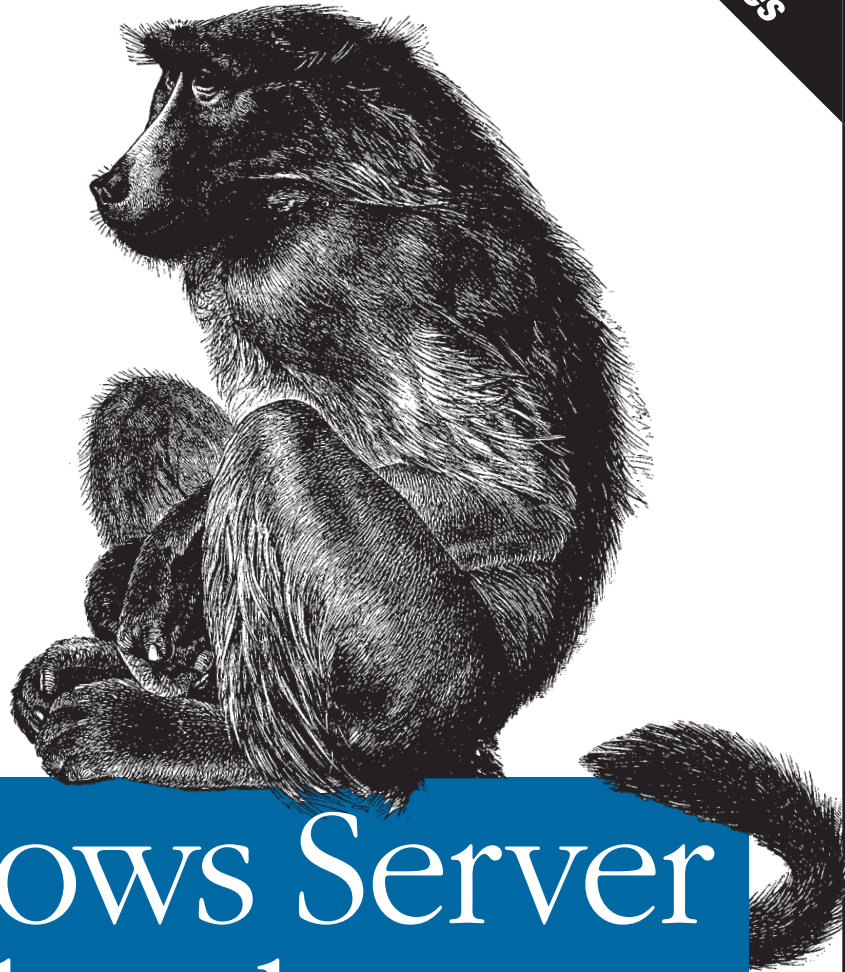


*Step-by-Step Procedures and Scripts  
for Every Occasion*

**Over  
300 Recipes**



# Windows Server Cookbook

*for Windows Server 2003 and Windows 2000*

**O'REILLY®**

*Robbie Allen  
Foreword by Mark Russinovich*

# Processes

## 6.0 Introduction

Processes are a fundamental component of the Windows operating system. Anything you do on a system, whether it is deleting a file, starting a service, or writing text in Notepad, has a process behind it. Since processes are so important, it is critical that administrators understand how to manage, monitor, and troubleshoot them.

Processes use system resources, such as CPU and memory, in order to run. But not all processes are created equal. Some use more resources than others and often you'll run into situations where you need to identify processes that are using more resources than they should, which may make it difficult for other processes to do work. Processes also frequently open files, DLLs, and Registry keys and values. These resources are known as *handles* and often when a process has one open, no other process can modify or delete the resource. This can make it problematic if you need, for example, to rename a file that a process has locked.

In Appendix E, I include a list of the default processes used in Windows. There are several processes that start by default whenever a Windows server boots. Any applications you've installed that run at system startup will also have one or more processes running, all without you doing a thing. It is for this reason that you need to be able to create, query, suspend, and terminate processes on demand or else it is very easy for you to lose control over how your system performs.

