Effettuare un Ping tramite WSA e ICMP.DLL

http://www.vbsimple.net/cliserv/clser_11.htm

Difficoltà: 3 / 5

Il termine **Ping** è ormai entrato nel dizionario comune di qualsiasi sistemista e molti utenti di computer; il termine tecnico di questo genere di operazione è **ICMP Echo** e si tratta di una richiesta (*ECHO Request*) fatta mediante il protocollo ICMP (*Internet Control Message Protocol*) a cui il destinatario deve rispondere (*ECHO Response*) con gli stessi dati ricevuti. Per questa ragione la funzione prende il nome di ICMP Echo (Eco) e la sua implementazione più comune è data dal programma *PING*.

Il progetto sviluppato in questo articolo utilizza le classi clsFBISocket e clsFBISocketInfo trattate negli articoli precedenti e vi aggiunge una semplice richiesta ICMP Echo e tutto ciò che vi gira attorno (tempo di risposta, dati di ritorno, codici di errore). L'attività di Echo è demandata alla libreria ICMP.DLL presente in quasi tutti i sistemi operativi Windows.

Microsoft scoraggia l'uso della libreria ICMP.DLL e da qualche anno avvisa che nelle versioni future del sistema operativo potrebbe essere assente; tuttavia fino all'ultima attuale versione di Windows (Windows 2003 Server) la libreria è ancora presente. Anche per questa ragione la documentazione sulle funzioni della libreria ICMP sono poche e confuse. Microsoft consiglia invece di utilizzare i cosiddetti socket grezzi (**RAW Sockets**) disponibili a partire da Windows Socket 2.0. L'implementazione tuttavia richiede notevoli complicazioni che al giorno d'oggi è possibile evitare.

Vediamo quindi il codice della classe **clsFBISocketPing**, come già detto basata sulle funzionailtà di ICMP.DLL:

```
 1. Option Explicit
 2.
 3. Private Type ICMP_OPTIONS
 4.     TTL As Byte
 5.     Tos As Byte
 6.     Flags As Byte
 7.     OptionsSize As Byte
 8.     OptionsData As Long
 9. End Type
10.
11. Private Type ICMP_ECHO_REPLY
12.     Address As Long
13.     Status As Long
14.     RoundTripTime As Long
15.     DataSize As Integer
16.     Reserved As Integer
17.     DataPointer As Long
18.     Options As ICMP_OPTIONS
19.     Data As String * 256
20. End Type
21.
```


Le due strutture ICMP_OPTIONS e ICMP_ECHO_REPLY sono ricavate da informazioni trovate sul web e da numerose prove. Trovare una documentazione ufficiale su queste non è cosa affatto facile. La prima struttura ***ICMP_OPTIONS*** definisce le opzioni relative all'invio (ECHO Request) ed alla risposta (ECHO Response). Il più importante di questi valori è il campo TTL (*Time To Live*) che definisce il numero massimo di salti che il pacchetto potrà fare, ovvero il numero massimo di router che il pacchetto potrà attraversare prima di essere scartato perché considerato scaduto. Nel nostro esempio utilizzeremo il valore massimo 255.

L'altro tipo di dati è ***ICMP_ECHO_REPLY*** e costituisce il pacchetto di ritorno dalla funzione che effettuerà il ping. Al suo interno contiene l'indirizzo IP decimale dell'host (*Address*) che ha risposto alla richiesta, un codice di ritorno (*Status*), il tempo impiegato dal pacchetto (*RoundTripTime*), i dati restituiti (*Data*) e la loro ampiezza (*DataSize*), assieme alle opzioni con cui il pacchetto di ritorno dall'host è stato inviato (*Options*).

```
22. Private Declare Function IcmpCreateFile Lib "ICMP.DLL" () As Long
23. Private Declare Function IcmpCloseHandle Lib "ICMP.DLL" (ByVal IcmpHandle
    As Long) As Long
24. Private Declare Function IcmpSendEcho Lib "ICMP.DLL" (ByVal IcmpHandle As
    Long, ByVal DestinationAddress As Long, ByVal RequestData As String, ByVal
    RequestSize As Integer, RequestOptions As ICMP_OPTIONS, ReplyBuffer As
    ICMP_ECHO_REPLY, ByVal ReplySize As Long, ByVal Timeout As Long) As Long
25.
26. Private WithEvents fbiSocket As clsFBISocketInfo
27. Private echoReply As ICMP_ECHO_REPLY
28. Private icmpOptions As ICMP_OPTIONS
29. Private m_Timeout As Long
30.
31. Public Event Error(ByVal ErrorCode As Long, ByVal Description As String,
    ByRef Cancel As Boolean)
32.
```

