by Jonathan Zuck

HOW DO YOU HANDLE STRESS?

You're probably aware that stress on system resources such as memory, file handles, and disk space can adversely affect application performance. Memory availability affects your ability to create arrays and load modules. System resources constrict your use of windows and graphical objects. Disk space and file handles affect your access to the system.

While lots of articles have been written on how to conserve these precious system resources, no one is writing about how to use more of them. Using more system resources sounds silly, but if you could consume resources in a controlled fashion, you'd have a great testing tool. For example, how do you know how your application functions in a low-resource environment unless you try it? Of course, the obvious solution is to run Word 6.0, but this is hardly a controlled test. Instead, we will use the Stress API found in Windows 3.1.

You never expected to encounter a Stress API, did you? The purpose of the API is low-resource testing, although the only application that exploits it comes with Visual C++, not with Visual Basic. The Stress API contains several consumption functions, for lack of a better term, that eat up memory, resources, disk space, and file handles. An additional function counts the available file handles (see Table 1).

These functions are not difficult to declare or use, but putting them together in a utility certainly makes them more useful. Displaying current information can be tricky. I created a module, STRESS.BAS, with all of the necessary declarations for the Stress API as well as some related API and utility functions (see Listing 1), and constructed an application called "Argh." (Download this application and source code, ARGH.ZIP, from the Magazine library of the *Visual Basic Programmer's Journal* Forum on Compu–Serve. ARGH.ZIP is also on Volume 1, Number 3 of the VB-CD Quarterly.)

MEASURE THE IMPACT OF SYSTEM RESOURCE USAGE ON YOUR PROGRAMS WITH THE WINDOWS STRESS API.

The Stress API Functions

AllocDiskSpace	Creates a file to consume space on a disk partition.
AllocFileHandles	Allocates up to 256 file handles.
AllocGDIMem	Allocates memory in the GDI heap.
AllocMem	Allocates global memory.
AllocUserMem	Allocates memory in the User heap.
FreeAllGDIMem	Frees all memory allocated by the AllocGDIMem function.
FreeAllMem	Frees all memory allocated by the AllocMem function.
FreeAllUserMem	Frees all memory allocated by the AllocUserMem function.
GetFreeFileHandles	Returns the number of free file handles.
UnAllocDiskSpace	Deletes file created by AllocDiskSpace and frees space.
UnAllocFileHandles	Frees file handles allocated by AllocFileHandles.

TABLE 1 ***Test All Kinds of Consumption.*** *The Stress API functions allow testing of disk space, resources, and system memory by consuming these resources and allowing you to return the current status of memory resources.*

MANAGE YOUR STRESS

STRESS.BAS is a reusable utility that allows you to selectively stress your system to see how it handles low-resource and low-memory situations. It also encourages you to learn how the Stress API operates. The main window displays the current available memory and resources. The information is updated every five seconds (see Figure 1). The system then displays information about global memory, User memory, GDI memory, disk space, and available file handles:

```
Sub Form_Paint ()
   lblHandles = GetFreeFileHandles()
   lblGMem = _
      Format(GetFreeSpace(0) \ 1024, "###,###,###") + "K"
   lblUserMem = GetFreeHeap("User") & " bytes"
   lblGDIMem = GetFreeHeap("GDI") & " bytes"
   lblDiskSpace = _
      Format(DiskSpaceFree() \ 1024, "###,###,###") + "K"
End Sub
```

By selecting Options from the Stress menu, you can determine how you want the system to be stressed (see Figure 2). The default is a static test in which you simply specify how much of each resource you want to remain. The default values are the values when Argh was started, so you can leave them alone if you don't want them changed. Enter a new value to specifiy how much of each of these you want left on the system when Argh is active.

Be aware that both global memory and disk space may be rounded if the number you choose doesn't fall on normal

Jonathan Zuck is vice president of client/server technology for Advanced Paradigms Inc. in Alexandria, Virginia. He is a regular contributor to Visual Basic Programmer's Journal *and a co–author of* Visual Basic How-To, *published by Waite Group Press. Jonathan can be reached on CompuServe at 76702,1605.*

boundaries. In the disk section of the options form, you can indicate whether you want memory eaten from the Windows disk, the current disk, or the temporary disk, if they are indeed three different entities.