In this chapter, I'll review how to identify process-related issues and cover many of the process-related tasks you should be familiar with. Now you may not need to use some of these tasks, but it is important to understand what is possible so if you run into a certain situation where, for example, you need to suspend a process, you know how to do it.

## Using a Graphical User Interface

When it comes to the GUI, there are only two tools you need to be familiar with to manage processes. Task Manager (*taskmgr.exe*) is a native Windows tool that lets you view and kill any running applications or processes, and lets you view the performance of processes including CPU and memory utilization. Task Manager was updated in Windows Server 2003 to include a new Networking tab and Users tab. The Networking tab lets you view the current network activity of the system (although it doesn't show network information by process). The Users tab lets you see which users are currently logged on and lets you disconnect or log them off.



You can type the **Ctrl-Shift-Esc** sequence to launch Task Manager quickly.

The other tool is Process Explorer (*procexp.exe*) from Sysinternals, and it is very similar to Task Manager except it provides much more process management functionality. It lets you view all the associated handles and DLLs of a process and even lets you search for specific processes, handles, and DLLs. Neither tool lets you manage processes on a remote server. To do that, you'll need to use one of the available command-line tools, of which there are many.

## Using a Command-Line Interface

There are several process-related command-line tools, many of them from Sysinternals. Windows 2000 didn't provide any good process management utilities natively, but there were a few in the Resource Kit. In Windows XP and Windows Server 2003, Microsoft added the *tasklist* and *taskkill* utilities, which are installed with Windows and are very powerful. For advanced process manipulation and query tools, look no farther than Sysinternals. See Table 6-1 for a complete list of command-line tools used in this chapter along with where they can be found and what Recipes they are used in.

Table 6-1. Command-line tools used in this chapter

Tool	Windows Server 2003	Windows 2000	Recipes
<i>handle</i>	Sysinternals	Sysinternals	6.11
<i>listdlls</i>	Sysinternals	Sysinternals	6.9
<i>netstat</i>	%SystemRoot%\system32	%SystemRoot%\system32	6.12
<i>netstatp</i>	Sysinternals	Sysinternals	6.12
<i>portqry</i>	MS KB 310099	MS KB 310099	6.12
<i>pslist</i>	Sysinternals	Sysinternals	6.4, 6.5, 6.8
<i>pskill</i>	Sysinternals	Sysinternals	6.3, 6.14
<i>psuspend</i>	Sysinternals	Sysinternals	6.2

Table 6-1. Command-line tools used in this chapter (continued)

Tool	Windows Server 2003	Windows 2000	Recipes
<i>taskkill</i>	%SystemRoot%\system32	N/A	6.3, 6.14
<i>tasklist</i>	%SystemRoot%\system32	N/A	6.4, 6.5, 6.6, 6.7, 6.13
<i>tlist</i>	N/A	Resource Kit Supplement 1	6.5, 6.6
<i>top</i>	N/A	Resource Kit Supplement 1	6.4
<i>wmic</i>	%SystemRoot%\system32	N/A	6.1, 6.4, 6.7

## Using VBScript

The Win32\_Process WMI class represents individual processes and is the only class I use extensively in this chapter. With it, you can create, terminate, and set the priority of a process. Additionally, you can get very detailed information about each process using the properties of Win32\_Process objects. For your convenience, I've included the complete list of methods and properties available with Win32\_Process in Tables 6-2 and 6-3.

Table 6-2. Win32\_Process methods

Name	Description
AttachDebugger	Launches the registered debugger for the process.
Create	Creates a new process.
GetOwner	Returns the user name that is running the process.
GetOwnerSid	Returns the user security identifier (SID) that is running the process.
SetPriority	Changes the priority of the process.
Terminate	Kills the process.

