviati dal computer remoto.

Una volta attivato il meccanismo di notifica, la ricezione di un carattere costituisce un evento che genera un messaggio WM_COMMNOTIFY, ad intercettare il quale provvede il metodo illustrato nella figura 5. In esso si verifica in primo luogo se il messaggio è del tipo CN_RECEIVE; in caso affermativo, si entra in un ciclo in cui i caratteri ricevuti vengono letti e posti in una variabile Buf mediante la funzione ReadComm. Questa ritorna il numero dei caratteri letti, e tale numero è positivo se è andato tutto bene, negativo se vi sono stati errori; per questo motivo, viene usato il valore assoluto del risultato come secondo parametro del metodo EchoChars. Non si esamina il segno, in quanto GetCommError va chiamata in ogni caso: il parametro di tipo TComStat non è nullo questa volta, ma viene usato per avere notizia del numero dei caratteri presenti nella coda di input; il ciclo termina, infatti, quando si verificano contemporaneamente due condizioni: presenza di altri messaggi (rilevata mediante PeekMessage) e presenza nella coda di input di un numero di caratteri inferiore a MAXCOLS. La chiamata di PeekMessage, per inciso, rende possibile l'esecuzione di altre applicazioni durante la ricezione dei caratteri, consentendo di evitare di bloccare Windows su W-LINK.

CMOpen, ReportError e WMCommNotify chiamano tutti EchoChars. Questo metodo (figura 6) riceve un certo numero di caratteri e provvede a disporli uno per uno nell'array Screen; alcuni caratteri vengono trattati in modo particolare: un Ctrl-G provoca un segnale acustico, un backspace decrementa xCursor mentre un carriage return lo azzera, un line feed viene gestito mediante un'apposita procedura che incrementa yCursor e, se questo arriva al bordo inferiore della finestra, sposta tutte le righe verso l'alto. La variabile xCursor viene incrementata ogni volta che un carattere viene aggiunto all'array Screen, fino a che non si arriva all'ultimo carattere di una riga; completata una riga, si passa alla successiva chiamando la procedura DoLineFeed e az-

```
procedure TCommWindow.EchoChars(Chars: PChar; Count: Integer);
var
  i: Integer;
procedure DoLineFeed;
  begin
    Inc(yCursor);
    if yCursor = MAXROWS then begin
      Dec(yCursor);
      Move(Screen[1,0], Screen[0,0], (MAXROWS-1) * MAXCOLS);
      FillChar(Screen[MAXROWS-1, 0], MAXCOLS, ' ');
      InvalidateRect(HWindow, nil, False);
    end;
  end;
begin
  for i := 0 to Count-1 do begin
    case Chars[i] of
      #07: MessageBeep(0);
      #08: if xCursor > 0 then Dec(xCursor);
      #10: DoLineFeed;
      #13: xCursor := 0;
      else begin
        Screen[yCursor, xCursor] := Chars[i];
        if xCursor < MAXCOLS-1 then begin
          Inc(xCursor);
          InvalidateRect(HWindow, nil, False);
        end
        else begin
          xCursor := 0;
          DoLineFeed;
        end;
      end;
    end;
  end;
  MoveCaret;
end;

procedure TCommWindow.Paint(PaintDC: HDC; var PaintInfo: TPaintStruct);
var
  Font: HFont;
  i: Integer;
begin
  Font := SelectObject(PaintDC, CreateFontIndirect(LogFont));
  for i := 0 to MAXROWS-1 do
    TextOut(PaintDC, 0, cyChar * i, @Screen[i,0], MAXCOLS);
  DeleteObject(SelectObject(PaintDC, Font));
end;
```

Figura 6 – I metodi che sovraintendono alla visualizzazione del dialogo con il computer remoto.

zerando xCursor. Terminato il trattamento di tutti i caratteri, si aggiusta la posizione del cursore con MoveCaret.

Le righe dell'array *Screen* vengono mostrate su video dal metodo *Paint* (anch'esso nella figura 6).

Chiusura del collegamento

L'utente può chiudere il collegamento in due modi: uscendo dall'applicazione (opzione Esci del menu Collegamenti, o Chiudi del menu di sistema), o selezionando l'opzione Chiudi del menu Collegamenti; le due situazioni chiamano in causa, rispettivamente, i metodi CanClose e CMClose, i quali entrambi si limitano a chiamare il metodo CommClose.

Nel caso che l'utente chiuda il collegamento, ma non l'applicazione, è probabile che intenda attivarne successivamente un altro; si disabilita, quindi, la notifica di eventi chiamando EnableCommNotification con uno zero come secondo parametro, si abbatte il segnale DTR chiamando EscapeCommFunction, si chiude la porta con CloseComm e si nasconde il cursore portandolo in una posizione (–1, –1) esterna ai limiti della finestra; oltre a questo, però, si aggiustano anche i menu, disabilitando l'opzione Chiudi e riabilitando Apri, Parametri di comunicazione, Salva e Ripristina.

