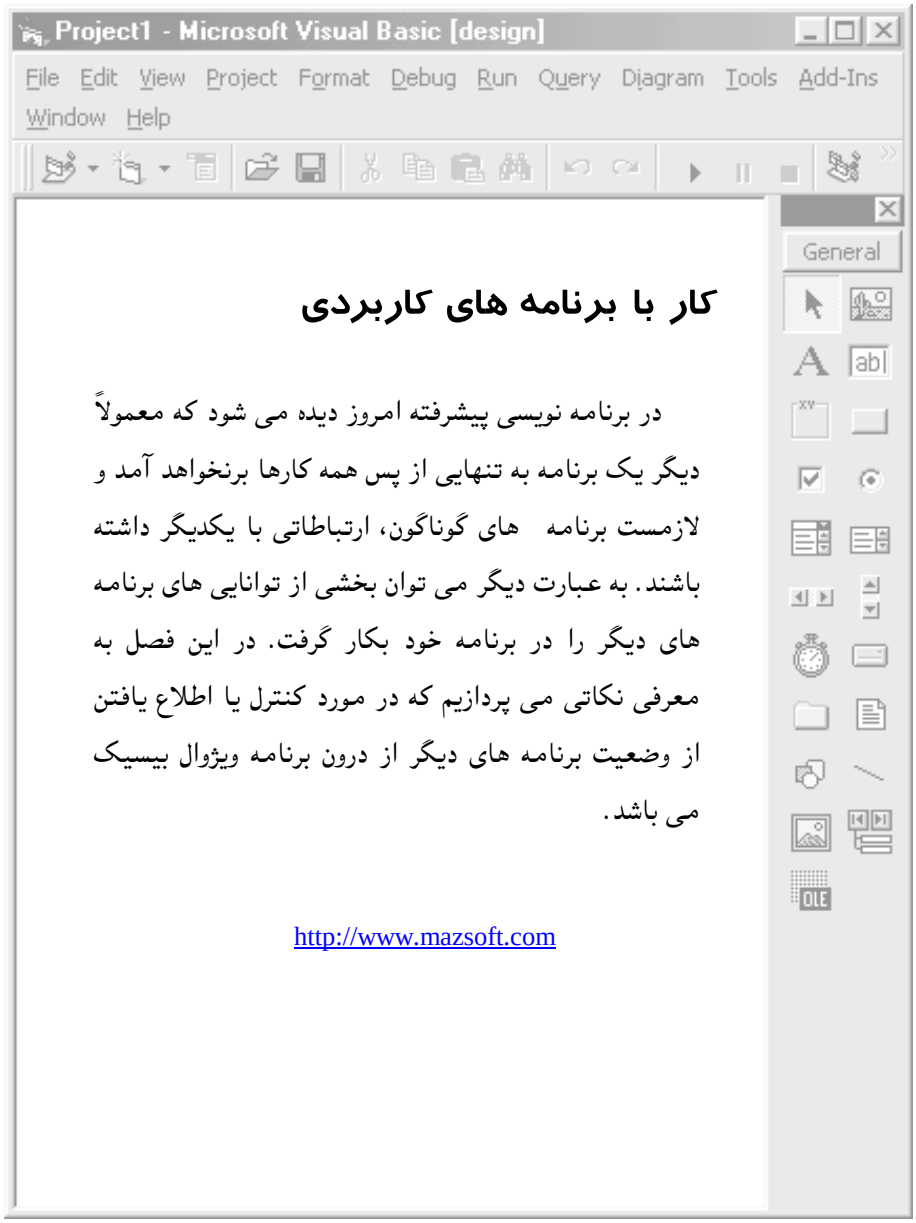

Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins
Window Help
General
کار با برنامه های کاربردی
در برنامه نویسی پیشرفته امروز دیده می شود که معمولاً دیگر یک برنامه به تنهایی از پس همه کارها برنخواهد آمد و لازمست برنامه های گوناگون، ارتباطاتی با یکدیگر داشته باشند. به عبارت دیگر می توان بخشی از توانایی های برنامه های دیگر را در برنامه خود بکار گرفت. در این فصل به معرفی نکاتی می پردازیم که در مورد کنترل یا اطلاع یافتن از وضعیت برنامه های دیگر از درون برنامه ویژوال بیسیک می باشد.
http://www.mazsoft.com

یافتن شناسه یک برنامه تحت MS-DOS و تغییر عنوان آن

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
110701.TXT	۲۱ ژانویه ۱۹۹۵	3	3.1

در این بخش خواهیم دید که چگونه شناسه یک برنامه تحت MS-DOS را یافته و از آن برای تغییر دادن عنوان پنجره یا خارج کردن پنجره آن برنامه از حافظه بوسیله ویژوال بیسیک استفاده نماییم.

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض ایجاد خواهد شد.

۲- دو CommandButton (Command1 و Command2) به Form1 بیافزایید.

۳- کد زیر را در بخش Declarations از فرم یا ماژول برنامه خود وارد کنید:

```
' Enter each of the following Declare statements on one, single line:
Declare Function PostMessage Lib "User"
   (ByVal hWnd As Integer, ByVal wMsg As Integer,
   ByVal wParam As Integer, lParam As Any) As Integer
Declare Sub SetWindowText Lib "User"
   (ByVal hWnd As Integer, ByVal lpString As String)
Declare Function GetActiveWindow Lib "User" () As Integer
Dim MhWnd as Integer
Const WM_CLOSE = &H10
```

۴- کد زیر را در واقعه Click از Command1 وارد کنید:

```
Sub Command1_Click()
   Dim X as Integer
   X = Shell("c:\windows\dosprmpt.pif", 1) ' Open an MS-DOS Window
   For X = 0 To 100
      DoEvents
      ' a bunch of DOEVENTS to wait for the MS-DOS Window to open.
   Next X

   ' Get the handle when the MS-DOS Window has the focus:
   Mhwnd = GetActiveWindow()

   ' Now pass the handle to the window with a new title:
   Call SetWindowText(Mhwnd, "My Application!")
End Sub
```

۵- کد زیر را در واقعه Click از Command2 وارد کنید:

```
Sub Command2_Click()
     Dim X as Integer
     ' Note: In order for this to work on a MS-DOS Window, you have
     ' to have a PIF setup that will allow an MS-DOS Window to be closed.
     ' In the PIF editor, select "Advanced", then click "Close When
     ' Active". This allows MS-DOS applications to be closed
     ' programatically
     X = PostMessage(MhWnd, WM_CLOSE, 0, 0) ' Return greater than zero
                                            ' if successful.
     If X = 0 Then
        MsgBox "Application did not close!"
     End If
End Sub
```

۶- برای اجرای برنامه کلید F5 را فشار دهید.

۷- روی Command1 کلیک کنید تا نتیجه را ببینید.

آیا یک برنامه در حال اجرا در محیط برنامه سازی ویژوال بیسیک است ؟

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
118819.TXT	۲۱ می ۱۹۹۸	3	3.1

در این بخش خواهیم دید که چگونه یک برنامه ویژوال بیسیک می تواند تشخیص دهد که در محیط برنامه سازی ویژوال بیسیک اجرا شده است یا بصورت یک فایل اجرایی. برای انجام این کار دو روش وجود دارد که در زیر به آنها اشاره می شود.

ویژوال بیسیک یک شئ به نام App در اختیار برنامه نویس قرار می دهد که این شئ دارای خصوصیت EXENAME است. هنگامی که برنامه بصورت یک فایل اجرایی در حال اجرا باشد App.EXENAME نام فایل اجرایی را مشخص می کند، اما در محیط ویژوال بیسیک، نام پروژه را برمی گرداند. اگر برای نام پروژه و نام فایل اجرایی از اسامی مختلفی استفاده نمایید، می توانید از این خصوصیت برای تشخیص اینکه برنامه بصورت فایل اجرایی است یا در درون محیط ویژوال بیسیک اجرا شده است استفاده کنید.