Table 6-3. Win32\_Process properties

Name	Description
Caption	Name of the process executable (e.g., notepad.exe).
CommandLine	Command line used to start the process.
CreationDate	Date the process was initially executed.
CSName	Name of the computer running the process.
Description	Name of the process executable (e.g., notepad.exe).
ExecutablePath	Path to the process executable.
Handle	Process ID (PID) of the process.
HandleCount	Total number of handles currently open by the process.
KernelModeTime	The amount of time (in 100 nanosecond units) the process has spent in kernel mode.
MaximumWorkingSetSize	Maximum working set size of the process.
MinimumWorkingSetSize	Minimum working set size of the process.
Name	Name of the process executable (e.g., notepad.exe).

Table 6-3. Win32\_Process properties (continued)

Name	Description
OtherOperationCount	Number of I/O operations performed by the process that were neither read nor write operations.
OtherTransferCount	Amount of data transferred (in bytes) by the process that were neither read nor write operations.
PageFaults	Number of total page faults the process has generated.
PageFileUsage	Amount of page file space (in kilobytes) that the process is using.
ParentProcessId	Process ID (PID) of the parent process.
PeakPageFileUsage	Peak amount of page file space (in kilobytes) that the process has used.
PeakVirtualSize	Peak virtual address space (in bytes) that the process has used.
PeakWorkingSetSize	The peak working set of a process (in kilobytes).
Priority	Current priority of the process ranging from 0 to 31 (0 is the lowest and 31 is the highest).
PrivatePageCount	Number of pages allocated to the process.
ProcessId	Process ID (PID) of the process.
QuotaNonPagedPoolUsage	Quota usage of nonpaged pool for the process.
QuotaPagedPoolUsage	Quota usage of paged pool for the process.
QuotaPeakNonPagedPoolUsage	Quota usage of peak nonpaged pool for the process.
QuotaPeakPagedPoolUsage	Quota usage of peak paged pool for the process.
ReadOperationCount	Number of read operations performed by the process.
ReadTransferCount	Amount of data read (in bytes) by the process.
SessionId	Session ID that initiated the process.
TerminationDate	Date the process was terminated. A handle to the process must be held open in order to get this.
ThreadCount	Number of threads the process has opened.
UserModeTime	The amount of time (in 100 nanosecond units) the process has spent in user mode.
VirtualSize	Current size of the virtual address space (in bytes) that a process is using.
WorkingSetSize	Amount of memory (in bytes) used for the working set of the process.
WriteOperationCount	Number of write operations performed by the process.
WriteTransferCount	Amount of data written (in bytes) by the process.

Another process-related WMI class that may be of interest to you is Win32\_ProcessStartup. While I don't cover it in this book, you may find it useful if you need control over how processes are created. You can pass an instance of the Win32\_ProcessStartup class as a parameter when you invoke the Win32\_Process.Create method. It allows you to specify various window settings and the priority of the new process. Search for Win32\_ProcessStartup at <http://msdn.microsoft.com/> for more information.

## 6.1 Setting the Priority of a Process

### Problem

You want to raise or lower the priority of a process. This is beneficial if you want to boost a CPU-starved process or limit a process that is hogging the CPU.

### Solution

#### Using a graphical user interface

1. Open Windows Task Manager (*taskmgr.exe*).
2. Click on the **Processes** tab.
3. If you do not see the process you want to set, be sure the box beside **Show processes from all users** is checked.
4. Right-click on the target process, select **Set Priority**, and select the desired priority.

You can also accomplish the same task using the Sysinternals Process Explorer (*proccexp.exe*) tool: right-click on the process and select **Set Priority**.

#### Using a command-line interface

With the *start* command, you can set the priority of a process when you initially run it. The following example shows how to create a process with a *high* priority:

```
> start /HIGH <ProgramPath>
```

The other valid priority options include */LOW*, */NORMAL*, */REALTIME*, */ABOVENORMAL*, and */BELOWNORMAL*.