Affinchè sia possibile inviare il pacchetto di richiesta è necessario ottenere prima un handle e ciò può essere fatto richiamando la funzione **IcmpCreateFile**. Lo stesso handle deve essere rilasciato mediante la funzione **IcmpCloseHandle**. Il cuore di tutto questo articolo è invece la terza funzione: **IcmpSendEcho**, che richiede il numero di handle (*IcmpHandle*), l'indirizzo di destinazione (*DestinationAddress*) espresso in maniera decimale, il pacchetto di dati da inviare (*RequestData*) e la sua ampiezza (*RequestSize*), la specifica delle opzioni (*RequestOption*) con cui inviare il pacchetto, una struttura per accogliere i risultati (*ReplyBuffer*) e la sua relativa ampiezza (*ReplySize*) ed infine il tempo in millesimi di secondo (*Timeout*) prima che la richiesta possa essere considerata scaduta.

Abbiamo naturalmente utilizzato un'istanza della classe *clsFBISocketInfo* trattata nell'articolo precedente, i cui errori saranno inviati al programma utilizzatore della presente classe mediante l'evento ***Error*** ridefinito in questa. Il membro **echoReply** conterrà i dati di risposta del pacchetto di richiesta, mentre **icmpOptions** sarà utilizzato per indicare le opzioni del pacchetto inviato, nel caso in dettaglio, per specificare il *TTL* di andata. Il membro m_Timeout conterrà il valore della proprietà Timeout definita in seguito.

```
33. Private Sub Class_Initialize()
34.     Set fbiSocket = New clsFBISocketInfo
35.     m_Timeout = 1000
36.     icmpOptions.TTL = 255
37. End Sub
```



```
38.
39. Private Sub Class_Terminate()
40.     Set fbiSocket = Nothing
41. End Sub
42.
43. Private Sub fbiSocket_Error(ByVal ErrorCode As Long, ByVal Description As
    String, Cancel As Boolean)
44.     RaiseEvent Error(ErrorCode, Description, Cancel)
45. End Sub
46.
```

Naturalmente durante la fase di inizializzazione è istanziato l'oggetto fbiSocket ed assegnati i valori predefiniti alle proprietà Timeout e TTL. Analogamente alla distruzione dell'oggetto sarà eliminato anche l'oggetto dipendente e quindi liberate le risorse. Come già detto tutti gli errori ricevuti dall'oggetto **fbiSocket** saranno propagati all'utilizzatore della presente classe che potrà provvedere ad annullare la visualizzazione del messaggio di errore.

```
47. Public Function Ping(ByVal Host As String, ByVal Data As String) As Long
48.     Dim hFile As Long
49.     Data = Left(Data, Len(echoReply.Data))
50.     hFile = IcmpCreateFile
51.     Ping = -1
52.     If hFile = 0 Then
53.         Call fbiSocket.Socket.HandleError(21, "Impossibile creare un handle
    valido")
54.     Else
55.         If IcmpSendEcho(hFile, fbiSocket.HostToLong(Host, 0), Data, Len
    (Data), icmpOptions, echoReply, Len(echoReply), m_Timeout) <> 0 Then _
56.             Ping = echoReply.Status
57.         Call IcmpCloseHandle(hFile)
58.     End If
59. End Function
60.
```

Il primo metodo e quello principale è sicuramente **Ping**. Da questo dipendono tutti i valori di ritorno e sostanzialmente consiste di poche e semplici righe; richiede unicamente l'host di destinazione ed i dati da inviare; la riga 49 assicura che i dati richiesti non superino l'ampiezza del buffer dedicato nella struttura ICMP_ECHO_REPLY, mentre la riga 51 inizializza il valore di ritorno della funzione a -1, ad indicare un avvenuto errore; in caso di successo (riga 56) questo valore sarà modificato.

