Dimostrazione del Nagle Algorithm

http://www.vbsimple.net/cliserv/clser_06.htm

Difficoltà: 2 / 5

Questo semplice esempio era stato scritto per dimostrare ad un amico l'azione del *Nagle Algorithm*, definito nella RFC 896, ma poiché la dimostrazione può interessare più di una persona ho deciso di renderlo pubblico all'interno di VBSimple.
Vedi anche TCP/IP Le risposte alle domande più frequenti.

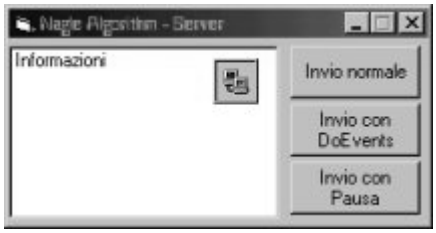
Il TCP/IP possiede alcuni naturali *'mezzi di difesa'* contro l'inondazione della rete con piccoli pacchetti. Tali difese sono maggiormente presenti all'invio dei dati da un punto di rete ad un altro. La principale di queste soluzioni di protezione è il Nagle Algorithm, un codice presente nello stack TCP/IP con lo scopo di ritardare l'invio di un nuovo pacchetto di dati fintanto che il TCP/IP non ha ricevuto l'ACK (Acknowledgment) per il pacchetto precedente.

Ciò determina che quando un programma prova ad inviare due o più pacchetti rapidamente, il primo viene inviato correttamente, qualunque sia la sua dimensione, mentre tutti i rimanenti pacchetti sono conservati e raggruppati in un unico pacchetto che sarà inviato soltanto dopo il ricevimento dell'ACK del primo pacchetto e **comunque** non prima di 200 millisecondi dall'invio del primo pacchetto.

Se ne deduce che non è possibile inviare ad un solo socket più di 5 pacchetti per secondo. Tutti gli eventuali pacchetti inviati tra una pausa e l'altra saranno raggruppati in uno solo e trasmessi allo scattare del momento giusto. Inoltre non sarà possibile inviare più di un piccolo pacchetto senza ACK.

Tale regola vale però soltanto per i pacchetti piccoli; tutti i pacchetti di dimensione uguale o maggiore a quella trasportabile con un singolo invio (costante API SO_SNDBUF - Vedi anche Trasferimento di files tra Client e Server) saranno inviati immediatamente, senza la normale attesa.

Vediamo un progettino molto semplice che dimostra quanto finora detto; un programma client si collegherà ad un programma server che dopo la connessione invierà una serie di numeri al client. Il client riprodurrà tutti i pacchetti di dati ricevuti in una ListBox e terrà il conto del numero di pacchetti ricevuti.



Il server si compone di un form con una *ListBox* di nome **Informazioni**, un controllo *Winsock* di nome **Socket** con la proprietà *LocalPort* impostata a 1500 e tre *CommandButton* di nome **InvioNormale**, **InvioDoEvents** e **InvioSleep** e riproducono tre test che effettueremo per dimostrare l'azione dell'algoritmo. Il codice è altrettanto semplice:


```
 1. Option Explicit
 2. Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
 3.
 4. Private Sub Form_Load()
 5.     Socket.Listen
 6.     Me.Caption = "Attesa connessione..."
 7. End Sub
 8.
```

Alla riga 2 abbiamo dichiarato la Sub API ***Sleep*** che utilizzeremo per effettuare una pausa all'interno del nostro progetto. Sarà vista più avanti.

All'avvio del form il socket sarà messo in ascolto e sarà modificata la *Caption* del form.

```
 9. Private Sub Informazioni_DblClick()
10.     Informazioni.Clear
11. End Sub
12.
```

Per ben valutare i dati del nostro esempio abbiamo aggiunto questa funzioncina che cancella il contenuto della ListBox **Informazioni** al doppio click dell'utente su di essa.

```
13. Private Sub Socket_ConnectionRequest(ByVal requestID As Long)
14.     Socket.Close
15.     Socket.Accept requestID
16.     Me.Caption = "Connessione accettata! Inviare i dati."
17. End Sub
18.
```

All'arrivo di una richiesta di connessione, essa sarà subito accettata e la *Caption* del form sarà modificata.

```
19. Private Sub Socket_SendProgress(ByVal bytesSent As Long, ByVal bytesRemaining As
    Long)
20.     Informazioni.AddItem bytesSent & " bytes inviati"
21. End Sub
22.
```

Tutti i pacchetti inviati tramite **Socket** saranno registrati nella ListBox **Informazioni**.