Abbiamo così terminato l'esposizione di W-LINK, in quanto il codice del programma vero e proprio rimane esattamente quello già illustrato nel numero di giugno; tutti i cambiamenti, rispetto alla versione provvisoria che vi proposi allora, sono infatti limitati alla unit COMMWIN.

```
procedure TCommWindow.CommClose;
begin
  if ComPort >= 0 then begin
    EnableMenuItem(Attr.Menu, cm_Open, mf_ByCommand + mf_Enabled);
    EnableMenuItem(Attr.Menu, cm_Close,
                   mf_ByCommand + mf_Grayed + mf_Disabled);
    EnableMenuItem(Attr.Menu, cm_CommSetup, mf_ByCommand + mf_Enabled);
    EnableMenuItem(Attr.Menu, cm_SaveSetup, mf_ByCommand + mf_Enabled);
    EnableMenuItem(Attr.Menu, cm_RestoreSetup,
                   mf_ByCommand + mf_Enabled);
    EnableCommNotification(ComPort, 0, -1, -1);
    EscapeCommFunction(ComPort, CLRDTR);
    CloseComm(ComPort);
    ComPort := -1;
    SetCaretPos(-1, -1);
  end;
end;

procedure TCommWindow.CMClose(var Msg: TMessage);
begin
  CommClose;
end;

function TCommWindow.CanClose: Boolean;
begin
  CommClose;
  CanClose := inherited CanClose;
end;
```

Figura 7 – I metodi implicati nella chiusura di un collegamento.

Spero che, seguendo gli ultimi articoli, abbiate maturato la convinzione che, a patto di usare le tecniche corrette, la gestione della comunicazione seriale sotto Windows è tutt'altro che difficile; in fondo, la maggior parte del codice di W-LINK si occupa della visualizzazione del dialogo con il computer remoto più che dell'invio e della ricezione di caratteri.

Per completare W-LINK, sarebbe necessario dotarlo della capacità di trasmettere e ricevere file, almeno con un protocollo semplice come XModem. Non l'ho fatto, per evitare di proporre lunghi listati a chi non fosse interessato ad argomenti forse un po' troppo specialistici; nulla vieta, comunque, di tornare sull'argomento se giungessero richieste non troppo sporadiche in tal senso.

Per il momento, proseguiremo sulla nostra strada: il mese prossimo vedremo come gestire la comunicazione seriale in Turbo Vision, cioè in un altro ambiente ad eventi.

Approfitterò, inoltre, del ritorno al mondo MS-DOS per proporvi un argomento di sicuro interesse: la comunicazione tra un programma MS-DOS compilato con il Pascal 7.0 per il modo protetto del processore 80386 e un programma residente compilato per il modo reale, mediante uso, in alternativa, delle funzioni dell'INT $31 rese disponibili dalle specifiche DPMI, o di alcune funzioni della API di Windows (accessibili anche da programmi DOS compilati con il Pascal 7.0, grazie alla unit WinAPI).

A presto. MC

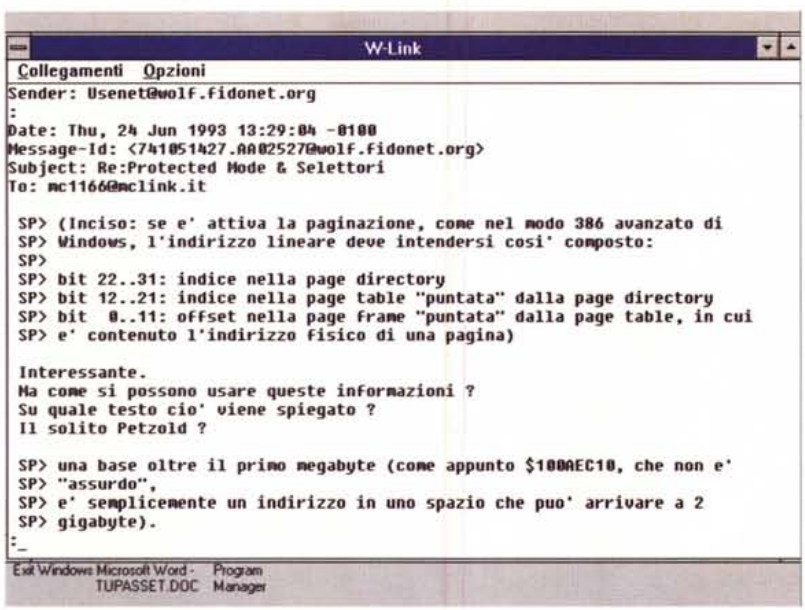

Figura 8 – Il programma W-Link usato per un collegamento con MC-link (sto leggendo un messaggio inviatomi via Internet da Massimo Penna, di Genova, nel quale si discute degli indirizzi di memoria nel modo reale e nel modo protetto dell'80386).

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