Ad ogni utilizzo sarà creato un handle mediante IcmpCreateFile: in caso di insuccesso verrà generato un errore e la funzione restituirà valore -1, mentre in caso di successo lo stesso handle sarà fornito alla funzione **IcmpSendEcho**, assieme a tutti gli altri dati richiesti. L'indirizzo di destinazione in maniera decimale è ottenuto richiamando il metodo HostToLong della classe *clsFBISocketInfo*. Con la richiesta saranno specificate anche le opzioni (**icmpOptions**), il pacchetto di ritorno (**echoReply**) ed il tempo massimo per eseguire la richiesta (**m_Timeout**).

Se la richiesta ha avuto buon fine, cioè è stato possibile effettuarla senza errori di sistema, il suo valore è differente da zero e pertanto la funzione restituirà il codice di errore contenuto nel campo ***Status*** della pacchetto di risposta (riga 56). In seguito sarà possibile liberare l'handle riservato in precedenza.

Seguono le numerose proprietà relative ordinate per tipologia: le prime indicheranno valori da utilizzare nella richiesta, mentre le ultime corrisponderanno al recupero dei dati restituiti dalla richiesta di echo.

```
61. Public Property Get Timeout() As Long
62.     Timeout = m_Timeout
63. End Property
64.
65. Public Property Let Timeout(ByVal newValue As Long)
66.     m_Timeout = newValue
67. End Property
68.
69. Public Property Get TTL() As Byte
70.     TTL = icmpOptions.TTL
71. End Property
72.
73. Public Property Let TTL(ByVal newValue As Byte)
74.     icmpOptions.TTL = newValue
75. End Property
76.
```

Le due proprietà relative alla richiesta sono **Timeout** e **TTL** che specificano rispettivamente il tempo massimo ed il numero di salti che il pacchetto è in grado di attraversare prima che la richiesta possa considerarsi scaduta.

```
77. Public Property Get EchoAddress() As String
78.     EchoAddress = fbiSocket.LongToIP(echoReply.Address)
79. End Property
80.
81. Public Property Get TripTime() As Long
82.     If echoReply.Status = 0 Then TripTime = echoReply.RoundTripTime
83. End Property
84.
85. Public Property Get Reply() As String
86.     If echoReply.Status = 0 Then Reply = Left$(echoReply.Data,
    echoReply.DataSize)
87. End Property
88.
89. Public Property Get TTLReply() As Byte
90.     TTLReply = echoReply.Options.TTL
91. End Property
92.
93. Public Property Get Status() As Long
94.     Status = echoReply.Status
95. End Property
96.
```