#### Using VBScript

```
' This code sets the priority of a process

Const NORMAL          = 32
Const IDLE            = 64
Const HIGH_PRIORITY  = 128
Const REALTIME       = 256
Const BELOW_NORMAL   = 16384
Const ABOVE_NORMAL    = 32768

' ----- SCRIPT CONFIGURATION -----
strComputer = "."
intPID      = 3280          ' set this to the PID of the target process
intPriority  = ABOVE_NORMAL ' Set this to one of the constants above
' ----- END CONFIGURATION -----
WScript.Echo "Process PID: " & intPID
set objWMIPProcess = GetObject("winmgmts:\\." & strComputer & _
                              "\root\cimv2:Win32_Process.Handle=" & intPID & ".")
WScript.Echo "Process name: " & objWMIPProcess.Name
```

```
intRC = objWMIProcess.SetPriority(intPriority)
if intRC = 0 Then
    Wscript.Echo "Successfully set priority."
else
    Wscript.Echo "Could not set priority. Error code: " & intRC
end if
```

## Discussion

Windows 2000 and Windows Server 2003 use a priority-driven, preemptive scheduling system. This means that processes are given priorities and those with higher priorities get more CPU time and subsequently, run quicker. This is useful for administrators to know because in certain situations, it can be advantageous to play with the priorities of processes to get the desired result from your system. For example, if you find a process that is pegging the CPU on a system, if you are able to run Task Manager, you can lower the priority of that process, which should help you launch other processes or applications to diagnose and troubleshoot the badly behaving process. Another way to tackle this problem is to suspend the process, which I describe in Recipe 6.2.

Windows supports six priority classes. The following is a list of the classes and their corresponding numeric value:

### *Realtime (24)*

This is the highest priority class. Be extremely careful when setting this priority because it preempts all other non-realtime processes, including operating system processes. A realtime process can interrupt normal functioning of the computer, including mouse and keyboard I/O and disk read/writes. Note that realtime priority does not make a program run in realtime, it simply gives as much CPU time as the program can use and is available.



Windows operating systems are not considered real-time operating systems because they do not (and cannot) give performance guarantees.

### *High (13)*

This priority class indicates that the process needs to perform time-sensitive functions that must be executed immediately upon being called. An example application that uses this priority is Task Manager, which needs a higher priority than normal processes so you can kill any that are CPU bound. Be careful when setting this priority class because a high priority process that is CPU bound can tie up the system indefinitely.

### *Above Normal (10)*

This is an intermediate priority class that is above Normal but below High.

### Normal (8)

This is the default priority class assigned to applications that do not require any special scheduling needs.

### Below Normal (6)

This is an intermediate priority class that is above Idle but below Normal.

### Idle or Low (4)

This priority indicates that the process only runs when the system is idle. Screen saver is an example application that utilizes this priority.

To view the current priority of all processes, run the following command:

```
> wmic process get name,processid,priority
```

You can also see the priority of processes in *pslist* output (the Pri column).

## See Also

Recipe 6.2, MS KB 110853 (PRB: Can't Increase Process Priority), and MS KB 193846 (HOWTO: Modify the Process Priority of a Shelled Application)

## 6.2 Suspending a Process

### Problem

You want to suspend a process from running. This is helpful if you want to temporarily stop an application from running, perhaps due to high CPU consumption, but you don't want to kill it. This can give you an opportunity to launch further diagnostic utilities to troubleshoot the process.

### Solution

#### Using a graphical user interface

Open the Sysinternals Process Explorer tool (*procexp.exe*). To suspend a process, right-click on the target process and select **Suspend**. To resume a process, right-click on the target process and select **Resume**.

#### Using a command-line interface

The following command suspends a process:

```
> pssuspend <PID>
```

Replace <PID> with the process ID of the target process.

The following command resumes a suspended process:

```
> pssuspend -r <PID>
```

## Using VBScript

Currently, no scripting API supports suspending processes. However, you can use the `SuspendThread` and `ResumeThread` functions that are defined in *kernel.lib* if you are using a high-level language such as Visual Basic or C++.