Once you have made your selections, choose OK. To start stressing the system, select Go from the Stress menu (this activates the code in Listing 2). You can then run your application and see how it handles the situation you created. Again, be aware that if you set some of these values too low they will affect actual system performance as well and painting of the desktop might be affected. For this reason, the Restore selection from the Stress menu can be activated with a shortcut key combination, Ctrl-R, when you can't access the menu.

In addition to a so-called static test, it is possible to make Argh randomly consume selected resources so that it is more difficult to predict when things will go wrong in your applications. Choose Random from the Stress section of the Options form to enable the text boxes in the Max column. Use these pairs of check boxes to specify a range for each resource. In this mode, Argh will randomly choose a value in the range and change resource settings to that value. Again, Ctrl-R stops the testing and restores all resources. Argh uses this code to restore all of your system resources to their original state:

```
Sub mnuStressRestore_Click ()
   tmrStress.Enabled = False
   UnAllocDiskSpace uDrive
   UnAllocFileHandles
   FreeAllGDIMem
   FreeAllUserMem
   FreeAllMem
   Me.Refresh
   mnuStressGo.Enabled = True
   mnuStressOptions.Enabled = True
End Sub
```

System memory is usually less scarce than resources, but when you start to run out of memory, your applications will have problems. Windows makes a lot of global memory available through its virtual memory mechanism, which treats disk space as additional memory. As a result, you'll rarely use up too much of your global memory. However, if a system has 4 MB, and the memory is more virtual than real, serious slowdowns can occur. You can test your application in low memory circumstances with Argh, even if your system has 32 MB.

Argh consumes global memory with the AllocMem function in STRESS.DLL (see Listing 2). To use this function, you simply pass the amount of memory you want remaining to the AllocMem function. To release this memory, simply call FreeAllMem. Argh uses the GetFreeSpace API function to determine the amount of available global memory.

EMPTY YOUR POCKETS

A more frequent problem than memory is the lack of system resources available to your application and others. This has been the bane of Windows users for some time and is hopefully

```
STRESS.BAS
'declarations for the STRESS API in Windows 3.1
'by Jonathan Zuck/Visual Basic Programmer's Journal

Declare Function AllocDiskSpace Lib "STRESS.DLL" _
  (ByVal lLeft As Long, ByVal uDrive As Integer) _
     As Integer

Global Const EDS_WIN = 1
Global Const EDS_CUR = 2
Global Const EDS_TEMP = 3

Declare Sub UnAllocDiskSpace Lib "STRESS.DLL" _
  (ByVal uDrive As Integer)

Declare Function GetFreeFileHandles Lib _
  "STRESS.DLL" () As Integer
Declare Function AllocFileHandles Lib _
  "STRESS.DLL" (ByVal nLeft As Integer) As Integer
Declare Sub UnAllocFileHandles Lib "STRESS.DLL" ()
Declare Function AllocGDIMem Lib "STRESS.DLL" _
  (ByVal uLeft As Integer) As Integer
Declare Sub FreeAllGDIMem Lib "STRESS.DLL" ()
Declare Function AllocMem Lib "STRESS.DLL" (ByVal _
  dwLeft As Long) As Integer
Declare Sub FreeAllMem Lib "STRESS.DLL" ()
Declare Function AllocUserMem Lib "STRESS.DLL" _
  (ByVal uLeft As Integer) As Integer
Declare Sub FreeAllUserMem Lib "STRESS.DLL" ()

'other declares which are relevant
Declare Function GetFreeSpace Lib "Kernel" (ByVal _
  wFlags As Integer) As Long
Declare Function GetModuleHandle Lib "Kernel" _
  (ByVal ModName As String) As Integer
Declare Function GetHeapSpaces Lib "Kernel" _
  (ByVal hModule As Integer) As Long
Declare Function DiskSpaceFree Lib "SETUPKIT.DLL" _
  () As Long

Function GetFreeHeap (ModuleName As String) As Long
  Dim rInfo As Long

  rInfo = GetHeapSpaces(GetModuleHandle(ModuleName))
  GetFreeHeap = LoWord(rInfo)
End Function

Function HiWord (LongInt As Long) As Long
  Temp = LongInt \ &H10000
  If Temp < 0 Then Temp = Temp + &H10000
  HiWord = Temp
End Function

Function L2I (L As Long) As Integer
  L2I = IIf(L > 32768, (L - 32768) * -1, L)
End Function

Function LoWord (LongInt As Long) As Long
  Temp = LongInt Mod &H10000
  If Temp < 0 Then Temp = Temp + &H10000
  LoWord = Temp
End Function
```