علاوه بر روش بالا، می توانید از توابع API نیز برای همین منظور استفاده نمایید. هنگامی که یک برنامه در محیط ویژوال بیسیک اجرا شود نام ماژول برنامه VB است. ولی هنگامی که برنامه بصورت یک فایل اجرایی باشد، نام ماژول برنامه همان نام فایل اجرایی برنامه خواهد بود که در هنگام کامپایل برنامه توسط ویژوال بیسیک به آن نام ایجاد شده است.

برای تعیین نام ماژول یک برنامه می توانید از توابع GetCurrentTask() و TaskFindHandle() استفاده نمایید. مثال زیر نشان می دهد که چگونه از این توابع استفاده نمایید:

۱- یک پروژه جدید ایجاد کنید (Form1 بطور پیش فرض بوجود می آید).

۲- یک ماژول جدید به برنامه اضافه کنید (MODULE1.BAS).

۳- کد زیر را در ماژول وارد کنید:

```
Type TASKENTRY
     dwSize As Long
     hTask As Integer
     hTaskParent As Integer
     hInst As Integer
     hModule As Integer
     wSS As Integer
     wSP As Integer
     wStackTop As Integer
     wStackMinimum As Integer
     wStackBottom As Integer
     wcEvents As Integer
     hQueue As Integer
     szModule As String * 10
     wPSPOffset As Integer
     hNext As Integer
End Type

  ' The following declare must be entered on a single line
  Declare Function TaskFindHandle Lib "Toolhelp" (lpte As TASKENTRY,
     ByVal hTask As Integer) As Integer
  Declare Function GetCurrentTask Lib "Kernel" () As Integer

  Function VBDesignEnvironment () As Integer
     Dim TE As TASKENTRY
     Dim ModuleName As String
     Dim hTask As Integer
     Dim r

     hTask = GetCurrentTask()
     TE.dwSize = Len(TE)
     r = TaskFindHandle(TE, hTask)
     ModuleName = Left(TE.szModule, InStr(TE.szModule, Chr(0)) - 1)

     If ModuleName = "VB" Then
```

```
      VBDesignEnvironment = True
   Else
      VBDesignEnvironment = False
   End If
End Function
```

۴- کد زیر را به واقعه Load از فرم اضافه کنید:

```
Sub Form_Load ()
   Me.Show
   If VBDesignEnvironment() Then
      Print "Design Environment"
   Else
      Print "Executable"
   End If
End Sub
```

۵- پروژه را ذخیره کنید.

٦- برنامه را در محیط ویژوال بیسیک اجرا کنید. فرم باید عبارت Design Environment را نشان دهد.

۷- یک فایل اجرایی از این پروژه ایجاد کنید.

۸- فایل اجرایی را از طریق خود ویندوز اجرا کنید. اکنون فرم باید عبارت Executable را نمایش دهد.

نکات

- تابع TaskFindHandle() در فایل ToolHelp.DLL قرار دارد. این کتابخانه به همراه ویندوز نگارش ۳/۰ ارایه نشده است.
- بهتر است به تابع VBDesignEnvironment() کدی اضافه نمایید که خطاهای بوجود آمده در هنگام اجرای توابع GetCurrentTask() و TaskFindHandle() را کنترل نماید.

تشخیص پایان یافتن یک پروسه اجرا شده توسط Shell (در برنامه ۳۲ بیتی)			
نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
129796.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	3.1

اجرای تابع Shell() در یک برنامه ویژوال بیسیک باعث آغاز اجرای یک برنامه دیگر شده و بلافاصله کنترل اجرا را به برنامه ویژوال بیسیک باز می گرداند. برنامه دوم به انجام عملیات خود ادامه می دهد تا اینکه کاربر آنرا ببندد.

با این حال، اگر در برنامه ویژوال بیسیک خود لازم می دانید تا پایان اجرای برنامه دوم صبر کنید، می توانید با کمک توابع API از وضعیت اجرای برنامه دوم آگاه شوید، اما این تکنیک چندان مناسب نیست. مثال زیر روش بهتری را ارایه می کند.

یک برنامه کاربردی ۱٦ بیتی از تکنیک کاملاً متفاوتی برای انجام این کار استفاده می کند. برای کسب اطلاعات بیشتر در مورد روش قابل استفاده برای برنامه های ۱٦ بیتی به بخش شماره 96844 در بانک اطلاعاتی مایکروسافت تحت عنوان زیر رجوع کنید:

```
ARTICLE-ID: Q96844
TITLE     : HOWTO: Determine When a Shelled Process Has Terminated
```

API ۳۲ بیتی ویندوز دارای قابلیتی است که یک برنامه را قادر می سازد تا پایان یافتن اجرای یک پروسه اجرا شده توسط Shell منتظر بماند. برای استفاده از این قابلیت، به شناسه پروسه اجرا شده توسط Shell نیاز دارید. ساده ترین روش برای بدست آوردن این شناسه، استفاده از تابع CreateProcess() موجود در API به جای تابع Shell() ویژوال بیسیک برای اجرای برنامه ها می باشد.

در یک برنامه ۳۲ بیتی باید یک پروسه قابل آدرس دهی ایجاد کنید. به این منظور باید از تابع CreateProcess() برای اجرای برنامه های دیگر استفاده کنید. تابع CreateProcess() از طریق یکی از پارامترهایش شناسه پروسه اجرا شده توسط Shell را به برنامه شما اعلام می کند.

بـــا داشتن شناسه پروسه مورد نظر، و ارسال آن برای تابع ()WaitForSingleObject می توان باعث شد که اجرای برنامه ویژوال بیسیک تا پایان یافتن آن پروسه متوقف شود.

در برنامه های تحت DOS یک کد خروج[1] نیز توسط خود برنامه تولید می شود تا وضعیت برنامه را مشخص نماید. با اینکه ویندوز روشهای دیگری نیز برای ارایه همین اطلاعات در اختیار برنامه نویسان قرار می دهد اما هنوز بعضی از برنامه ها فقط کد خروج ایجاد می کنند. ارسال شناسه پروسه برای تابع ()GetExitCodeProcess این امکان را فراهم می آورد تا بتوانید این اطلاعات را بدست آورید.

مراحل لازم برای ساختن یک برنامه ویژوال بیسیک که از تابع ()CreateProcess برای اجرای برنامه NotePad استفاده می کند را در زیر ملاحظه می نمایید. این کد نشان می دهد که چگونه از توابع ()CreateProcess و ()WaitForSingleObject استفاده کنید تا پس از اجرای یک برنامه به وسیله Shell، برنامه ویژوال بیسیک شما منتظر پایان یافتن آن بشود. این مثال از تابع ()GetExitCodeProcess نیز به منظور دریافت کد خروج پروسه مورد نظر (در صورت وجود) استفاده می کند.

ترکیب تابع ()CreateProcess بسیار پیچیده است، بنابراین در این مثال این تابع در یک تابع به نام ()ExecCmd بکار رفته است. تابع ()ExecCmd فقط یک پارامتر که همان سطر فرمان مربوط به برنامه مورد نظر برای اجرا باشد را دریافت می کند.

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض بوجود می آید.