## Discussion

Applications are much better behaved these days than they were a few years ago, but that still doesn't mean you won't see one from time to time peg the CPU on a system and render it virtually useless. If this happens on a remote system, it can be difficult to even use Terminal Services to access the machine. So what can you do? Well, if you can find out which process is causing the problem—perhaps by using one of the methods in Recipe 6.4—you can use *pssuspend* to remotely suspend it. You can also specify alternate credentials using the `-u` (user) and `-p` (password) options. If you specify `-u` without `-p`, it will prompt you to enter the password (this is the more secure way to do it). Here is an example command line that does this:

```
> pssuspend \\jamison -u rallen notepad.exe
```

## See Also

Recipe 6.4

## 6.3 Killing a Process

### Problem

You want to terminate a process. Even though Windows has come a long way in the past 10 years, the operating system can't prevent buggy or poorly written applications from becoming unresponsive.

### Solution

#### Using a graphical user interface

1. Open the Windows Task Manager (*taskmgr.exe*).
2. Click on the **Processes** tab.
3. If you do not see the process you want to set, be sure the box beside **Show processes from all users** is checked.
4. Right-click on the target process, select **End Process**, and select the desired priority.



You can also accomplish the same task using the Sysinternals Process Explorer (*procexp.exe*) tool by right-clicking the process and selecting **Kill Process**.

## Using a command-line interface

The following command kills a process by PID:

```
> taskkill -pid <PID>
```

And this command kills a process by name on a remote server:

```
> taskkill /s <ServerName> -im <ProcessName>
```

Use the */f* option to forcefully kill the process.

The *pskill.exe* utility works in a very similar manner. Here are two examples:

```
> pskill <PID>
> pskill \\<ServerName> <ProcessName>
```

## Using VBScript

```
' This code terminates the specified process.
' ----- SCRIPT CONFIGURATION -----
intPID = 2560 ' PID of the process to terminate
strComputer = "."
' ----- END CONFIGURATION -----
WScript.Echo "Process PID: " & intPID
set objWMIPProcess = GetObject("winmgmts:\\." & strComputer & _
                             "\root\cimv2:Win32_Process.Handle='" & intPID & "'")
WScript.Echo "Process name: " & objWMIPProcess.Name
intRC = objWMIPProcess.Terminate()
if intRC = 0 Then
    Wscript.Echo "Successfully killed process."
else
    Wscript.Echo "Could not kill process. Error code: " & intRC
end if
```

## Discussion

Manually killing processes is not something you should be in the habit of doing, but it is a necessary evil of system administration. Be selective about forcibly killing a process, because it will also terminate any child processes in an ungraceful manner and can leave lingering remnants of the process in memory, which may cause problems if you attempt to restart the process later.



There are some processes that you won't be able to manually terminate. Generally this applies to system-level process, such as *lsass*.

## 6.4 Viewing the Running Processes

### Problem

You want to see all processes that are currently running on a system.

### Solution

#### Using a graphical user interface

1. Open the Windows Task Manager (*taskmgr.exe*).
2. Click on the **Processes** tab.

You can also accomplish the same task using the Sysinternals Process Explorer (*procxp.exe*) tool.

#### Using a command-line interface

There are several options for viewing the running processes via the command line. You can use *tasklist.exe* on Windows XP and Windows Server 2003 (use the */S* option to target a remote system):

```
> tasklist
```

Another Windows XP and Windows Server 2003 tool that you can use to get a process list is *wmic* as shown here (use the */node:* option to target a remote system):

```
> wmic process list brief
```

The Sysinternals *pslist.exe* utility is available for Windows Server 2003 or Windows 2000 and can be run against a remote host:

```
> pslist \\<ServerName>
```

There is also the *top.exe* command, which is available in the Windows 2000 Resource Kit. It provides a continually updated view of the top running process (by CPU):

```
> top
```



You can do something similar to *top* with *pslist* by specifying the *-s* option.