LISTING 1 ***Get Stressed Out.*** *The STRESS.DLL functions allow you to selectively "stress" your system by eating up resources. As these resources dissappear, it's easy to see how your application handles low-resource and low-memory situations.*

```
Sub mnuStressGo_Click ()
  mnuStressOptions.Enabled = False
  mnuStressRestore.Enabled = True
  mnuStressGo.Enabled = False

  'now start stressing out!
  If frmconfig.fraStress.Tag = "Random" Then
     tmrStress.Enabled = True
     Exit Sub
  End If

  'if they have chosen a static test then do it here
   Ok = AllocMem(Val(frmconfig.txtMin(0)))
   Ok = AllocUserMem(L2I(Val(frmconfig.txtMin(1))))
   Ok = AllocGDIMem(L2I(Val(frmconfig.txtMin(2))))
   Ok = AllocFileHandles(Val(frmconfig.txtMin(3)))
   Ok = AllocDiskSpace(Val(frmconfig.txtMin(4)), _
     Val(frmconfig.fraDisk.Tag))
End Sub
```

LISTING 2 ***Choosing Between Static and Random Testing.*** *The mnuStressGo_Click routine activates the type of test chosen by the user. When testing for static memory conditions, pass the amount of memory you want consumed to the function.*

being addressed in the next version of Windows, as it has been in NT 3.5. Barry Seymour and Tim O'Pry explained in the February issue that both window and graphical resources are constrained by a 64K heap limit in USER.EXE and GDI.EXE ["Conserving Windows Resources," *VBPJ* February 1995]. They both maintain their object lists in this space. System resources are a measure of the percentage of this space remaining.

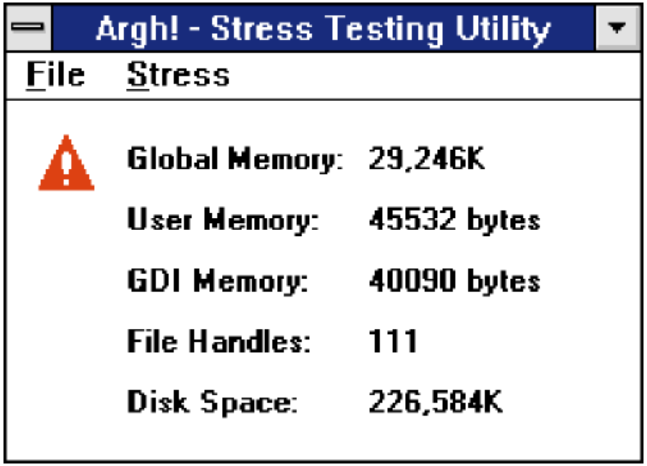
HOW DO YOU KNOW
HOW YOUR APPLICATION WORKS
IN A LOW-RESOURCE ENVIRONMENT
UNLESS YOU TRY IT?