۲- کد زیر را به بخش Declarations از Form1 اضافه نمایید:

```
Private Type STARTUPINFO
   cb As Long
   lpReserved As String
   lpDesktop As String
   lpTitle As String
   dwX As Long
   dwY As Long
   dwXSize As Long
   dwYSize As Long
   dwXCountChars As Long
   dwYCountChars As Long
   dwFillAttribute As Long
```

[1] Exit Code

```
     dwFlags As Long
     wShowWindow As Integer
     cbReserved2 As Integer
     lpReserved2 As Long
     hStdInput As Long
     hStdOutput As Long
     hStdError As Long
  End Type

  Private Type PROCESS_INFORMATION
     hProcess As Long
     hThread As Long
     dwProcessID As Long
     dwThreadID As Long
  End Type

Private Declare Function WaitForSingleObject Lib "kernel32" (ByVal _
    hHandle As Long, ByVal dwMilliseconds As Long) As Long

Private Declare Function CreateProcessA Lib "kernel32" (ByVal _
    lpApplicationName As Long, ByVal lpCommandLine As String, ByVal _
    lpProcessAttributes As Long, ByVal lpThreadAttributes As Long, _
    ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, _
    ByVal lpEnvironment As Long, ByVal lpCurrentDirectory As Long, _
    lpStartupInfo As STARTUPINFO, lpProcessInformation As _
    PROCESS_INFORMATION) As Long

Private Declare Function CloseHandle Lib "kernel32" _
    (ByVal hObject As Long) As Long

Private Declare Function GetExitCodeProcess Lib "kernel32" _
    (ByVal hProcess As Long, lpExitCode As Long) As Long

Private Const NORMAL_PRIORITY_CLASS = &H20&
Private Const INFINITE = -1&

Public Function ExecCmd(cmdline$)
     Dim proc As PROCESS_INFORMATION
     Dim start As STARTUPINFO

     ' Initialize the STARTUPINFO structure:
     start.cb = Len(start)

     ' Start the shelled application:
     ret& = CreateProcessA(0&, cmdline$, 0&, 0&, 1&, _
        NORMAL_PRIORITY_CLASS, 0&, 0&, start, proc)

     ' Wait for the shelled application to finish:
     ret& = WaitForSingleObject(proc.hProcess, INFINITE)
     Call GetExitCodeProcess(proc.hProcess, ret&)
     Call CloseHandle(proc.hProcess)
     ExecCmd = ret&
End Function

Sub Form_Click()
     Dim retval As Long
     retval = ExecCmd("notepad.exe")
     MsgBox "Process Finished, Exit Code " & retval
End Sub
```

۳- برای اجرای برنامه کلید F5 را فشار دهید.

۴- به کمک ماوس روی پنجره Form1 کلیک کنید. در این زمان برنامه NotePad اجرا می شود.

۵- برنامه NotePad را خاتمه دهید. یک پنجره پیام ظاهر می شود که نشان دهنده خاتمه یافتن برنامه NotePad است و کد خروج را صفر نشان می دهد. برای استفاده از این مثال با یک برنامه که کد خروج برمی گرداند، از بخش شماره 178357 بانک اطلاعاتی مایکروسافت استفاده کرده و پارامتر ارسالی به تابع ExecCmd را به "project1.exe" تغییر دهید.

☜ توجه : دستور MsgBox که بعد از تابع ()ExecCmd قراردارد اجرا نمی شود، زیرا تابع ()WaitForSingleObject اجازه نمی دهد. تا زمانی که NotePad بسته نشود (یعنی کاربر از منوی File برنامه NotePad گزینه Exit را انتخاب کند) پنجره پیام ظاهر نمی شود.

اجرای یک برنامه ۳۲ بیتی از داخل یک برنامه ویژوال بیسیک

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
129797.TXT	۲۰ آگوست ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

یک برنامه تحت ویندوز می تواند از بیش از یک پروسه تشکیل شود، و یک پروسه می تواند خود دارای بیش از یک دنباله[2] باشد. API ۳۲ بیتی مایکروسافت از قابلیت چندپردازشی[3] پشتیبانی می کند. این قابلیت باعث می شود اجرای چند پروسه و دنباله به

[۲] Thread

[۳] Multitasking

صورت هم زمان بنظر برسد. در این بخش به توضیح پروسه ها و دنباله ها پرداخته و روش ایجاد و استفاده از آنها را توسط ویژوال بیسیک بیان می کنیم.

پروسه برنامه ای است که به حافظه بار شده و برای اجرا آماده است. هر پروسه دارای یک فضای آدرس مجازی اختصاصی[4] است. یک پروسه تشکیل شده است از کد، داده و دیگر منابع سیستم شامل فایلها، ارتباطات و اشیای هم زمان کننده که برای دنباله های پروسه قابل دسترس هستند. هر پروسه با یک دنباله تکی آغاز می شود، ولی دنباله های اضافه ای نیز می توانند بوجود آیند.

یک دنباله می تواند هر بخشی از کد برنامه را اجرا نماید، از جمله بخشی از کد که توسط دنباله دیگری اجرا شده باشد. دنباله ها کوچکترین اجزایی هستند که سیستم عامل برای آنها زمانی از CPU را اختصاص می دهد. هر دنباله در مدتی که منتظر نوبت برای اجراست، یک سری ساختمان های داده را برای ذخیره محتویات خود مورد استفاده قرار می دهد. محدوده ای از حافظه که مورد استفاده دنباله است، محتوی سری رجیسترهای ماشین مربوط به دنباله، پشته کرنل[5]، بلوک محیط دنباله و یک پشته کاربر[6] می باشد. تمامی دنباله های یک پروسه از فضای آدرس مجازی مشترکی استفاده می کنند و می توانند به متغیرهای عمومی و منابع سیستم مربوط به پروسه دسترسی داشته باشند.

یک سیستم عامل چندکاره، زمان CPU را بین دنباله هایی که به آن نیاز دارند تقسیم می کند. در ویندوز، Win32 API برای چندکارگی از نوع نوبتی طراحی شده است. به این معنا که سیستم قطعات کوچکی از زمان سی پی یو را برای هر یک از دنباله ها اختصاص می دهد. دنباله ای که اکنون در حال اجراست، پس از اتمام قطعه زمان مربوط به خود متوقف می شود و به یک دنباله دیگر اجازه اجرا داده می شود. هنگامی که سیستم از یک دنباله به دنباله دیگری سویچ می کند، وضعیت دنباله متوقف شده را ذخیره کرده و وضعیت ذخیره شده دنباله بعدی را بازیابی می کند.

---

[4] Private Virtual Address Space

[5] Kernel Stack

[6] User Stack

از آنجا که واحدهای زمانی بسیار کوچک هستند (تقریبا ۲۰ هزارم ثانیه)، این طور به نظر خواهد رسید که چند دنباله بطور هم زمان اجرا می شوند. در سیستم های چند پردازنده، عملا دنباله های مختلف بین پردازنده های سیستم تقسیم می شوند. در سیستم های تک پردازنده استفاده از چند دنباله بطور هم زمان به معنی اجرای تعداد بیشتری دستورات در واحد زمان نیست. در حقیقت، اگر تعداد زیادی دنباله در حال اجرا داشته باشیم، بازده سیستم حتی کاهش نیز می یابد.

چگونه برنامه های Win32 را از درون ویژوال بیسیک اجرا کنیم

برای اجرای برنامه های Win32 از درون یک برنامه ویژوال بیسیک دو راه وجود دارد:

- استفاده از فرمان Shell ویژوال بیسیک. این روش یک پروسه جدید ایجاد کرده و مشخصه[۷] آن پروسه را برمی گرداند. اما اگر بخواهید کارهای مفیدی با پروسه انجام دهید، باید شناسه[۸] پروسه را در دست داشته باشید. این شناسه از طریق فراخوانی تابع OpenProcess قابل حصول است.
- استفاده از تابع CreateProcess که هم پروسه و هم دنباله اصلی را ایجاد می کند. هم پروسه و هم دنباله اولیه دارای یک مشخصه ۳۲ بیتی خواهند بود که تا زمان خاتمه یافتن پروسه یا دنباله قابل استفاده خواهد بود. می توان از این مشخصه ۳۲ بیتی برای یافتن پروسه یا دنباله مربوطه در سیستم استفاده نمود. به علاوه شناسه ای نیز برای پروسه و دنباله بوجود می آید. تمامی این چهار مقدار در ساختمان داده PROCESS_INFORMATION ذخیره می شوند. این ساختمان داده بصورت ارجاعی برای تابع CreateProcess ارسال می شود.

شناسه پروسه ای که توسط هر یک از دو روش فوق بدست می آید، قابل استفاده برای دیگر توابع Win32 API مانند TerminateProcess نیز می باشد.

درک این نکته بسیار اهمیت دارد که سیستم در زمان ایجاد هر شئ (مثلا پروسه یا دنباله) یک شمارشگر استفاده با مقدار اولیه ۱ به آن شئ نسبت می دهد. سپس درست قبل از اینکه

---

[۷] ID

[۸] Handle

تابع CreateProcess خاتمه یابد و بازگردد، این تابع هر دو شئ پروسه و دنباله را باز کرده و شناسه های مربوط به هریک را در اعضای hProcess و hThread از ساختمان داده PROCESS_INFORMATION قرار می دهد.

