Trasferimento di files tra Client e Server (prima parte)

http://www.vbsimple.fbi/cliserv/clser_04.htm

Difficoltà: 4 / 5

Scambiare uno o più files in un'applicazione Client / Server con un protocollo di nostra progettazione può essere relativamente semplice. Basterà infatti aprire il file in maniera binaria, leggerne il contenuto in più volte e mandari i singoli pezzetti mediante il metodo ***SendData*** del controllo Winsock .

Il protocollo TCP/IP si occuperà di verificare se il pacchetto è arrivato a destinazione e comunicare la risposta all'altro capo della comunicazione.

Tuttavia in questa maniera la nostra applicazione non saprà nulla sullo stato di avanzamento del trasferimento dei files; la parte client si occuperà di inviare il file e basta, mentre la parte server salverà il file senza mandare risposte esplicite al client.

In questo esempio vedremo invece come controllare l'avanzamento del trasferimento del file mediante l'invio di piccoli pacchetti di conferma. Questo genere di pacchetti di risposta sono detti **ACK** (Acknowledgement - mettere a conoscenza).
Sfruttando i pacchetti di ACK potremo sapere esattamente quando mandare i prossimi dati e così potremo monitorare l'avanzamento del trasferimento.

Il progetto si comporrà di due parti: una Client ed una Server. Il trasferimento di files di cui ci occuperemo andrà unicamente dal client verso il server, sarò ovvero un upload. Utilizzeremo un metodo di trasferimento molto simile al protocollo FTP. Il server, dopo aver accettato il trasferimento del file, aprirà un socket passivo su una porta casuale e comunicherà il numero della porta aperta al Client. Tutti i dati che arriveranno sul socket passivo saranno salvati all'interno del file.

Prima di vedere l'interfaccia utente ed il codice definiamo un nostro protocollo, un set di regole, per far comunicare queste due applicazioni.

Il Client potrà inviare soltanto due comandi:

- `/FILE Nome Dimensione`
  Richiede il trasferimento di un file.
  Dove ***Nome*** è il nome del file, senza percorso ma con estensione e ***Dimensione*** è la dimensione in bytes del file da inviare.

- `/FINE`
  Termina il trasferimento corrente.

Il Server da parte sua avrà altri comandi unicamente di risposta:

- `-ERR: NON ACCETTATO`
  Indicherà che il server ha rifiutato il file da trasferire.

- `+OK: PORTA NumeroPorta`
  Il server ha accettato il trasferimento del file ed ha aperto un socket passivo in ascolto per il trasferimendo dei dati.
  Dove ***NumeroPorta*** è il numero della porta messa in ascolto per il socket passivo di trasferimento.

- `+OK: RECV Dimensione`
  Il pacchetto ACK di risposta per il ricevimento dei dati.
  Dove ***Dimensione*** è la dimensione in bytes dei dati finora ricevuti.

- `+OK: FINE`
  Indica la fine del trasferimento del file.

Il procedimento da seguire è molto semplice: il server apre un socket in attesa su una porta specifica a cui il client si collegherà. Stabilita la connessione il client invia la richiesta di trasferimento di un file mediante il comando `"/FILE nomefile x"`. Il server potrà decidere se accettare il file o meno: in caso di non accettazione risponderà con il messaggio `" -ERR: NON ACCETTATO"`.
Se il server accetta il file si occuperà di aprire un socket passivo su una porta specifica e comunicherà il numero della porta mediante la risposta `" +OK: PORTA x"`.
Il client attenderà tale risposta. Arrivata la risposta leggerà il numero della porta aperta e si connetterà ad essa mediante un socket di trasferimento.

A questo punto la connessione di trasferimento è stabilita: il client potrà leggere un primo pacchetto di dati dal file ed inviarlo mediante il nuovo socket aperto collegato al socket passivo sul server.

Inviato il primo pacchetto il client rimarrà in attesa del pacchetto di ACK di risposta. Il server, al ricevimento dei dati sul socket passivo, li memorizzerà nel file di destinazione ed invierà il pacchetto di ACK `" +OK: RECV x"` con il numero di bytes ricevuti.
All'arrivo dell'ACK il client leggerà un altro pacchetto di dati dal file e ripeterà l'operazione fino alla conclusione del file.

