



Aprire e chiudere un programma eseguibile

(prima parte) 

http://www.vbsimple.net/howto/ht_032.htm

Difficoltà:  5 / 5

In svariate situazioni può essere necessario eseguire un programma esterno che effettui determinate operazioni. Quasi sempre non è necessario occuparsi della sua chiusura, ma talvolta si rende necessario farlo. Come fare allora a chiudere un processo esterno chiamato dalla nostra applicazione?

Prima di vedere il funzionamento di questo codice si raccomanda di leggere attentamente la sezione dedicata alle [informazioni aggiuntive su Processi e Threads](#).

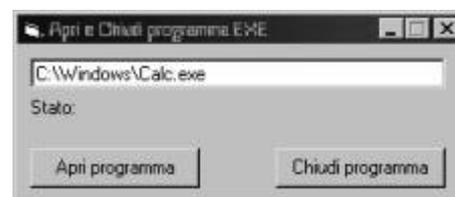
Il progetto si comporrà di un solo form con pochi controlli.

Inseriamo allora una *TextBox*  di nome

NomeProgramma, una *Label*  di nome **LabelStato** e

due *CommandButton*  di nome **PulsanteApri** e

PulsanteChiudi.



Il funzionamento è semplice quanto banale: l'utente immette il percorso del programma da eseguire nella casella di testo, preme il pulsante **Apri programma** e questo si avvia.

Quando arriva il momento di chiudere il programma, l'utente potrà premere il pulsante **Chiudi programma** sul form. Se il programma richiamato si chiuderà normalmente tutto filerà tranquillo, ma se il programma ritarda la chiusura apparirà una finestra di richiesta di terminazione dello stesso. Se l'utente risponderà SI a tale finestra, il programma sarà terminato senza alcuna richiesta di salvataggio o conferma.

Vediamo il codice, peraltro alquanto complesso:

```
1. Option Explicit
2.
3. Private Const WM_CLOSE = &H10
4. Private Const DETACHED_PROCESS = &H8
5.
```

Alla riga 3 vediamo una prima [costante](#)  **API** di nome WM_CLOSE. Esso è un [messaggio](#) che verrà inviato alla finestra per richiederne la chiusura. Alla riga 4 abbiamo un'altra costante API per indicare il tipo di processo da aprire.

```
6. Private Type STARTUPINFO
7.     cb As Long
8.     lpReserved As String
9.     lpDesktop As String
10.    lpTitle As String
11.    dwX As Long
12.    dwY As Long
13.    dwXSize As Long
14.    dwYSize As Long
```

```

15.     dwXCountChars As Long
16.     dwYCountChars As Long
17.     dwFillAttribute As Long
18.     dwFlags As Long
19.     wShowWindow As Integer
20.     cbReserved2 As Integer
21.     lpReserved2 As Byte
22.     hStdInput As Long
23.     hStdOutput As Long
24.     hStdError As Long
25. End Type
26.
27. Private Type PROCESS_INFORMATION
28.     hProcess As Long
29.     hThread As Long
30.     dwProcessId As Long
31.     dwThreadId As Long
32. End Type
33.
34. Private StartProc As STARTUPINFO
35. Private InfProc As PROCESS_INFORMATION
36.

```

Alle righe 6-25 viene dichiarato un nuovo tipo di dati  definito dall'utente di nome **STARTUPINFO**. Alle righe 27 sarà dichiarato un altro tipo di dati di nome **PROCESS_INFORMATION**.

Entrambi i tipi di dati saranno utilizzati dalla funzione API **CreateProcess** che vedremo subito sotto.

```

37. Private Declare Function CreateProcess Lib "kernel32" Alias "CreateProcessA" (ByVal
    lpApplicationName As String, ByVal lpCommandLine As String, ByVal
    lpProcessAttributes As Any, ByVal lpThreadAttributes As Any, ByVal bInheritHandles
    As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal
    lpCurrentDirectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As
    PROCESS_INFORMATION) As Long
38. Private Declare Function TerminateThread Lib "kernel32" (ByVal hThread As Long,
    ByVal dwExitCode As Long) As Long
39. Private Declare Function TerminateProcess Lib "kernel32" (ByVal hProcess As Long,
    ByVal uExitCode As Long) As Long
40. Private Declare Function GetExitCodeThread Lib "kernel32" (ByVal hThread As Long,
    lpExitCode As Long) As Long
41. Private Declare Function GetExitCodeProcess Lib "kernel32" (ByVal hProcess As Long,
    lpExitCode As Long) As Long
42. Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
43. Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd As Long,
    lpdwProcessId As Long) As Long
44. Private Declare Function GetForegroundWindow Lib "user32" () As Long
45. Private Declare Function IsWindow Lib "user32" (ByVal hwnd As Long) As Long
46. Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd
    As Long, ByVal wParam As Long, ByVal lParam As Any) As Long
47. Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
48.