هنگامی که CreateProcess این اشیاء را باز می کند، شمارشگر استفاده هر یک را افزایش داده و به ۲ می رساند. به همین دلیل پیش از اینکه سیستم ویندوز بتواند شئ پروسه را از بین ببرد، پروسه باید خاتمه یابد (و شمارشگر استفاده به ۱ کاهش یابد) و پروسه تولید کننده این پروسه باید CloseHandle را فرابخواند (که شمارشگر استفاده را به صفر کاهش می دهد). برای از بین بردن شئ دنباله، باید دنباله خاتمه یابد و پروسه تولید کننده آن دنباله نیز باید شناسه مربوط به شئ دنباله را معدوم کند.

اخطار: معدوم کردن و بستن این شناسه ها بسیار اهمیت دارد. در صورتی که این کار درست انجام نشود بخشهایی از حافظه سیستم به هدر می رود، زیرا بعضی از اشیای سیستم ویندوز همچنان حافظه را در اشغال خود نگه می دارند.

هنگامی که توسط OpenProcess یک شناسه پروسه ایجاد می کنید نیز چنین ملاحظاتی لازم است. در این مورد نیز، شمارشگر استفاده یک واحد افزایش یافته و تا زمانی که شناسه معدوم نشود، شئ پروسه در حافظه باقی می ماند، حتی اگر اجرای پروسه خاتمه یافته باشد.

مثال

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض بوجود می آید.

۲- کد زیر را در Form1 وارد کنید:

```
Option Explicit

Private Type PROCESS_INFORMATION
   hProcess As Long
   hThread As Long
   dwProcessId As Long
   dwThreadId As Long
End Type

Private Type STARTUPINFO
   cb As Long
   lpReserved As String
   lpDesktop As String
   lpTitle As String
```

```
   dwX As Long
   dwY As Long
   dwXSize As Long
   dwYSize As Long
   dwXCountChars As Long
   dwYCountChars As Long
   dwFillAttribute As Long
   dwFlags As Long
   wShowWindow As Integer
   cbReserved2 As Integer
   lpReserved2 As Long
   hStdInput As Long
   hStdOutput As Long
   hStdError As Long
End Type

Private Declare Function CreateProcess Lib "kernel32" _
   Alias "CreateProcessA" _
   (ByVal lpApplicationName As String, _
   ByVal lpCommandLine As String, _
   lpProcessAttributes As Any, _
   lpThreadAttributes As Any, _
   ByVal bInheritHandles As Long, _
   ByVal dwCreationFlags As Long, _
   lpEnvironment As Any, _
   ByVal lpCurrentDriectory As String, _
   lpStartupInfo As STARTUPINFO, _
   lpProcessInformation As PROCESS_INFORMATION) As Long

Private Declare Function OpenProcess Lib "kernel32.dll" _
   (ByVal dwAccess As Long, _
   ByVal fInherit As Integer, _
   ByVal hObject As Long) As Long

Private Declare Function TerminateProcess Lib "kernel32" _
   (ByVal hProcess As Long, _
   ByVal uExitCode As Long) As Long

Private Declare Function CloseHandle Lib "kernel32" _
   (ByVal hObject As Long) As Long

Const SYNCHRONIZE = 1048576
Const NORMAL_PRIORITY_CLASS = &H20&

Private Sub Form_Click()
   Dim pInfo As PROCESS_INFORMATION
   Dim sInfo As STARTUPINFO
   Dim sNull As String
   Dim lSuccess As Long
   Dim lRetValue As Long

   sInfo.cb = Len(sInfo)
   lSuccess = CreateProcess(sNull, _
                            "Calc.exe", _
                            ByVal 0&, _
                            ByVal 0&, _
                            1&, _
                            NORMAL_PRIORITY_CLASS, _
                            ByVal 0&, _
```

```
                                sNull, _
                                sInfo, _
                                pInfo)

    MsgBox "Calculator has been launched!"

    lRetValue = TerminateProcess(pInfo.hProcess, 0&)
    lRetValue = CloseHandle(pInfo.hThread)
    lRetValue = CloseHandle(pInfo.hProcess)

    MsgBox "Calculator has terminated!"
End Sub
```

۳- از منوی Run گزینه Start را انتخاب کنید یا کلید F5 را فشار دهید تا برنامه اجرا شود. روی Form1 کلیک کنید تا برنامه ماشین حساب اجرا شود. یک پنجره پیام ظاهر شده و مشخص می کند که برنامه مورد نظر با موفقیت اجرا شده است. روی دکمه OK کلیک کنید تا پنجره پیام و برنامه ماشین حساب بسته شود. پنجره پیام دیگری ظاهر می شود که نشان دهنده خاتمه برنامه ماشین حساب است.

خاتمه دادن به یک برنامه در حال اجرا به کمک منوی سیستم

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
142817.TXT	۲۱ می ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

ویژوال بیسیک می تواند به کمک تابع SendMessage هر پنجره فعالی را ببندد. به کمک تابع SendMessage می توانید یک پیام را برای هر پنجره ای در محیط ویندوز که شناسه آنرا بدانید ارسال کنید. برای تعیین شناسه یک پنجره می توانید از تابع FindWindow استفاده کنید.

برای مثال برنامه ای می سازیم که ماشین حساب ویندوز را اجرا کرده و سپس آنرا می بندد. (برای اینکه این مثال در محیط ویندوز **۹۵** اجرا شود تغییراتی در اصل مثال داده شد. م .):

۱- یک فرم به نام Form1 ایجاد کنید.

۲- دو Command Button به نامهای Command1 و Command2 روی فرم قرار دهید.

۳- کد زیر را در واقعه کلیک از Command1 قرار دهید:

```
Private Sub Command1_Click()
   X = Shell("Calc.exe")
End Sub
```

۴- کد زیر را در واقعه کلیک از Command2 قرار دهید:

```
Private Sub Command2_Click()
   Const NILL = 0&
   Const WM_SYSCOMMAND = &H112
   Const SC_CLOSE = &HF060

   lpClassName$ = "SciCalc"
   lpCaption$ = "Calculator"

   '* Determine the handle to the Calculator window.
   Handle = FindWindow(lpClassName$, lpCaption$)

   '* Post a message to Calc to end its existence.
   X& = SendMessage(Handle, WM_SYSCOMMAND, SC_CLOSE, NILL)

End Sub
```

۵- در بخش Declarations کد زیر را وارد کنید:

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
      (ByVal lpClassName As Any, ByVal lpCaption As Any) As Long
Private Declare Function SendMessage& Lib "user32" Alias "SendMessageA" _
      (ByVal hwnd%, ByVal wMsg%, ByVal wParam%, ByVal lParam As Long)
```

۶- برنامه را اجرا کنید. روی Command1 کلیک کنید تا برنامه ماشین حساب اجرا شود. روی Command2 کلیک کنید تا پنجره ماشین حساب بسته شود.

شبیه سازی فشار کلیدهای صفحه کلید برای برنامه های تحت DOS

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
142819.TXT	۲۴ آگوست ۱۹۹۸	4 , 5 , 6	3.1 / 95 / 98

در این بخش تکنیکی را معرفی می کنیم که به کمک آن می توانید فشار کلیدهای صفحه کلید را برای یک برنامه تحت DOS شبیه سازی کنید. این روش در نگارش های ۳/۱ ، ۹۵ و ۹۸ ویندوز قابل اجرا است ولی در نگارش NT عمل نمی کند.

تابع SendKeys می تواند فشار کلیدها را برای پنجره فعال شبیه سازی نماید، درست مانند اینکه این کلیدها روی صفحه کلید تایپ شده باشند. اگرچه این امکان وجود ندارد که توسط این تابع فشار کلیدها را مستقیماً برای یک برنامه تحت DOS شبیه سازی کنیم، اما می توانیم متنی را در Clipboard قرار داده و با استفاده از تابع SendKeys عمل چسباندن متن را در پنجره مربوط به برنامه تحت DOS شبیه سازی نماییم تا محتویات Clipboard به درون پنجره حاوی برنامه تحت DOS منتقل شود.

به منظور اجرای یک برنامه تحت DOS در یک پنجره (نه بصورت تمام صفحه)، باید ویندوز در حالت ۳۸۶ گسترش یافته[9] اجرا شده باشد (ویندوز ۳/۱). برای اینکه مطمئن شوید برنامه مورد نظر همیشه در یک پنجره اجرا می شود، باید از برنامه ویرایشگر PIF استفاده کنید و گزینه مربوط به اجرا در پنجره را انتخاب کنید.

مثال زیر نشان می دهد که چگونه کلید هایی را برای یک برنامه تحت DOS که در یک پنجره اجرا می شود، ارسال نمایید:

۱- یک پنجره MS-DOS باز کنید.

۲- یک پروژه جدید در ویژوال بیسیک ایجاد کنید.

۳- کد زیر را در بخش Declarations از Form1 وارد کنید:

```
Dim progname As String
```

[9] 386 Enhanced Mode

۴- دو Label روی فرم قرار دهید. خصوصیت Caption اولین Label را به MS-DOS App Title تغییر دهید. خصوصیت Caption دومین Label را به Keys to send تغییر دهید.

۵- دو Text Box روی فرم در مقابل Label های فوق قرار دهید. محتویات پیش فرض آنها را پاک کنید. این ابزارها برای این بکار می روند تا به کاربر امکان دهند عنوان پنجره برنامه تحت DOS و کلیدهای مورد نظر برای ارسال به آنرا در آنها مشخص کند. خصوصیت Name این Text box ها را به ترتیب به DosTitle و DosKeys تغییر دهید.

٦- یک Command Button روی فرم قرار دهید و خصوصیت Caption آنرا به Send Keys تغییر دهید.

۷- کد زیر را به واقعه Click از Command1 اضافه کنید:

```
Private Sub Command1_Click()
   'Ensure that progname is set to the titlebar of Visual Basic while
   ' running.
   progname = "Project1 - Microsoft Visual Basic [run]"
   clipboard.Clear
   clipboard.SetText DosKeys.Text + Chr$(13)  ' Append a <CR>.
   AppActivate DosTitle.Text
   SendKeys "% ep", 1
   AppActivate progname
End Sub
```

اگر متنی که برای پنجره DOS ارسال می کنید فرمان DIR یا فرمان دیگری باشد که برای اجرا مدتی زمان لازم دارد، فراخوانی تابع AppActivate که بلافاصله بعد از فراخوانی SendKeys صورت می گیرد، می تواند وقفه ای در پردازش فرمان ایجاد کند. فراخوانی AppActivate باید در یک Timer قرار بگیرد که مدت زمان آن به اندازه کافی تنظیم شده باشد، و این Timer باید در روتین مربوط به Command_Click فعال شده باشد. پیش از خروج از روتین Timer باید آنرا غیر فعال نمود.

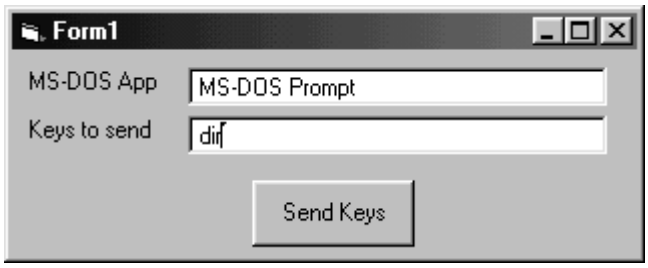
۸- برنامه را اجرا کنید.

۹- عنوان پنجره برنامه تحت DOS مورد نظر را در Text Box مربوطه وارد کنید. عنوان پیش فرض یک پنجره DOS عبارتست از MS-DOS Prompt .

۱۰- کلید های مورد نظر خود برای ارسال به برنامه را در text box به نام DosKeys وارد کنید.

۱۱- روی دکمه Send Keys کلیک کنید. کلیدهای مورد نظر به Clipboard ارسال شده و سپس در پنجره MS-DOS ظاهر می شوند.

برای استفاده از این تکنیک در یک برنامه کامپایل شده ویژوال بیسیک باید progname از Microsoft Visual Basic به نام فایل اجرایی برنامه شما تغییر کند. به علاوه، اگر بخواهید متنی که در Clipboard قرار می گیرد را مشاهده کنید می توانید برنامه Clipboard Viewer را اجرا نمایید.



شکل 1 - نمای ظاهر برنامه شبیه سازی فشار کلیدها

جلوگیری از اجرای هم زمان چند مورد از یک برنامه ویژوال بیسیک

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
142937.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	3.1 / 95 / 98

در این بخش خواهیم دید که چگونه زمانی که برنامه یکبار اجرا شده باشد می توان از اجرای مجدد آن جلوگیری کرد. به علاوه، زمانی که سعی کنید مجددا برنامه ای را اجرا کنید، کنترل به اولین مورد اجرا شده از برنامه منتقل می شود.

معمولا برنامه نویسان مایل هستند که در هر لحظه فقط یک نمونه از برنامه ای که می نویسند در حال اجرا باشد. مثلا اگر سعی کنید برنامه File Manager ویندوز را اجرا کنید و در همان لحظه پنجره دیگری از همان برنامه نیز در حال اجرا باشد، اولین پنجره فعال شده و روی صفحه قرار می گیرد. مثال زیر روش انجام این کار را در یک برنامه ویژوال بیسیک نشان می دهد:

۱- در فرم آغازین برنامه خود (مثلاً Form1) کد زیر را در واقعه Form_Load وارد کنید:

```
Private Sub Form_Load ()
  If App.PrevInstance Then
    SaveTitle$ = App.Title
    App.Title = "... duplicate instance."
    Form1.Caption = "... duplicate instance."
    AppActivate SaveTitle$
    SendKeys "% R", True
    End
  End If
End Sub
```

۲- از منوی فایل، گزینه Make EXE File را انتخاب کنید.

۳- از محیط ویژوال بیسیک خارج شوید.

۴- فایل اجرایی برنامه خود را اجرا کنید.

۵- برنامه ای را که در مرحله ۴ اجرا نموده اید، مینیمایز کنید.

۶- سعی کنید یک بار دیگر برنامه را مطابق مرحله ۴ اجرا کنید.

☜ توجه : اگر پیام خطای illegal function call را در برنامه خود دریافت کردید، مطمئن شوید که خصوصیت Caption فرم مورد نظر دقیقاً با عنوان برنامه یکی باشد.

هنگامی که سعی می کنید برنامه را دوباره در پنجره دیگری اجرا کنید، برنامه به این ترتیب عمل می کند:

۱- خصوصیت PrevInstance از شئ App را کنترل می کند تا ببیند آیا مورد دیگری از این برنامه با همان App.Title قبلاً اجرا شده است یا خیر.

۲- اگر پاسخ مثبت باشد، مورد دوم از برنامه خصوصیت App.Title خود را در یک متغیر موقتی ذخیره می کند تا از آن برای فعال کردن مورد اول استفاده کند.

۳- سپس نام خودش را عوض می کند تا از ایجاد اشکال در فراخوانی AppActivate جلوگیری کند.

۴- آنگاه، AppActivate را فرامی خواند، که باعث می شود پنجره اولین مورد از برنامه، پنجره جاری باشد.

۵- اکنون که اولین مورد از برنامه فعال شده است، مورد دوم به کمک SendKeys باعث می شود پنجره مورد اول از حالت مینیمایز شده به حالت معمولی برگردد.

٦- در نهایت، مورد دوم از برنامه به اجرای خود خاتمه می دهد.

بستن پنجره یک برنامه بعد از پایان اجرای آن

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
145701.TXT	۲۴ آگوست ۱۹۹۸	4 , 5 , 6	95 / 98

هنگامی که از فرمان Shell در ویژوال بیسیک برای اجرای یک بچ فایل یا یک برنامه تحت DOS در ویندوز ۹۵ یا ۹۸ استفاده می کنید، پنجره DOS پس از پایان اجرای برنامه بطور پیش فرض باز می ماند. در ویندوز ۳/۱ و ویندوز برای گروههای کاری ۳/۱۱ و ویندوز NT نگارش ۳/۵۱ ، پنجره DOS پس از پایان یافتن برنامه بسته می شود. در این بخش خواهیم دید که

چگونه می توان ویندوز ۹۵ یا ۹۸ را مجبور کرد پس از پایان اجرای برنامه پنجره DOS را ببندد.

برای اینکار می توانید از تابع Shell ویژوال بیسیک به همراه گزینه C/ مربوط به Command.Com استفاده کنید. این گزینه باعث می شود پنجره DOS در محیط ویندوز ۹۵ یا ۹۸ پس از پایان برنامه بسته شود. نحوه استفاده از گزینه C/ در ویژوال بیسیک بصورت زیر است که در آن <pathname> مسیر قرار گیری و نام برنامه مورد نظر برای اجرا به همراه گزینه های سطر فرمان آن برنامه می باشد.