Il server, al ricevimento dell'ultimo pacchetto non dovrà rispondere con una risposta di ACK ma in un altro modo, per non far richiedere dati oltre la fine del file. Il pacchetto che invierà sarà `" +OK: FINE"` per indicare la fine del trasferimento e chiuderà il socket passivo.
Il client al ricevimento del pacchetto di fine chiuderà il file ed il socket aperti.

Se il client vuole interrompere l'invio del file gli basterà mandare il messaggio `/FINE` al cui ricevimento il server associerà la chiusura del file e del socket passivo.

Cominciamo a vedere il Client, la parte più semplice del progetto:

Il form si compone di pochi e semplici controlli: due *CommandButton* di nome **Connetti** ed **Invia**, due *TextBox* di nome **IndirizzoIP** e **NomeFile** e due

controlli *Winsock* di nome **Socket** e **SocketInvio**.

Il funzionamento sfiora il ridicolo per la sua semplicità: l'utente inserirà l'indirizzo IP del server e premerà il tasto Connetti per stabilire la connessione. Fatto questo inserirà il nome di un file nella casella NomeFile e premerà il pulsante Invia per richiedere il trasferimento al server. L'utente ha finito e non dovrà fare altro che attendere la fine del trasferimento.

Il codice non si presenta molto complesso:

```
1. Option Explicit
2. Private Declare Function getsockopt Lib "wsock32.dll" (ByVal s As Long, ByVal Level
   As Long, ByVal optname As Long, optval As Any, optlen As Long) As Long
3. Private Const SO_SNDBUF = &H1001
4. Private Const SOL_SOCKET = &HFFFF&
5.
6. Private FILEHANDLE As Integer
7. Private DIMENSIONEPACCHETTO As Long
8.
```

Al fine di evitare alcuni problemi legati all'invio di una quantità di dati superiore a quella trasferibile in un singolo pacchetto con un socket TCP/IP, abbiamo dichiarato la funzione API ***getsockopt***, che utilizzeremo più avanti in combinazione con le due costanti API **SO_SNDBUF** e **SOL_SOCKET** allo scopo di ottenere la massima quantità di dati in bytes trasferibili in un singolo pacchetto.

Alla riga 6 abbiamo definito una variabile chiamata **FILEHANDLE** che utilizzeremo per memorizzare l'handle del file aperto da trasferire.
Alla riga 7 definiamo la variabile **DIMENSIONEPACCHETTO** che conterrà il numero di bytes da trasferire per ogni pacchetto.

```
 9. Private Sub Connetti_Click()
10.     If Socket.State <> sckClosed Then Socket.Close
11.     Socket.LocalPort = 0
12.     Socket.Connect IndirizzoIP.Text, 1500
13. End Sub
14.
```

Nel momento in cui l'utente clicca sul pulsante Connetti dovrà essere avviata la connessione con il server in attesa. Prima di fare questo, però, è necessario terminare un'eventuale precedente connessione (riga 10) e, molto importante, scegliere un'altra porta da utilizzare per collegarsi, poiché la precedente connessione avrebbe potuto lasciare la porta utilizzata nello stato WAIT_STATE (vedi informazioni sugli stati di NETSTAT), impostando la porta locale a 0 (riga 11). Solo allora potrà essere avviata la connessione con il server, il cui indirizzo è specificato nella TextBox **IndirizzoIP** sulla porta 1500.

```
15. Private Sub Form_Unload(Cancel As Integer)
16.     If SocketInvio.State <> sckClosed Then SocketInvio.Close
17.     If Socket.State <> sckClosed Then Socket.Close
18.     Socket.LocalPort = 0
19. End Sub
20.
```

Al momento della chiusura del form verranno chiusi tutti i socket aperti.

```
21. Private Sub Invia_Click()
22.     Dim POSIZIONE As Integer
23.     If Socket.State <> sckConnected Then Exit Sub
```



```
24.     If Dir(NomeFile.Text) = "" Then
25.         MsgBox "Il file non esiste!", vbCritical + vbOKOnly
26.     Else
27.         POSIZIONE = 0
28.         While InStr(POSIZIONE + 1, NomeFile.Text, "\") > 0
29.             POSIZIONE = InStr(POSIZIONE + 1, NomeFile.Text, "\")
30.         Wend
31.         Socket.SendData "/FILE " & Mid(NomeFile.Text, POSIZIONE + 1) & " " &
    FileLen(NomeFile.Text)
32.     End If
33. End Sub
34.
```