Le funzioni che seguono sono i nostri tre test di dimostrazione del Nagle Algorithm.

```
23. Private Sub InvioNormale_Click()
24.     Dim i As Integer
25.     For i = 1 To 8192
26.         Socket.SendData CStr(i)
27.     Next i
28. End Sub
29.
```

Il primo è un invio normale, non controllato e senza alcuna attesa dei numeri compresi tra 1 8192, trasformati in stringa.

```
30. Private Sub InvioDoEvents_Click()
31.     Dim i As Integer
32.     For i = 1 To 500
33.         Socket.SendData CStr(i)
34.         DoEvents
35.     Next i
36. End Sub
37.
```


Il secondo test effettua l'invio dei numeri da 1 a 500 con l'utilizzo dell'istruzione DoEvents, da molti considerata la soluzione a tanti problemi.
Essa in realtà non risolve il problema ma forza il rilascio dei dati del Winsock e rallenta leggermente l'esecuzione del programma.

```
38. Private Sub InvioSleep_Click()
39.     Dim i As Integer
40.     For i = 1 To 50
41.         Socket.SendData CStr(i)
42.         DoEvents
43.         Sleep 201
44.     Next i
45. End Sub
```

Il terzo test invece è mirato ad evitare il Nagle Algorithm a patto di possedere una linea di trasferimento veloce. Infatti, dopo l'invio, confermato con l'istruzione DoEvents, tra ogni pacchetto forzeremo un'attesa di oltre 200 millisecondi facendo scattare il tempo di raccolta dei dati. L'attesa viene generata tramite chiamata della Sub API *Sleep*. Vedremo l'applicazione dei tre test più avanti, dopo aver analizzato il semplicissimo client che si collegherà al nostro server.

Il programma si compone di un form con soli quattro controlli: una *ListBox* di nome **Informazioni**, un controllo *Winsock* di nome **Socket**, un *CommandButton* di nome **Connetti** ed una *TextBox* di nome **IndirizzoIP**.

Il funzionamento è assolutamente semplicissimo: l'utente inserisce l'indirizzo IP nella casella apposita e preme il pulsante **Connetti**. Il codice è altrettanto semplice:

```
 1. Option Explicit
 2.
 3. Private Sub Connetti_Click()
 4.     Socket.Connect IndirizzoIP.Text, 1500
 5.     Me.Caption = "0"
 6. End Sub
 7.
```

Alla pressione del pulsante **Connetti** il socket si collegherà all'host specificato nella *TextBox* **IndirizzoIP** sulla porta 1500 e la Caption del form segnerà il numero 0.

```
 8. Private Sub Informazioni_DblClick()
 9.     Informazioni.Clear
10.     Me.Caption = "0"
11. End Sub
12.
```

Anche nel client il doppio click sulla ListBox **Informazioni** ne azzererà il contenuto e riporterà il conteggio nella Caption del form a 0.

```
13. Private Sub Socket_DataArrival(ByVal bytesTotal As Long)
14.     Dim DATI As String
15.     Me.Caption = CStr(Val(Me.Caption) + 1)
```



```
16.     Socket.GetData DATI
17.     Informazioni.AddItem Now & " - " & DATI
18. End Sub
```

All'arrivo di dati sul socket, il numero contenuto nella *Caption* sarà aumentato di 1, i dati in arrivo saranno estratti mediante il metodo ***GetData*** e saranno inseriti nella *ListBox* **Informazioni** assieme all'orario di arrivo.

Possiamo passare adesso alla dimostrazione pratica dell'azione del Nagle Algorithm.

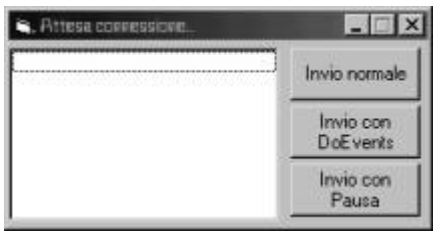


Figura 3

Figura 4

Avviamo i due progetti (nel nostro caso sullo stesso computer, in modo da evitare tutti gli eventuali ritardi di trasferimento dei dati), inseriamo il numero IP del server nel nostro Client e premiamo il pulsante ***Connetti***.

Se il server ed il client sono in esecuzione sullo stesso computer specificare l'indirizzo di loopback ***127.0.0.1***. Stabilita la connessione possiamo passare ai tre test.

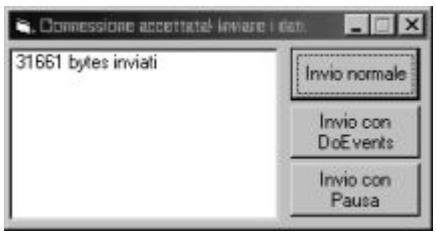


Figura 5

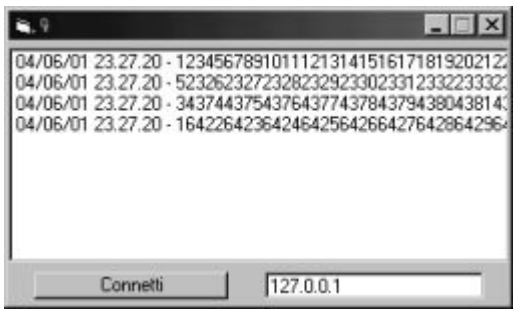


Figura 6

Il primo test consiste nell'inviare i primi 8192 numeri senza effettuare alcuna pausa. Possiamo notare che l'invio dei dati è fatto in un solo pacchetto; infatti sono inviati in un solo pacchetto 31661 bytes, eccedendo quindi il limite massimo di dati in un pacchetto.