While there is now an API function, SystemHeapInfo, for getting at these percentages, there isn't a documented way to get at the actual values. To display these values, I've gone back to the old "undocumented" way of determining system resources: the GetHeapSpaces API function. This function takes a module handle and returns the total and remaining bytes in the heap for that module. You can get a module handle by calling the documented GetModuleHandle function. I've used these functions in the GetFreeHeap utility function in STRESS.BAS:

```
Function GetFreeHeap (ModuleName As String) As Long
  Dim rInfo As Long

  rInfo = GetHeapSpaces(GetModuleHandle(ModuleName))
  GetFreeHeap = LoWord(rInfo)
End Function
```

One trick to using this function is that GetHeapSpaces returns a long integer, the hiword of which contains the total heap space and the loword of which contains the remaining bytes. Two problems encountered when trying to use this function are

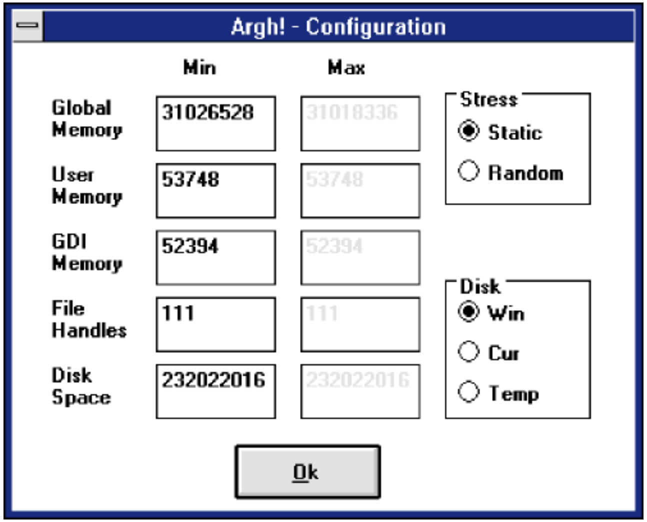


```
Global Memory:   29,246K
User Memory:     45532 bytes
GDI Memory:      40090 bytes
File Handles:    111
Disk Space:      226,584K
```

FIGURE 1 ***The Argh Main Screen.*** *The information about available global memory, User memory, GDI memory, disk space, and file handles is updated every five seconds.*

that VB doesn't have hiword/loword functions, nor does it have words at all. Therefore, I wrote the hiword/loword utility functions that convert the integer values to longs in order to account for negative values. Later, when the testing routine needs to retrieve these values, they are converted back to integers from longs with the L2I function (see Listing 1).

The STRESS.DLL functions for manipulating resources are



	Min	Max
Global Memory	31026528	31018336
User Memory	53748	53748
GDI Memory	52394	52394
File Handles	111	111
Disk Space	232022016	232022016

Stress: [✓ Static] Random

Disk: [✓ Win] Cur Temp

FIGURE 2 ***Argh Configuration.*** *You can test memory in a static or random fashion. When the random test is used, you need to specify the range of values in which to constrain the resource consumption.*

AllocUserMem, AllocGDIMem, FreeAllUserMem and FreeAll–GDIMem. Again, the Alloc* functions take a single parameter to specify the number of bytes to leave in the respective heap. These are the functions likely to have the greatest effect on your system.

Besides monitoring the impact of system memory use, the Stress API also includes functions for allocating disk space and file handles. The AllocDiskSpace function takes two parameters: the disk on which to consume space, and the number of bytes to leave there. AllocDiskSpace then creates a file called STRESS.EAT on the specified drive. If you have a system crash, you should look for this file and delete it before restarting Windows.

In order to display the currently remaining space on disk, Argh uses the DiskFreeSpace function found in the SETUPKIT.DLL. To use this utility, make sure that both STRESS.DLL and SETUPKIT.DLL can be accessed. SETUPKIT.DLL is included with Visual Basic 3, and can be found in \VB\SETUPKIT\KITFILES.

Because the process of counting available file handles is quite complex, especially under Windows, the Stress API includes a function called GetFreeFileHandles that returns the number of file handles currently available. The AllocFileHandles and FreeFileHandles functions are called to manipulate these. It's pretty hard to run out of file handles because you have more than 100 available, but the more robust you can make your application the better.

Argh could benefit from a number of improvements, such as storing previous choices in an INI file and logging resource changes to disk to track when a fatal error occurred in random mode. Still, the Stress API is a good starting point for building more robust applications. ■