Il click sopra il pulsante Invia effettuerà l'invio della richiesta di trasferimento del file, ma prima di farlo è necessario controllare che la connessione con il server sia attiva (riga 23) e che il file specificato nella casella di testo **NomeFile** esista effettivamente (riga 24). Se esso non dovesse esistere sarà mostrato un messaggio di errore (riga 25). Per una spiegazione del metodo di verifica dell'esistenza di un file consultare l'HowTo dedicato.

Soltanto quando abbiamo la certezza che il file da inviare esista effettueremo la richiesta di trasferimento. La richiesta è una stringa formata da `"/FILE Nomefile Dimensione"`. Quello che invieremo nella richiesta di trasferimento sarà il semplice nome del file, senza alcun percorso fisico della cartella. Sarà pertanto necessario estrarre il nome del file dalla locazione specificata nella casella di testo **NomeFile**. Per fare questo utilizzeremo una variabile di nome **POSIZIONE**. Essa definirà la posizione finale dei dati da eliminare dal percorso del file. Per fare ciò utilizzeremo un semplice ciclo *While* con l'utilizzo dell'istruzione *Instr* (righe 27-30). Se si utilizza Visual Basic 6 sarà possibile utilizzare l'istruzione *InstrRev* per semplificare enormemente il problema.

Trovato il punto della stringa in cui inizia il nome del file vero e proprio, sarà mandata la richiesta di trasferimento; la dimensione del file da inviare sarà ritrovata mediante l'utilizzo della funzione *FileLen* (riga 31).

Se il server risponderà positivamente alla richiesta di trasferimento, sarà mandato un messaggio specifico che vedremo fra poco da cui il nostro client recupererà il numero di porta del socket passivo a cui connettersi mediante il socket **SocketInvio**.

```
35. Private Sub SocketInvio_Connect()
36.     Dim DATI(511) As Byte
37.     FILEHANDLE = FreeFile
38.     Open NomeFile.Text For Binary As FILEHANDLE
39.     Get FILEHANDLE, , DATI
40.     SocketInvio.SendData DATI
41.     Call getsockopt(SocketInvio.SocketHandle, SOL_SOCKET, SO_SNDBUF,
    DIMENSIONEPACCHETTO, Len(DIMENSIONEPACCHETTO))
42.     If DIMENSIONEPACCHETTO = 0 Then DIMENSIONEPACCHETTO = 8192
43. End Sub
44.
```

Al momento della connessione tra **SocketInvio** ed il socket passivo del server viene aperto il file da trasferire in modalità binaria (riga 38), viene letto un primo pacchetto di dati della grandezza di 512 bytes al solo scopo di avviare il trasferimento vero e proprio che sarà gestito da una procedura successiva. Il buffer letto sarà poi inviato mediante **SocketInvio** (righe 39 e 40).

Alla riga 41 abbiamo utilizzato la funzione API ***getsockopt*** per ritrovare la dimensione massima in bytes per ogni singolo pacchetto, senza che questo venga spezzettato in due o più parti, con effetti distruttivi. La dimensione massima per pacchetto trasferibile verrà salvata nella variabile **DIMENSIONEPACCHETTO** (riga 41).
Se l'operazione non è andata a buon termine la varibiale **DIMENSIONEPACCHETTO** conterrà il valore 0; in tal caso specificheremo la dimensione di default, ovvero 8192 bytes.

```
45. Private Sub SocketInvio_Close()
46.     Close FILEHANDLE
47.     If SocketInvio.State <> sckClosed Then SocketInvio.Close
48. End Sub
49.
```

Alla chiusura della connessione tra SocketInvio ed il socket passivo sul server, verranno chiusi l'handle del file ed il socket di trasferimento (righe 46 e 47).

