Home Page
Informazioni
Aiuto

Aprire e chiudere un programma eseguibile (seconda parte) API

http://www.vbsimple.net/howto/ht_032_2.htm

Difficoltà: 5 / 5

<< Continua dalla parte 1

Viste le tre funzioni di supporto del programma passiamo alla funzione di apertura del processo e della chiusura del thread principale per il processo avviato.

```
75. Private Function ApriApp(FileApp As String) As Boolean
76.   StartProc.cb = Len(StartProc)
77.   ApriApp = CreateProcess(FileApp, vbNullString, vbNullString, vbNullString, True,
    DETACHED_PROCESS, vbNullString, vbNullString, StartProc, InfProc)
78. End Function
79.
```

La prima funzione è ApriApp che avvia il processo esterno passato come stringa. Alla riga 77 viene richiamata la funzione API ***CreateProcess*** fornendole i parametri di default: il nome del file da aprire, senza parametri. Sono richieste due strutture particolari per contenere i dati descrittori del processo: tali due strutture sono **StartProc** e **InfProc** definite all'inizio del codice. La prima struttura non verrà più utilizzata all'interno del codice, mentre la seconda verrà richiamata più volte: essa conterrà infatti quattro dati fondamentali per il nostro progetto. Tali dati sono: l'handle del processo creato (*hProcess*), l'handle del thread principale (*hThread*), il numero identificativo del processo (*dwProcessId*) ed il numero identificativo del thread (*dwThreadId*).

La funzion **ApriApp** restituirà il valore True soltanto se l'apertura del processo è andata a buon fine.

```
80. Private Function ChiudiThread() As Boolean
81.   Dim Esito As Boolean
82.   Dim hWindow As Long
83.   Dim ThreadId As Long
84.   Dim CicloNum As Integer
85.   On Error GoTo Error
86.   Esito = True
87.   AppActivate InfProc.dwProcessId
```

La variabile booleana **Esito** serve per verificare la riuscita di alcune operazioni richieste per chiudere il thread principale. All'avvio viene impostata a True (riga 86).

Alla riga 85 viene inserita una procedura di gestione degli errori; il verificarsi di un errore non bloccherà il programma ma eseguirà il codice che si trova alla label ***Error***. Tale codice imposterà il valore di Esito a False e ritornerà l'esecuzione al punto subito successivo a quello dove è stato generato l'errore.

Con questa minima gestione degli errori possiamo lanciare la chiamata alla funzione AppActivate passsandole come *PID* (Process IDentifier) il valore contenuto nella struttura

 Copyright Visual Basic Simple 2002

InfProc. Se la finestra è attivabile essa sarà attivata regolarmente; nel caso contrario sarà generato un errore che sarà catturato dalla funzione di gestione degli errori.

Il verificarsi dell'errore imposta il valore di Esito a False e ripristina la normale esecuzione del programma mediante l'istruzione *Resume Next*.

```
 88.   If Esito Then
 89.     hWindow = GetForegroundWindow()
 90.     ThreadId = GetWindowThreadProcessId(hWindow, 0)
 91.     If IsThread(InfProc.hThread) Then
 92.       If InfProc.dwThreadId = ThreadId Then
```

Il valore di Esito sarà True soltanto se la finestra è stata attivata senza alcun errore.
Alla riga 89 viene ritrovato l'handle della finestra in primo piano (che dovrebbe essere quella attivata) tramite la funzione API ***GetForegroundWindow*** ed alla riga successiva viene ritrovato il numero identificativo del thread che ha generato la finestra tramite la funzione API ***GetWindowThreadProcessID***.

Alla riga 91 viene verificato se il thread principale del processo lanciato è in esecuzione; in tal caso viene controllato se il numero identificativo del thread del processo è lo stesso di quello appena ritrovato, ovvero quello della finestra in primo piano.
Tutti questi controlli sono necessari per verificare che la finestra che stiamo per chiudere sia effettivamente la finestra del processo lanciato.

```
 93.         Call SendMessage(hWindow, WM_CLOSE, 0, 0)
 94.         For CicloNum = 0 To 15
 95.           If IsWindow(hWindow) Then
 96.             Call Sleep(200)
 97.           Else
 98.             ChiudiThread = True
 99.             Call TerminateThread(InfProc.hThread, 0)
100.             Call TerminateProcess(InfProc.hProcess, 0)
101.             Exit For
102.           End If
103.         Next CicloNum
104.       End If
```

Fatti tutti questi controlli possiamo tranquillamente chiudere la finestra in primo piano inviandole il messaggio **WM_CLOSE** tramite la funzione API ***SendMessage***.

In seguito viene avviato un ciclo che verrà eseguito 16 volte. Questo ciclo servirà soltanto per dare il tempo alla finestra di chiudersi e terminare il processo di chiusura.

Alla riga 95 viene verificato se, dopo la chiusura tramite ***SendMessage***, la finestra è ancora attiva (funzione API ***IsWindow***): in tal caso verrà eseguita una pausa di 200 millesimi di secondo, per rallentare l'esecuzione del programma e dare il tempo alla finestra di chiudersi.

Se invece, la finestra è stata chiusa dalla funzione ***SendMessage***, si procede alla chiusura completa del processo terminando il thread principale ed il processo tramite le funzioni ***TerminateThread*** e ***TerminateProcess***. Questo perché alcuni programmi possono mantenere il thread principale attivo anche dopo la chiusura della finestra. Verrà impostato il valore di uscita della funzione e verrà interrotto il ciclo, che condurra la funzione alla fine.

