

Kurapica

.NET Reversing Tips

7

Memory Optimization



Published for educational Purposes only

Hi every one...

I'm glad I'm here again and I hope you enjoy this new tutor, as you can see this is not a reversing tutor, we are not cracking anything today, as a change we will talk a technical problem regarding .NET applications, If you ever wrote an application using .NET platform then you must have noticed that these applications tend to consume a considerable amount of RAM!

To prove this fact you can simply create a do-nothing application in .NET and just run the application, now go to Windows task manager and find your application and check the amount of RAM taken by your do-nothing Application, usually this type of applications require from 15 to 20 Mb of RAM!, this is not good compared to a Delphi executable which takes about 3 to 5 MB of RAM.

I've recently written a memory manger class for my applications and it you have to do is to create a new instance of MemoryManager class in your main form and it will do the job, It's built on a very simple idea and it takes advantage of the OnIdle event in the .net application, the OnIdle event is raised regularly whenever your application is not busy for a while, the MemoryManager class will dramatically reduce the amount of memory used by your application by automatically releasing the memory no longer needed, which will free up more RAM for other applications.

Next I will list the source code for the class and I will explain some points too.

```
Public Class MemoryManager
```

```
    ' APIs
```

```
    Declare Function SetProcessWorkingSetSize Lib "kernel32" Alias  
        "SetProcessWorkingSetSize" (ByVal process As IntPtr, ByVal  
        minimumWorkingSetSize As Integer, ByVal maximumWorkingSetSize As  
        Integer) As Integer
```

```
    ' Fields
```

```
    Private lastFlush As Long = DateTime.Now.Ticks
```

```
    ' Methods
```

```
    Public Sub New()
```

```
        If (Environment.OSVersion.Platform = PlatformID.Win32NT) Then
```

```
            AddHandler Application.Idle, New EventHandler(AddressOf  
                Me.OnIdle)  
            Me.Flush()
```

```
        End If
```

```
    End Sub
```

```
    Private Sub Flush()
```

```
        Try
```

```
            Dim PR As Process = Process.GetCurrentProcess  
            MemoryManager.SetProcessWorkingSetSize(PR.Handle, -1, -1)  
            PR.Dispose()
```

```
        Catch ex As Exception
```

```
        End Try
```

```
    End Sub
```

```
    Private Sub OnIdle(ByVal sender As Object, ByVal e As EventArgs)
```

```
        Dim ticks As Long = DateTime.Now.Ticks  
        If ((ticks - Me.lastFlush) > &H989680) Then  
            Me.lastFlush = ticks  
            Me.Flush()  
        End If
```

```
    End Sub
```

```
End Class
```

APIs

SetProcessWorkingSetSize:

The SetProcessWorkingSetSize function sets the minimum and maximum working set sizes for a specified process.

The working set of a process is the set of memory pages currently visible to the process in physical RAM memory. These pages are resident and available for an application to use without triggering a page fault. The size of the working set of a process is specified in bytes. The minimum and maximum working set sizes affect the virtual memory paging behavior of a process.

BOOL SetProcessWorkingSetSize(

```
HANDLE hProcess, // open handle to the process of interest
DWORD dwMinimumWorkingSetSize, // specifies minimum working set size
DWORD dwMaximumWorkingSetSize // specifies maximum working set size
);
```

Parameters

hProcess

An open handle to the process whose working set sizes will be set.

Windows NT: The handle must have PROCESS_SET_QUOTA access rights. For more information, see Process Objects.

dwMinimumWorkingSetSize

Specifies a minimum working set size for the process. The virtual memory manager attempts to keep at least this much memory resident in the process whenever the process is active.

If both dwMinimumWorkingSetSize and dwMaximumWorkingSetSize have the value 0xffffffff, the function temporarily trims the working set of the specified process to zero. This essentially swaps the process out of physical RAM memory.

dwMaximumWorkingSetSize

Specifies a maximum working set size for the process. The virtual memory manager attempts to keep no more than this much memory resident in the process whenever the process is active and memory is in short supply.

If both dwMinimumWorkingSetSize and dwMaximumWorkingSetSize have the value 0xffffffff, the function temporarily trims the working set of the specified process to zero. This essentially swaps the process out of physical RAM memory.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. Call `GetLastError` to obtain extended error information.

Remarks

The working set of the specified process can be emptied by specifying the value `0xffffffff` for both the minimum and maximum working set sizes.

If the values of either `dwMinimumWorkingSetSize` or `dwMaximumWorkingSetSize` are greater than the process' current working set sizes, the specified process must have the `SE_INC_BASE_PRIORITY_NAME` privilege. Users in the Administrators and Power Users groups generally have this privilege. For more information about security privileges, see [Privileges](#).

The operating system allocates working set sizes on a first-come, first-served basis. For example, if an application successfully sets 40 megabytes as its minimum working set size on a 64-megabyte system, and a second application requests a 40-megabyte working set size, the operating system denies the second application's request.

Using the `SetProcessWorkingSetSize` function to set an application's minimum and maximum working set sizes does not guarantee that the requested memory will be reserved, or that it will remain resident at all times. When the application is idle, or a low-memory situation causes a demand for memory, the operating system can reduce the application's working set. An application can use the `VirtualLock` function to lock ranges of the application's virtual address space in memory; however, that can potentially degrade the performance of the system.

When you increase the working set size of an application, you are taking away physical memory from the rest of the system. This can degrade the performance of other applications and the system as a whole. It can also lead to failures of operations that require physical memory to be present; for example, creating processes, threads, and kernel pool. Thus, you must use the `SetProcessWorkingSetSize` function carefully. You must always consider the performance of the whole system when you are designing an application.

How it works

The idea is very simple, we just add an event handler to find out when the OnIdle event of the application is raised, and when this happens we call the flush Procedure which reduces the amount of ram consumed by the calling process, this event should be raised at most once every 10 seconds.

The ticks variable in Procedure OnIdle is the number of 100-nanosecond intervals that have elapsed since 12:00 A.M., January 1, 0001, so we make sure that at least 10 seconds have elapsed since last flush.

All you have to do is to create an instance of this class in your main form like this

```
Dim MeMan as new MemoryManager
```

THE END

Greetings fly to every one who reads this

Kurapica

Thursday, March 22, 2007