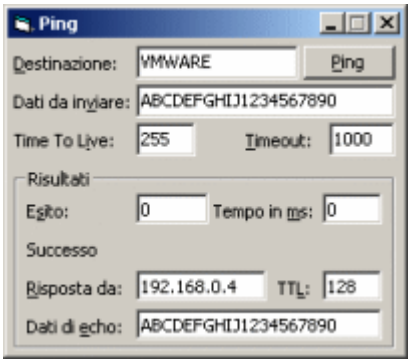
Le proprietà a sola lettura **EchoAddres**, **TripTime**, **Reply**, **TTLReply** e **Status** restituiscono rispettivamente l'indirizzo da cui proviene la risposta, il tempo che il pacchetto ha impiegato per ritornare, i dati contenuti nel pacchetto di ritorno (che per il pacchetto ECHO_RESPONSE dovrebbero essere gli stessi della richiesta ECHO_REQUEST), il TTL del pacchetto restituito ed il codice di ritorno dell'operazione. Un codice di 0 indica un successo mentre tutti gli altri valori indicano errori che trovano la loro descrizione nella proprietà seguente:

```
97. Public Property Get StatusDescription() As String
98.     Select Case echoReply.Status
99.         Case 0: StatusDescription = "Successo"
100.        Case 200: StatusDescription = "Tempo massimo superato"
```



```
101.         Case 11001: StatusDescription = "Buffer troppo piccolo"
102.         Case 11002: StatusDescription = "Rete di destinazione
     irraggiungibile"
103.         Case 11003: StatusDescription = "Host di destinazione
     irraggiungibile"
104.         Case 11004: StatusDescription = "Protocollo di destinazione
     irraggiungibile"
105.         Case 11005: StatusDescription = "Porta di destinazione
     irraggiungibile"
106.         Case 11006: StatusDescription = "Risorse insufficienti"
107.         Case 11007: StatusDescription = "Opzioni errate"
108.         Case 11008: StatusDescription = "Errore hardware"
109.         Case 11009: StatusDescription = "Pacchetto troppo grande"
110.         Case 11010: StatusDescription = "Richiesta scaduta"
111.         Case 11011: StatusDescription = "Richiesta errata"
112.         Case 11012: StatusDescription = "Rotta errata"
113.         Case 11013: StatusDescription = "TTL scaduto nella trasmissione"
114.         Case 11014: StatusDescription = "TTL scaduto nel riassemblaggio"
115.         Case 11015: StatusDescription = "Errori nei parametri"
116.         Case 11016: StatusDescription = "Origine spenta"
117.         Case 11017: StatusDescription = "Opzioni troppo grandi"
118.         Case 11018: StatusDescription = "Destinazione errata"
119.         Case 11019: StatusDescription = "Indirizzo cancellato"
120.         Case 11050: StatusDescription = "Errore generale"
121.         Case Else: StatusDescription = "Errore sconosciuto"
122.     End Select
123. End Property
```

Credo che quest'ultima proprietà non necessiti di commenti, una lunga sequenza di controlli di un unico valore determina il messaggio di errore corrispondente.



Scriveremo un semplicissimo front-end a questa classe che già svolge totalmente il suo lavoro; si comporrà di un solo form ed una serie di caselle di testo per contenere i dati delle richieste e ricevere quelli delle risposte.

Naturalmente tutta la sezione superiore è dedicata ai dati di input ed inizialmente conterrà i valori predefiniti: TTL di 255 salti e Timeout di 1000 millisecondi (1 secondo). Il campo della destinazione potrà contenere un indirizzo IP oppure il nome di un host da interrogare.

Il suo utilizzo è semplice quanto banale e di fianco è riportato un esempio del suo utilizzo su una macchina in rete locale. Possiamo notare di curioso che le macchine interrogate rispondono sempre con un TTL di 128 passaggi.

Se volessimo simulare il funzionamento dell'applicazione Ping di Windows dovremmo semplicemente impostare un TTL di 255 salti ed un pacchetto dati di tutte le lettere alfabetiche minuscole e consecutive dalla a alla w, eventualmente ripetendo la sequenza se i 23 caratteri non dovessero bastare. L'opzione standard di Ping di Windows prevede un pacchetto dati composto da abcdefghijklmnopqrstuvwabcdefghi per un totale di 32 caratteri.

Come già detto, l'articolo utilizza il protocollo ICMP, e non i protocolli TCP o UDP utilizzabili mediante il controllo OCX Winsock. Questo genere di applicazione è possibile soltanto utilizzando le funzioni WSA oppure appoggiandosi ad un controllo esterno che effettui questo lavoro per conto nostro.

Fibia FBI
28 Marzo 2004


Torna all'indice Client/Server