Sul client infatti i dati non arrivano in un solo pacchetto, ma arrivano 4 pacchetti (vedi *Caption* del form) molto grossi nello stesso momento.

Il Nagle Algorithm non ha avuto applicazione poiché i dati sono stati inviati in un solo grande pacchetto, che poi il TCP/IP ha frammentato nel trasporto.

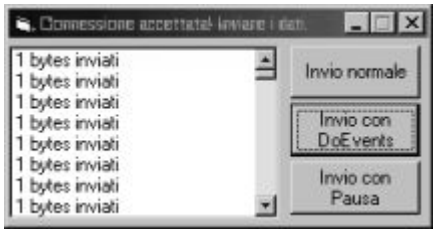


Figura 7

Figura 8

Il secondo test consiste nell'inviare al client i primi 500 numeri e richiamare dopo ogni invio l'istruzione DoEvents che permette al controllo Winsock di svolgere la sua azione di invio, che non ha potuto effettuare nel test precedente.

Così infatti accade che i dati dal server sono inviati in maniera regolare, ovvero byte per byte a secondo dell'ampiezza del numero (1 byte per i numeri compresi tra 1 e 9, 2 bytes per i numeri compresi tra 10 e 99, etc..).

Tuttavia, i dati ricevuti dal client sono totalmente differenti da quelli inviati. Possiamo notare un primo pacchetto di un solo byte e tanti altri pacchetti di più bytes. I 31661 bytes inviati sono stati separati in 28 pacchetti, di cui il primo costituito da un solo byte.

Tutto questo perché il Nagle Algorithm è entrato in azione. Esso infatti stabilisce che può essere inviato soltanto 1 pacchetto (di qualsiasi dimensione) senza che sia ritornato l'ACK di tale pacchetto. Pertanto, nell'attesa dell'ACK e del trascorrere dei 200 millisecondi, il buffer di dati da inviare si ingrandisce concatenando i numeri da inviare.

Possiamo anche notare, scrollando la lista, che non saranno inviati più di 5 pacchetti per secondo, proprio a causa della necessaria attesa di 200 millisecondi tra ogni pacchetto.

Figura 9

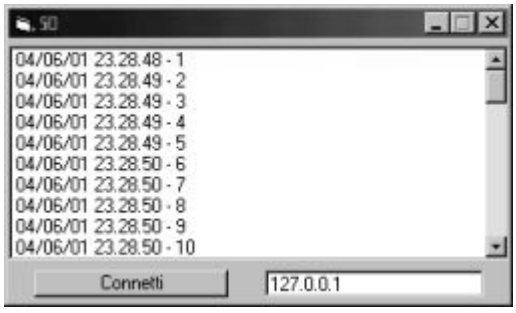


Figura 10

L'ultimo test servirà invece per evitare l'azione del Nagle Algorithm (almeno su linee veloci). Infatti tra un invio e l'altro sarà eseguita l'istruzione DoEvents che permette al socket di inviare i dati (come abbiamo visto nel test precedente) ed effettueremo anche una pausa di 201 millisecondi, per far scattare il limite di attesa minimo per ogni pacchetto. Poiché il test è ovviamente più lento a causa delle pause forzate, i numeri che saranno inviati andranno da 1 a 50.

Il test, nel nostro esempio, ha inviato i dati nella maniera corretta. Il server ha inviato i dati

byte per byte a seconda della dimensione del numero da trasferire. Il client ha ricevuto i dati nella maniera corretta. Sono arrivati infatti 50 pacchetti ed i numeri sono rappresentati nel formato originale, senza alcun raggruppamento.

Infatti nei 201 millisecondi di pausa l'ACK ha avuto il tempo di arrivare e non sono stati inviati più di 5 pacchetti per secondo; pertanto tutti i dati inviati sono arrivati nella maniera originale.

I tre test dimostrano le tre modalità di invio dei dati: consecutivamente, mediante DoEvents e con forzatura d'attesa.

Sebbene in molti casi l'utilizzo di un semplice DoEvents può bastare a regolare il flusso di dati tra client e server, in questo caso è palese che non è assolutamente sufficiente inserire un DoEvents per risolvere tutti i problemi di sincronizzazione.

È possibile disabilitare l'azione del ***Nagle Algorithm*** mediante l'utilizzo della funzione API ***setsockopt***, combinata con la costante API **TCP_NODELAY**.

Questo esempio serviva soltanto a dimostrare l'esistenza ed il funzionamento dell'algoritmo di protezione dall'inondamento di dati nella rete e non a fornire un'efficiente soluzione per evitarlo.

Fibia FBI
4 Giugno 2001

Torna all'indice Client / Server