```
h = Shell("COMMAND.COM /C <pathname>")
```

اگر از این گزینه C/ در ویندوز ۳/۱ و ویندوز برای گروههای کاری ۳/۱۱ و ویندوز NT نگارش ۳/۵۱ استفاده کنید نیز نتیجه مورد نظر حاصل می شود. بنابراین می توانید از این تکنیک در تمامی نگارشهای ویندوز استفاده نمایید.

تشخیص اینکه یک برنامه در حال اجرا بصورت DLL است یا EXE

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
150211.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

در این بخش خواهیم دید که چگونه می توان تشخیص داد که یک برنامه ویژوال بیسیک در حال اجرا از نوع EXE است یا از نوع DLL. کاربرد چنین چیزی زمانی است که شما برنامه ای را به هر دو صورت EXE و DLL ایجاد می کنید و در هر مورد لازمست عملیات مختلفی در برنامه صورت دهید.

برای تعیین اینکه یک برنامه از نوع EXE است یا DLL روشهای متفاوتی وجود دارد.

روش اول - اطلاعات را در یکی از برچسب های resource ها قرار دهید.

هنگام ساخت یک EXE یا DLL دکمه Options را کلیک کنید. در بخش Version Information در مقابل File Description نوع فایل کامپایل شده را وارد کنید. برای آزمایش این روش در یک برنامه، کد زیر را در هر بخشی از برنامه که مورد نظر شماست وارد کنید.

```
If App.FileDescription = "EXE" Then
   MsgBox "I am an EXE"
ElseIf App.FileDescription = "DLL" Then
   MsgBox "I am a DLL"
End If
```

روش دوم – کامپایل کردن برنامه بر حسب وضعیت

اگر فیلدهای version information مورد استفاده قرار می گیرند، از روش کامپایل کردن بر حسب وضعیت استفاده کنید، مانند مثال زیر:

```
#If EXE Then
   Msgbox "I am an EXE"
#Else
   MsgBox "I am a DLL"
#EndIf
```

پیش از کامپایل کردن فایل EXE ، از منوی Tools گزینه Options را انتخاب کرده و در تابلوی Advanced، در فیلد Conditional Compilation Arguments سطر زیر را وارد کنید:

```
EXE=1
```

اگر قصد ایجاد DLL دارید، سطر زیر را در Conditional Compilation Arguments وارد کنید:

```
EXE=0
```

در روش دوم، کدی که مورد نیاز DLL یا EXE نباشد کامپایل نخواهد شد. اگر قرار است نگارش DLL یا EXE از یک برنامه بطور متفاوتی عمل کنند، این نتیجه قابل توجه است.

روش سوم – استفاده از توابع API برای تعیین پسوند فایل

اگر شناسه برنامه برای تابع GetModuleFileName ارسال شود، مسیر کامل فایل برنامه مشخص می شود. ویژوال بیسیک شناسه برنامه را در خصوصیت hInstance از شئ App ذخیره می کند.

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض ایجاد می شود.

۲- در بخش Declarations از Form1 کد زیر را وارد کنید:

```
Private Declare Function GetModuleFileName Lib "kernel32" Alias _
"GetModuleFileNameA" (ByVal hModule As Long, _
ByVal lpFileName As String, ByVal nSize As Long) As Long

Option Compare Text
```

۳- در واقعه کلیک از Form1 کد زیر را وارد کنید:

```
Private Sub Form_Click()
   Dim sFilePath As String * 255
   Dim sVarFilePath As String
   Dim sFileExt As String

   GetModuleFileName App.hInstance, sFilePath, Len(sFilePath)
   'Trim out the trailing characters

   sVarFilePath = Trim(sFilePath)

   'Capture the file extension
   sFileExt = Mid(sVarFilePath, Len(sVarFilePath) - 3, 3)

   'Make the comparison
   If sFileExt = "EXE" Then
      MsgBox "I am an EXE"
   Else
      MsgBox "I am a DLL"
   End If

End Sub
```

۴- برنامه را بصورت EXE کامپایل کنید. روی Form1 کلیک کنید تا برنامه نوع فایل اجرایی برنامه را تعیین کند.

پایان دادن به دیگر برنامه ها از درون ویژوال بیسیک

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
153463.TXT	۱۰ نوامبر ۱۹۹۸	4 , 5 , 6	All

ممکنست در بعضی موارد لازم شود از درون کد برنامه ویژوال بیسیک برنامه های دیگری را خاتمه دهید. برای مثال، برنامه شما ممکنست با بعضی برنامه های مشخص تداخل داشته باشد، بنابراین باید بتوانید برنامه مزاحم را ببندید تا برنامه شما بدرستی کار خود را انجام دهد. مثال ارایه شده در این بخش روش انجام این کار را نشان می دهد:

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض بوجود می آید.

۲- یک Command button (به نام Command1) به برنامه خود اضافه کنید.

۳- کد زیر را به بخش Declarations از Form1 اضافه کنید:

```
Option Explicit

Private Declare Function FindWindow Lib "User" (ByVal lpClassName _
   As Any, ByVal lpWindowName As Any) As Integer
Private Declare Function PostMessage Lib "User" (ByVal hWnd _
   As Integer, ByVal wMsg As Integer, ByVal wParam As Integer, _
   lParam As Any) As Integer

Private Const WM_QUIT = &H12

Private Sub Command1_Click()
  Dim sTitle As String
  Dim iHwnd As Integer
  Dim ihTask As Integer
  Dim iReturn As Integer
  sTitle = "Notepad - (Untitled)"
  iHwnd = FindWindow(0&, sTitle)
  iReturn = PostMessage(iHwnd, WM_QUIT, 0, 0&)
  MsgBox "Notepad has been Closed Down"
End Sub
```

۴- برنامه NotePad را اجرا کنید.

۵- در ویژوال بیسیک F5 را فشار دهید تا برنامه اجرا شود. روی Command Button کلیک کنید تا برنامه NotePad بسته شود. یک پنجره پیام این موضوع را اطلاع می دهد.

☜ توجه: برای نگارش ۳۲ بیتی ویژوال بیسیک ۴ و ویژوال بیسیک ۵ و٦ در مرحله ۳ باید از کد زیر استفاده کنید:

```
Option Explicit

Private Declare Function FindWindow Lib "user32" Alias "FindWindowA"_
      (ByVal lpClassName As Any, ByVal lpWindowName As Any) As Long
Private Declare Function PostMessage Lib "user32" Alias "PostMessageA"_
      (ByVal hwnd As Long, ByVal wMsg As Long, _
       ByVal wParam As Long, ByVal lParam As Long) As Long
Private Const WM_QUIT = &H12

Private Sub Command1_Click()
    Dim sTitle As String
     Dim iHwnd As Long
     Dim ihTask As Long
     Dim iReturn As Long
     sTitle = "Untitled - Notepad"
     iHwnd = FindWindow(0&, sTitle)
     iReturn = PostMessage(iHwnd, WM_QUIT, 0&, 0&)
     MsgBox "Notepad has been Closed Down"
End Sub
```

کاربرد ShellExecute در نمایش فایلهای Associate شده

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
170918.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	32 bit

تابع ShellExecute قادر است برنامه های کاربردی که به پسوندهای خاصی از فایلها نسبت داده شده اند را بدون دانستن نام برنامه کاربردی اجرا نماید. برای مثال می توانید با ارسال نام فایل ARCADE.BMP برای تابع ShellExecute برنامه PaintBrush را اجرا نمایید.

تابع ShellExecute فایل مشخص شده را در برنامه مربوطه باز کرده یا چاپ می کند. نحوه تعریف این تابع در ویژوال بیسیک به صورت زیر است:

```
Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
            (ByVal hwnd As Long, ByVal lpszOp As String, _
             ByVal lpszFile As String, ByVal lpszParams As String, _
             ByVal LpszDir As String, ByVal FsShowCmd As Long) As Long
```

در زیر توضیحاتی در مورد هر یک از پارامترهای فوق ارایه می شود:

hwnd مشخص کننده پنجره فراخواننده تابع است. این پنجره تمامی پیام هایی را که از طرف برنامه فراخوانده شده ارسال شود، دریافت می کند.

lpszOp اشاره گر به یک رشته حرفی که با کاراکتر null خاتمه یافته است. این رشته مشخص کننده عمل مورد درخواست است مانند open یا print. اگر این پارامتر null باشد، بطور پیش فرض open مورد استفاده قرار می گیرد.

lpszFile اشاره گر به یک رشته حرفی که با کاراکتر null خاتمه یافته است. این رشته مشخص کننده نام فایل مورد نظر برای باز شدن است.

lpszParams اشاره گر به یک رشته حرفی که با کاراکتر null خاتمه یافته است. زمانی که lpszFile حاوی نام یک فایل اجرایی باشد، این رشته مشخص کننده پارامترهای ارسالی برای آن برنامه است. زمانی که lpszFile حاوی نام یک فایل غیر اجرایی باشد این پارامتر null است.

lpszDir اشاره گر به یک رشته حرفی که با کاراکتر null خاتمه یافته است. این رشته حاوی دایرکتوری پیش فرض می باشد.

fsShowCmd مشخص می کند که پنجره برنامه کاربردی زمانی که اجرا شد نمایش داده بشود یا خیر.

مثال زیر نشان می دهد که چگونه یک برنامه را اجرا نمایید یا یک فایل را در برنامه مربوط به همان فایل بار کنید.

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض ایجاد می شود.

۲- کد زیر را به بخش Declarations از Form1 اضافه کنید:

```
Option Explicit

Private Declare Function ShellExecute Lib "shell32.dll" Alias _
"ShellExecuteA" (ByVal hwnd As Long, ByVal lpszOp As _
String, ByVal lpszFile As String, ByVal lpszParams As String, _
```

```
ByVal lpszDir As String, ByVal FsShowCmd As Long) As Long

Private Declare Function GetDesktopWindow Lib "user32" () As Long

Const SW_SHOWNORMAL = 1

Const SE_ERR_FNF = 2&
Const SE_ERR_PNF = 3&
Const SE_ERR_ACCESSDENIED = 5&
Const SE_ERR_OOM = 8&
Const SE_ERR_DLLNOTFOUND = 32&
Const SE_ERR_SHARE = 26&
Const SE_ERR_ASSOCINCOMPLETE = 27&
Const SE_ERR_DDETIMEOUT = 28&
Const SE_ERR_DDEFAIL = 29&
Const SE_ERR_DDEBUSY = 30&
Const SE_ERR_NOASSOC = 31&
Const ERROR_BAD_FORMAT = 11&

Function StartDoc(DocName As String) As Long
    Dim Scr_hDC As Long
    Scr_hDC = GetDesktopWindow()
    StartDoc = ShellExecute(Scr_hDC, "Open", DocName, _
    "", "C:\", SW_SHOWNORMAL)
End Function

Private Sub Form_Click()
    Dim r As Long, msg As String
    r = StartDoc("C:\WINDOWS\ARCADE.BMP")
    If r <= 32 Then
        'There was an error
        Select Case r
            Case SE_ERR_FNF
                msg = "File not found"
            Case SE_ERR_PNF
                msg = "Path not found"
            Case SE_ERR_ACCESSDENIED
                msg = "Access denied"
            Case SE_ERR_OOM
                msg = "Out of memory"
            Case SE_ERR_DLLNOTFOUND
                msg = "DLL not found"
            Case SE_ERR_SHARE
                msg = "A sharing violation occurred"
            Case SE_ERR_ASSOCINCOMPLETE
                msg = "Incomplete or invalid file association"
            Case SE_ERR_DDETIMEOUT
                msg = "DDE Time out"
            Case SE_ERR_DDEFAIL
                msg = "DDE transaction failed"
            Case SE_ERR_DDEBUSY
                msg = "DDE busy"
            Case SE_ERR_NOASSOC
                msg = "No association for file extension"
            Case ERROR_BAD_FORMAT
                msg = "Invalid EXE file or error in EXE image"
            Case Else
                msg = "Unknown error"
        End Select
```

```
            MsgBox msg
        End If
    End Sub
```

۳- برنامه را اجرا نموده و روی فرم کلیک کنید.

اطلاعات کلی پیرامون مراحل عمل تابع ShellExecute

اگر تابع با موفقیت اجرا شود، مقدار برگردانده شده شناسه برنامه ای است که اجرا می شود. اگر مشکلی پیش بیاید، مقدار بازگردانده شده کوچکتر یا مساوی ۳۲ خواهد بود.

فایلی که توسط پارامتر lpszFile مشخص می شود، می تواند یک فایل اجرایی یا یک فایل غیر اجرایی باشد. اگر فایل غیر اجرایی باشد، بر حسب مقدار پارامتر lpszOp توسط برنامه مخصوص به خود باز شده یا چاپ می شود. اگر این فایل یک فایل اجرایی باشد، این تابع برنامه را اجرا می کند، حتی اگر مقدار پارامتر lpszOp برابر PRINT باشد.

جلوگیری از اجرای بیش از یک مورد از یک برنامه ۳۲ بیتی

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
185730.TXT	۱۷ آگوست ۱۹۹۸	4 , 5 , 6	32 bit

در این بخش خواهیم دید که چگونه از اجرای همزمان بیش از یک مورد از یک برنامه که قبلا یک بار اجرا شده است، جلوگیری نماییم. در روش گفته شده در زیر نشان داده شده است که چگونه هنگام اجرای مورد دوم از برنامه، کنترل اجرا به مورد اول منتقل می شود.

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید.

۲- کد زیر را در Form1 وارد کنید:

```
Option Explicit

Private Sub Form_Load()
   If App.PrevInstance Then
      ActivatePrevInstance
```

```
   End If
End Sub
```

۳- یک ماژول به پروژه اضافه کنید.

۴- کد زیر را در ماژول وارد کنید:

```
Option Explicit

Public Const GW_HWNDPREV = 3

Declare Function OpenIcon Lib "user32" (ByVal hwnd As Long) As Long
Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
   (ByVal lpClassName As String, ByVal lpWindowName As String) _
   As Long
Declare Function GetWindow Lib "user32" _
   (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Declare Function SetForegroundWindow Lib "user32" _
  (ByVal hwnd As Long) As Long

Sub ActivatePrevInstance()
   Dim OldTitle As String
   Dim PrevHndl As Long
   Dim result As Long

   'Save the title of the application.
   OldTitle = App.Title

   'Rename the title of this application so FindWindow
   'will not find this application instance.
   App.Title = "unwanted instance"

   'Attempt to get window handle using VB4 class name.
   PrevHndl = FindWindow("ThunderRTMain", OldTitle)

   'Check for no success.
   If PrevHndl = 0 Then
      'Attempt to get window handle using VB5 class name.
      PrevHndl = FindWindow("ThunderRT5Main", OldTitle)
   End If

   'Check if found
   If PrevHndl = 0 Then
   'Attempt to get window handle using VB6 class name
   PrevHndl = FindWindow("ThunderRT6Main", OldTitle)
   End If

   'Check if found
   If PrevHndl = 0 Then
      'No previous instance found.
      Exit Sub
   End If

   'Get handle to previous window.
   PrevHndl = GetWindow(PrevHndl, GW_HWNDPREV)

   'Restore the program.
   result = OpenIcon(PrevHndl)
```

```
    'Activate the application.
    result = SetForegroundWindow(PrevHndl)