```

Alla riga 37 viene dichiarata la funzione API **CreateProcess** che consente di avviare un nuovo [processo](#) esterno definendo parametri quali la priorità ed il tipo di processo da avviare e riceverne i descrittori del processo in uscita, quali l'[handle](#) del processo, l'handle del thread ed altri dati.

Alla riga 38 dichiariamo la funzione API **TerminateThread** che consente la chiusura immediata di un [thread](#). Analogamente, alla riga 39, dichiariamo la funzione API **TerminateProcess** che termina un [processo](#) di cui si conosce l'handle.

Segue alla riga 40 la dichiarazione della funzione **GetExitCodeThread** che restituisce il

codice di uscita di un thread. Essa verrà utilizzata per comprendere se il thread lanciato è terminato. Alla riga 41 abbiamo la dichiarazione della funzione **GetExitCodeProcess** che restituisce il codice di uscita di un processo e verrà utilizzata anche questa per comprendere se il processo è terminato.

Una funzione fondamentale si trova alla riga 42. La funzione in questione è la **CloseHandle** che libera la memoria dagli [handle allocati](#) e li rende nuovamente disponibili al sistema.

La funzione alla riga 43 è la **GetWindowThreadProcessId** e consente di risalire al thread ed al processo che ha generato una certa finestra. La funzione API successiva è la **GetForegroundWindow** che restituisce l'handle della finestra attualmente in primo piano. Verrà utilizzata per determinare la finestra da chiudere.

Alla riga 45 abbiamo dichiarato la funzione **IsWindow** che verifica che l'handle passato sia effettivamente l'handle di una finestra.

La funzione alla riga 46 è la classica funzione **SendMessage** utilizzata per inviare [messaggi](#) alle finestre. Il messaggio da inviare l'abbiamo dichiarato alla riga 3.

L'ultima funzione dichiarata si trova alla riga 47. La funzione in questione è la **Sleep** e consente di effettuare delle pause all'interno del programma in esecuzione.

Prima di vedere il codice di apertura e chiusura del programma vediamo alcune funzioni utili che verranno utilizzate più avanti:

```
49. Private Function IsThread(hThread As Long) As Boolean
50.     Dim lpExitCode As Long
51.     If GetExitCodeThread(hThread, lpExitCode) Then
52.         If lpExitCode > 0 Then IsThread = True
53.     End If
54. End Function
55.
```

La prima funzione è la **IsThread**, che verifica che il thread richiesto sia in esecuzione. Essa utilizza la funzione **GetExitCodeThread** per richiedere lo stato di chiusura del thread. Se esso è ancora in esecuzione restituirà il valore API **STILL_ACTIVE** (259). Per cui se il codice di uscita del thread è maggiore di 0, il codice sarà in esecuzione o in pausa, ma esiste. Pertanto il valore di uscita della funzione viene settato a **True** (riga 52).

```
56. Private Function IsProcess(hProcess As Long) As Boolean
57.     Dim lpExitCode As Long
58.     If GetExitCodeProcess(hProcess, lpExitCode) Then
59.         If lpExitCode > 0 Then IsProcess = True
60.     End If
61. End Function
62.
```

In maniera del tutto analoga alla funzione precedente, la funzione **IsProcess** verifica che il processo richiesto esista. Viene controllato lo stato di chiusura tramite la funzione **GetExitCodeProcess** e soltanto se il valore restituito è maggiore di 0 (quindi il processo esiste) la funzione restituirà il valore **True** (riga 59).

```
63. Private Function CloseAllHandle() As Boolean
64.     If InfProc.hThread > 0 Then
65.         Call CloseHandle(InfProc.hThread)
66.         InfProc.hThread = 0
```

```
67. End If
68. If InfProc.hProcess > 0 Then
69.     Call CloseHandle(InfProc.hProcess)
70.     InfProc.hProcess = 0
71. End If
72. CloseAllHandle = True
73. End Function
74.
```

La funzione **CloseAllHandle** è estremamente semplice. Essa verifica se il campo `hThread` della struttura `InfProc` è maggiore di 0. Il verificarsi di tale condizione significa che c'è un handle di thread aperto. Sarà pertanto necessario chiuderlo e questo viene fatto tramite la funzione **CloseHandle** (riga 69). Fatto questo viene azzerato tale valore.

La stessa operazione viene effettuata con il valore di `hProcess`.

Tale funzione serve soltanto per liberare completamente le risorse bloccate da altre funzioni ed evitare i blocchi o rallentamenti della macchina.

[Segue parte 2 >>](#)

[Giuseppe Della Bianca](#)
23 Aprile 2001



[Torna all'indice degli HowTo](#)