#### Using VBScript

```
' This code displays the running processes on the target computer.  
' ----- SCRIPT CONFIGURATION -----  
strComputer = "." ' Can be a hostname or "." to target local host  
' ----- END CONFIGURATION -----  
set objWMI = GetObject("winmgmts:\\." & strComputer & "\root\cimv2")  
set colProcesses = objWMI.InstancesOf("Win32_Process")
```

```
for each objProcess In colProcesses
    WScript.Echo objProcess.Name & " (" & objProcess.ProcessID & ")"
next
```

## Discussion

Sometimes it is difficult to associate an application (e.g., Internet Explorer) with its underlying process (e.g., *iexplore.exe*). In each of the command-line solutions, only the process name will be shown, which may be completely different from the name of the application. With Internet Explorer, it is pretty easy to figure out that *iexplore.exe* is probably the underlying process, but how can you tell for sure? One way is to look at Sysinternals Process Explorer. It displays a Description field that generally contains the application name of the process. Alternatively, you can specify the `/v` option with the *tasklist* command, which displays the Window Title field for each process. This typically includes the name of the application. Here is an example command you can run:

```
> tasklist /v /fo list
```

Unfortunately, there you can't retrieve the Window Title using the `Win32_Process` class.

## 6.5 Searching Processes

### Problem

You want to find processes that match certain criteria. This is useful if you want to find processes that have a certain process name or that are utilizing a certain amount of memory.

### Solution

#### Using a graphical user interface

1. Open the Sysinternals Process Explorer tool (*procexp.exe*).
2. From the menu, select **Find** → **Find Handle**.
3. Type the name of a process or handle to match (substring searches are allowed) and click the **Search** button.

#### Using a command-line interface

The Windows Server 2003 *tasklist.exe* command is very flexible. It provides several options for searching processes. This command searches for all *iexplore* (Internet Explorer) processes being run by the Administrator user:

```
> tasklist /FI "IMAGENAME eq iexplore*" /FI "USERNAME eq Administrator"
```

You can also use *tasklist.exe* to perform searches based on PID, memory usage, CPU time, and other attributes. The following command finds all processes running on host *dhcp01* that are consuming more than 10 MB of memory:

```
> tasklist /S dhcp01 /FI "MEMUSAGE gt 10240"
```

On Windows 2000, you can use the *tlist.exe* (or *pstlist.exe*) command in combination with *findstr.exe* to find processes. This returns all CMD processes:

```
> tlist | findstr cmd.exe
```

## Using VBScript

```
' This code finds the processes that have a memory usage greater
' than the specified amount. To search on different criteria,
' modify the WQL used in the ExecQuery call.
' ----- SCRIPT CONFIGURATION -----
strComputer = "."
intMaxMemKB = 1024 * 10000
' ----- END CONFIGURATION -----
set objWMI = GetObject("winmgmts:\\." & strComputer & "\root\cimv2")
set colProcesses = objWMI.ExecQuery("Select * from Win32_Process " & _
                                     " Where workingsetsize > " & intMaxMemKB )
WScript.Echo "Process, Size (in KB)"
for each objProcess in colProcesses
    WScript.Echo objProcess.Name & ", " & objProcess.WorkingSetSize / 1024
next
```

## Discussion

Sometimes it is necessary to take a snapshot of the current status of processes in a system and have a means of documenting that status. Using the command-line and scripting solutions provides a mechanism for generating a snapshot containing custom information.

## 6.6 Finding the Services Run from a Process

### Problem

You want to find the services being run from a process. In some cases, multiple services may be run from a single process.

### Solution

#### Using a graphical user interface

1. Open the Sysinternals Process Explorer tool (*procexp.exe*).
2. Double-click on the process you want to view.
3. If there are services being run from the process, a **Services** tab will be available from the process properties window.