    'End the application.
    End
End Sub
```

۵- پروژه را کامپایل کنید تا فایل EXE ساخته شود.

٦- از ویژوال بیسیک خارج شوید.

۷- مرحله ۷ را تکرار کنید.

☜ نتیجه: با اجرا کردن مورد دوم از برنامه ، کنترل اجرا به مورد اول منتقل شده و مورد دوم بسته می شود. اگر پنجره مورد اول مینیمایز شده باشد، به اندازه طبیعی بازگردانده می شود.

چه زمانی کنترل اجرا به/از برنامه ویژوال بیسیک منتقل می شود

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
186908.TXT	۱۷ آگوست ۱۹۹۸	5 , 6	95 / 98 / NT

ویژوال بیسیک دارای یک مکانیزم داخلی نیست که برنامه نویس بتواند از طریق آن متوجه شود چه زمانی کنترل اجرا به برنامه اش منتقل شده یا از آن گرفته شده است. در بعضی موارد ممکنست لازم باشد که از اتفاق افتادن وقایع فوق مطلع شویم. با استفاده از عملگر[10] AddressOf می توانیم کنترل پیام های ویندوز را در اختیار بگیریم تا بفهمیم چه موقع این وقایع اتفاق می افتند.

💣 اخطار: مسئولیت مشکلات احتمالی که در اثر هر گونه استفاده از مثال زیر بوجود آید متوجه خود شما خواهد بود.

---

[10] Operator

مثال زیر برنامه ای است که می تواند انتقال کنترل از برنامه به یک برنامه دیگر یا به عکس را تشخیص دهد. برای انجام این کار باید از تکنیک کنترل پیامهای ویندوز استفاده شود تا زمانی که پیام WM_ACTIVATEAPP ارسال می شود از آن مطلع شویم. برای کسب اطلاعات بیشتر در این زمینه می توانید به بخش شماره ۱۶۸۷۹۵ از بانک اطلاعاتی مایکروسافت تحت عنوان "کنترل پیامهای ویندوز بوسیله AddressOf" مراجعه کنید.

☜ توجه: در صورتیکه پیش از پاک شدن پنجره از حافظه یا از بین رفتن آن عملیات کنترل پیام ها را متوقف نکنید، خطاهایی در برنامه بوجود آمده و باعث بهم ریختن سیستم شده و درنهایت اطلاعات شما از دست خواهد رفت. دلیل این اتفاق اینست که تابع WindowProc که قرار است مورد استفاده قرار گیرد دیگر وجود ندارد. همیشه پیش از خروج از برنامه یا پاک کردن پنجره از حافظه به عملیات کنترل پیام خاتمه دهید.

زمانی که در محیط ویژوال بیسیک به رفع اشکال چنین برنامه ای مشغول هستید، این نکته اهمیت بیشتری پیدا می کند. اگر پیش از خاتمه عملیات کنترل پیام دکمه خاتمه برنامه را در محیط ویژوال بیسیک فشار دهید یا از منوی Run گزینه End را انتخاب کنید در مدیریت حافظه ویژوال بیسیک اشکالی بروز کرده و باعث بسته شدن ویژوال بیسیک می شود.

مراحل ساخت مثال:

۱- یک پروژه جدید ایجاد کنید. Form1 بطور پیش فرض ایجاد می شود.

۲- کد زیر را در Form1 وارد کنید:

```
Sub Form_Load()
    'Store handle to this form's window
    gHW = Me.hWnd

    'Call procedure to begin capturing messages for this window
    Hook
End Sub

Private Sub Form_Unload(Cancel As Integer)
   'Call procedure to stop intercepting the messages for this window
   Unhook
End Sub
```

۳- یک ماژول به پروژه اضافه کنید.

۴- کد زیر را در ماژول وارد کنید:

```
Option Explicit
```

```
Declare Function CallWindowProc Lib "user32" Alias _
   "CallWindowProcA" (ByVal lpPrevWndFunc As Long, _
   ByVal hwnd As Long, ByVal Msg As Long, _
   ByVal wParam As Long, ByVal lParam As Long) As Long

Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" _
   (ByVal hwnd As Long, ByVal nIndex As Long, _
   ByVal dwNewLong As Long) As Long

Public Const WM_ACTIVATEAPP = &H1C
Public Const GWL_WNDPROC = -4

Global lpPrevWndProc As Long
Global gHW As Long

Public Sub Hook()
    'Establish a hook to capture messages to this window
    lpPrevWndProc = SetWindowLong(gHW, GWL_WNDPROC, _
      AddressOf WindowProc)
End Sub

 Public Sub Unhook()
    Dim temp As Long

    'Reset the message handler for this window
    temp = SetWindowLong(gHW, GWL_WNDPROC, lpPrevWndProc)
 End Sub

Function WindowProc(ByVal hw As Long, ByVal uMsg As Long, _
   ByVal wParam As Long, ByVal lParam As Long) As Long
    'Check for the ActivateApp message

    If uMsg = WM_ACTIVATEAPP Then
        'Check to see if Activating the application
        If wParam <> 0 Then
            'Application Received Focus
            Form1.Caption = "Focus Restored"
        Else
            'Application Lost Focus
            Form1.Caption = "Focus Lost"
        End If
    End If

    'Pass message on to the original window message handler
    WindowProc = CallWindowProc(lpPrevWndProc, hw, uMsg, wParam, _
      lParam)
End Function
```

۵- مثال را ذخیره کرده و اجرا کنید.

٦- روی یک برنامه دیگر (مثلاً زمینه ویندوز) کلیک کنید، و توجه کنید که عنوان Form1 نشان می دهد که کنترل اجرا از برنامه به جای دیگری منتقل شده است. (اگر Form1 مینیمایز شده باشد، عنوان پنجره را در نوار عملیات ویندوز بیابید).

۷- روی پنجره مربوط به مثال کلیک کنید و توجه کنید که عنوان پنجره نشان می دهد که کنترل اجرا به برنامه منتقل شده است.

شکل 2 - کنترل اجرا روی برنامه است

شکل 3 - کنترل اجرا روی برنامه قرار ندارد

مینیمایز کردن تمامی پنجره ها از طریق ویژوال بیسیک

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
194914.TXT	۲۹ اکتبر ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

در مواردی ممکن است بخواهید از طریق برنامه نویسی تمامی پنجره های روی صفحه ویندوز را به حالت مینیمایز درآورید. این کار به کمک تابع keybd_event به راحتی امکان پذیر است.

ترفند به کار رفته در این مثال اینست که عمل فشار کلیدهای لازم برای احضار منوی Popup مربوط به نوار عملیات ویندوز، و سپس فشار کلید M در این منو که گزینه

Minimize All Windows را انتخاب می کند، شبیه سازی می شود. این کار با سه بار فراخوانی تابع keybd_event انجام می شود.

پارامتر دوم در فراخوانی keybd_event کد اسکن سخت افزاری است و در مورد این مثال مقدار ۹۱ باید بکار رود. با این حال، از آنجا که برنامه های کاربردی نباید از کدهای اسکن استفاده کنند، باید این مقدار را صفر قرار دهید.

۱- یک پروژه جدید ایجاد کنید. Form1 بطور پیش فرض ایجاد می شود.

۲- یک CommandButton روی این فرم قرار دهید.

۳- کد زیر را در Form1 وارد کنید:

```
Private Declare Sub keybd_event Lib "user32" ( _
   ByVal bVk As Byte, _
   ByVal bScan As Byte, _
   ByVal dwFlags As Long, _
   ByVal dwExtraInfo As Long)

Const KEYEVENTF_KEYUP = &H2
Const VK_LWIN = &H5B

Private Sub Command1_Click()
   ' 77 is the character code for the letter 'M'
   Call keybd_event(VK_LWIN, 0, 0, 0)
   Call keybd_event(77, 0, 0, 0)
   Call keybd_event(VK_LWIN, 0, KEYEVENTF_KEYUP, 0)
End Sub
```

۴- برای اجرای برنامه کلید F5 را فشار دهید و روی Command1 کلیک کنید. تمامی پنجره های موجود در ویندوز به حالت مینیمایز در می آیند.