Quella che segue è il cuore del programma. Il socket utilizzato per la connessione iniziale servirà per ricevere le risposte dal server. Le risposte che esso dovrà verificare saranno 3: l'accettazione del file da parte del server, l'ACK di trasferimento ed il termine del trasferimento del file. La procedura che gestirà queste funzioni sarà ovviamente quella legata all'evento *DataArrival* sul primo socket.

```
50. Private Sub Socket_DataArrival(ByVal bytesTotal As Long)
51.     Dim DATI() As Byte
52.     Call Socket.GetData(DATI)
53.     DATI = StrConv(DATI, vbUnicode)
```

I dati arrivati saranno estratti mediante il metodo GetData e successivamente convertiti in Unicode per essere utilizzati validamente come stringhe da analizzare.

```
54.     If Left(DATI, 11) = " +OK: PORTA" Then
55.         If SocketInvio.State <> sckClosed Then SocketInvio.Close
56.         SocketInvio.LocalPort = 0
57.         SocketInvio.Connect Socket.RemoteHostIP, Mid(DATI, 12)
58.     End If
```

Se i primi 11 caratteri dei dati riportati sono la stringa `" +OK: PORTA"`, ciò significherà che il server ha accettato il trasferimento del file, ha aperto una porta specifica per il socket passivo ed ha inviato il numero della porta aperta. L'operazione che dovremo svolgere sarà quella di chiudere ogni eventuale socket di trasferimento aperto precedente (riga 55), impostare una porta locale automatica (riga 56) e collegare **SocketInvio** con lo stesso server a cui è collegato il primo socket, ma sulla porta specificata nella stringa ricevuta (riga 57)

```
59.     If (Left(DATI, 10) = " +OK: RECV") Then
60.         If FILEHANDLE <> 0 Then
61.             DATI = Space(DIMENSIONEPACCHETTO / 2)
62.             Get FILEHANDLE, , DATI
63.             SocketInvio.SendData DATI
64.         End If
65.     End If
```

Se i primi 10 caratteri della risposta ricevuta corrispondono alla stringa `" +OK: RECV"`, avremo ricevuto l'ACK di trasferimento. Sarà necessario inviare un nuovo pacchetto di dati al server collegato.

Alla riga 60 abbiamo voluto aggiungere un controllo che non dovrebbe essere necessario ma può essere utile in quei casi in cui la linea è molto lenta oppure i pacchetti per qualche motivazione vengono spezzettati. La lettura e l'invio di dati del file sarà effettuata soltanto se l'handle del file aperto è diverso da 0.

Alla riga 61 viene allocato un buffer di dimensione in bytes specificata dalla variabile **DIMENSIONEPACCHETTO**. L'allocazione dei dati viene effettuata mediante l'utilizzo della funzione Space; ma poiché la funzione crea una stringa e tutte le stringhe in Visual Basic sono codificato secondo lo standard UNICODE a doppio byte, la lunghezza della stringa da creare sarà dimezzata.

Alle righe 62 e 63 viene effettuata la lettura dei dati dal file aperto e l'invio degli stessi mediante **SocketInvio**.

```
66.     If (Left(DATI, 10) = " +OK: FINE") Then
67.         Close FILEHANDLE
68.         SocketInvio.Close
69.         MsgBox "Trasferimento completato!", vbInformation + vbOKOnly
70.     End If
71. End Sub
```

L'ultima risposta da verificare riguarda la chiusura della connessione dal lato server per completata ricezione del file aspettato. Se quindi la stringa ricevuta è `" +OK: FINE"` saranno chusi l'handle del file ed il socket di trasferimento **SocketInvio**.
Sarà mostrato anche un messaggio informativo del completamento del trasferimento.

Il nostro client termina qui. I comandi sono pochi e semplici ed il programma non si occupa di controllare eventuali errori di connessione o di altro genere. Esso presuppone che tutte le operazioni siano svolte in maniera corretta.

Inoltre la dimensione per pacchetto di dati da inviare sarà controllata soltanto sul client, ma non verrà controllata la dimensione di ricezione del server.

Segue parte 2 >>

Fibia FBI
2 Giugno 2001
Corretto il 25 Gennaio 2004

Torna all'indice Client/Server