## Using a command-line interface

The following command displays the services that are run from the *lsass.exe* process:

```
> tasklist /svc /FI "IMAGENAME eq lsass.exe"
```

You can also use the *tlist.exe* command from the Windows 2000 Support Tools to show similar information:

```
> tlist -s | findstr Svcs: | findstr lsass.exe
```

## Using VBScript

```
' This code displays the services run from the specified process.
' ----- SCRIPT CONFIGURATION -----
strComputer = "."
strProcess = "lsass.exe" ' name of process
' ----- END CONFIGURATION -----
set objWMI = GetObject("winmgmts: \\\" & strComputer & "\root\cimv2")
set colProcess = objWMI.ExecQuery("Select ProcessID from Win32_Process " & _
                                " Where Name = '" & strProcess & "'" )

for each objProcess in colProcess
    intPID = objProcess.ProcessID
next

WScript.Echo "Services run from process: " & strProcess
set colProcesses = objWMI.ExecQuery("Select Name from Win32_service " & _
                                    " Where ProcessID = " & intPID)

for each objProcess in colProcesses
    WScript.Echo " " & objProcess.Name
next
```

## Discussion

It is not uncommon for a single process to host multiple services. A good example of this is the *svchost.exe* process. Typically, you'll see several *svchost* processes running at any time on a system. That is because *svchost* is a generic process that is used by services that are run from dynamic link libraries (DLLs). If all of the code for a service is housed in a DLL, it still needs a process to accept and respond to SCM requests and handle other process management functions. *svchost* provides this functionality.



This recipe shows you how to find all of the services that a particular process is responsible for; to find the reverse of this (i.e., the process a particular service is associated with), refer to Recipe 7.9.

## See Also

MS KB 250320 (Description of Svchost.exe in Windows 2000) and MS KB 314056 (A description of Svchost.exe in Windows XP)

## 6.7 Viewing the Properties of a Process

### Problem

You want to view the properties of a process. This includes the process executable path, command line, current working directory, parent process (if any), owner, and startup timestamp.

### Solution

#### Using a graphical user interface

1. Open the Sysinternals Process Explorer tool (*procxp.exe*).
2. Double-click the process you want to view.
3. The **Image** tab contains process properties.

Some of this information can also be viewed using Windows Task Manager (*taskmgr.exe*). After starting *taskmgr.exe*, click on the **Processes** tab. Select **View** → **Select Columns** from the menu, and check the boxes beside the properties you want to see.

#### Using a command-line interface

The *tasklist.exe* command can display a subset of the properties described in the Problem section. Here is an example that displays properties for a specific process:

```
> tasklist /v /FI "IMAGENAME eq <ProcessName>" /FO list
```

#### Using VBScript

```
' This code displays the properties of a process.
' ----- SCRIPT CONFIGURATION -----
intPID = 3280 ' PID of the target process
strComputer = "."
' ----- END CONFIGURATION -----
WScript.Echo "Process PID: " & intPID
set objWMIProcess = GetObject("winmgmts:\\\" & strComputer & _
    "\root\cimv2:Win32_Process.Handle='" & intPID & "'")
WScript.Echo "Name: " & objWMIProcess.Name
WScript.Echo "Command line: " & objWMIProcess.CommandLine
WScript.Echo "Startup date: " & objWMIProcess.CreationDate
WScript.Echo "Description: " & objWMIProcess.Description
WScript.Echo "Exe Path: " & objWMIProcess.ExecutablePath
WScript.Echo "Parent Process ID: " & objWMIProcess.ParentProcessId
objWMIProcess.GetOwner strUser,strDomain
WScript.Echo "Owner: " & strDomain & "\" & strUser
```

## Discussion

Another option from the command line is to use *wmic* to harness the power of WMI. You can retrieve all of the properties defined by the `Win32_Process` class (see Table 6-3) by running this simple command:

```
> wmic process list full
```

You can also limit the output to a single process. The following example retrieves the properties for the *snmp.exe* process:

```
> wmic process where name="snmp.exe" get /format:list
```

## See Also

Recipe 6.1 for a list of `Win32_Process` properties

# 6.8 Viewing the Performance Statistics of a Process

## Problem

You want to view the memory, I/O, and CPU statistics of a process. This is useful if you want to examine the resources a process is using. If you find that you are running low on memory on a particular system, it can often be attributed to a single process that has consumed a large amount. If you can terminate that process, the system should go back to a stable state.