 Copyright Visual Basic Simple 2002

```
105.     Else
106.       ChiudiThread = True
107.     End If
108.   Else
109.     ChiudiThread = Not IsThread(InfProc.hThread)
110.   End If
111.   Exit Function
```

Se invece il controllo fatto alla riga 91 rivela che il thread principale ha terminato la sua esecuzione, la funzione non deve occuparsi di chiudere nulla in quanto il lavoro si è già svolto naturalmente. Verrà impostato il valore di uscita di **ChiudiThread** a True (riga 106).

Il primo controllo effettuato in cima alla funzione riguardava l'attivazione della finestra tramite chiamata alla funzione AppActivate. Se tale attivazione non è andata a buon fine, il valore di Esito sarà impostato su False.

È pertanto chiaro che la finestra non è attivabile.
Cosa fare in tal caso?
La funzione si limiterà a controllare se il thread principale della funzione è terminato spontaneamente, tramite chiamata alla funzione **IsThread**.

Se il thread si è chiuso naturalmente il valore di uscita della funzione sarà True. Se invece il thread principale è ancora attivo, la funzione **ChiudiThread** non potrà applicare i suoi effetti e non procederà alla chiusura del processo. Il valore di ritorno della funzione **ChiudiThread** sarà allora False.

La funzione termina qui.

```
112. Error:
113.   Esito = False
114.   Resume Next
115. End Function
116.
```

Alle righe 112-114 è presente la funzione di gestione di errori che imposta a False la variabile **Esito** e ripristina il funzionamento del programma.

L'ultima funzione da vedere prima della gestione dei controlli del form è la funzione **ChiudiApp** che si occuperà di effettuare la chiusura del processo in due maniere: controllata oppure mediante terminazione immediata del processo.
La funzione **ChiudiApp** richiede un parametro booleano di nome **Termina**. Se esso sarà impostato su False, sarà effettuata la normale chiusura dell'applicazione; se invece il parametro è impostato su True, oltre alla normale chiusura verrà effettuata anche la terminazione del thread e del processo in maniera forzata.

```
117. Private Function ChiudiApp(Termina As Boolean) As Boolean
118.   If IsProcess(InfProc.hProcess) Then
119.     If IsThread(InfProc.hThread) Then
120.       If Not ChiudiThread() And Termina Then
121.         Call TerminateThread(InfProc.hThread, 0)
122.         Call TerminateProcess(InfProc.hProcess, 0)
123.       End If
124.       ChiudiApp = Not IsProcess(InfProc.hProcess)
125.     End If
126.   Else
127.     ChiudiApp = True
```

 Copyright Visual Basic Simple 2002

```
128.    End If
129.    If ChiudiApp Then Call CloseAllHandle
130.  End Function
131.
```

Il codice è alquanto compatto, ma abbastanza semplice. Il primo controllo effettuato è d'obbligo: il processo è ancora in esecuzione? (riga 118) Se non lo fosse la funzione non dovrebbe procedere oltre; il codice eseguirà il codice alla riga 127, poi quello alla riga 129 ed uscirebbe dalla funzione.

Naturalmente nella maggior parte dei casi il processo non si è ancora chiuso ed il nostro programma ha invocato tale funzione perché il processo risulta essere ancora in esecuzione.

Alla riga 119 viene controllato se il thread principale è ancora in esecuzione.
Se non lo fosse la situazione sarebbe complessa: il processo è in esecuzione mentre il thread principale no. Ciò significa che il thread primario, dopo aver lanciato un thread secondario, si è chiuso. La vita del processo sussiste soltanto per un secondo thread che non sappiamo quale sia. In questo sfortunato caso la funzione non può applicarsi e si avvierà alla chiusura con il valore False.

Fortunatamente, nella maggior parte dei programmi il thread primario è quello che regola l'intera esecuzione del programma.
Rivediamo il codice che segue, perché è molto compatto, ma critico:

```
120.        If Not ChiudiThread() And Termina Then
121.          Call TerminateThread(InfProc.hThread, 0)
122.          Call TerminateProcess(InfProc.hProcess, 0)
123.        End If
124.        ChiudiApp = Not IsProcess(InfProc.hProcess)
```

Alla riga 120 viene effettuato un particolare controllo: se la funzione **ChiudiThread** (abbiamo lasciato le parentesi nella chiamata per rendere maggiormente chiaro che si tratta di una funzione) restituisce valore True, che con il *Not* precedente crea un False, ed il parametro Termina è impostato su True, verranno eseguite le righe 121-122.

Se la funzione ChiudiThread restituisce valore True, ciò significherà che il thread principale è stato chiuso con successo. In tal caso non ci importa di forzare la chiusura.
Se invece la funzione ChiudiThread restituisce il valore False, il thread principale non sarà stato chiuso.

Il Not posto prima della funzione fa sì che il ciclo If debba controllare il valore apposto a quello restituito dalla funzione ChiudiThread.
Pertanto, soltanto se la funzione ChiudiThread **non** è riuscita a chiudere il thread principale **e** il parametro Termina è impostato su True, si procederà alla terminazione del thread prima e del processo dopo (righe 121 e 122). Gli handle del thread e del processo sono memorizzate nella struttura **InfProc**, riempita all'apertura del processo.

Fatto questo, il risultato della funzione sarà determinato da un ultimo controllo: se il processo continua ad esistere la nostra funzione non ha avuto effetto e restituirà il valore False. Se il processo invece non esiste più abbiamo ottenuto ciò che volevamo (riga 124).

 Copyright Visual Basic Simple 2002

La funzione si chiude con la riga 129. Se la chiusura ha avuto effetto e pertanto il valore di ***ChiudiApp*** è True, effettueremo la chiusura degli handle aperti, mediante la funzione **CloseAllHandle**.

Segue parte 3 >>

Giuseppe Della Bianca
23 Aprile 2001

Torna all'indice degli HowTo

 Copyright Visual Basic Simple 2002