## Solution

### Using a graphical user interface

1. Open the Sysinternals Process Explorer tool (*procexp.exe*).
2. Double-click the process you want to view.
3. The **Performance** tab contains the process properties.

This information can also be viewed using Windows Task Manager (*taskmgr.exe*). After starting *taskmgr.exe*, click on the **Processes** tab. Select **View** → **Select Columns** from the menu and check the boxes beside the properties you want to see. And for yet another way to trend out process performance metrics (using more granular metrics), open Performance Monitor and look at the Process object.

### Using a command-line interface

The following command displays all of the performance metrics for a process:

```
> pslist -x <ProcessName>
```

Replace *<ProcessName>* with the name of the process without its extension. For example:

```
> pslist -x iexplore
```

## Using VBScript

```
' This code displays the performance stats of a process.
' ----- SCRIPT CONFIGURATION -----
intPID = 3280 ' PID of target process
strComputer = "."
' ----- END CONFIGURATION -----
WScript.Echo "Process PID: " & intPID
set objWMIProcess = GetObject("winmgmts:\\." & strComputer & _
                             "\root\cimv2:Win32_Process.Handle='" & intPID & "'")
arrProps = Array("Name", "KernelModeTime", "UserModeTime", _
                 "MaximumWorkingSetSize", "MinimumWorkingSetSize", _
                 "PageFaults", "PageFileUsage", "VirtualSize", _
                 "WorkingSetSize", "PeakPageFileUsage", "PeakVirtualSize", _
                 "PeakWorkingSetSize", "PrivatePageCount", _
                 "QuotaNonPagedPoolUsage", "QuotaPagedPoolUsage", _
                 "QuotaPeakNonPagedPoolUsage", "QuotaPeakPagedPoolUsage", _
                 "ThreadCount")
for each strProp in arrProps
    WScript.Echo strProp & ": " & objWMIProcess.Properties_(strProp)
next
```

## Discussion

If you need to get serious about analyzing performance statistics for one or more processes, you should consider using Performance Monitor (*perfmon.exe*). With the Process performance object (click the little + icon in the **System Monitor** and select Process under **Performance object**), you can graph a variety of metrics for individual processes or all of them together using the `_Total` instance.

Even if you don't want to use Performance Monitor to monitor processes, the tool provides some good information about process metrics, such as Working Set. Click the **Explain** button when you view the **Process** performance object, which will cause another dialog to appear that contains additional information about what each counter means. These counters are mostly the same ones as you'll find in Task Manager, *pslist*, and `Win32_Process`.

## 6.9 Viewing the DLLs Being Used by a Process

### Problem

You want to view the DLLs being used by a process or find the processes using a specific DLL. This can come in handy if you need to update a DLL and want to find out which programs are actively using it, or if you are trying to delete a DLL, but cannot due to a lock on the file by a process that is using it.

## Solution

### Using a graphical user interface

To view the DLLs being used by a process, do the following:

1. Open the Sysinternals Process Explorer tool (*procexp.exe*).
2. From the menu, select **View** → **Lower Pane View** → **DLLs**.
3. Click on the process you want to view. In the bottom window, the list of DLLs being used by that process is displayed.

To view the processes using a specific DLL, do the following:

1. Open the Sysinternals Process Explorer tool (*procexp.exe*).
2. From the menu, select **Find** → **Find DLLs**.
3. Type the name of the DLL (partial string accepted) and click the **Search** button.

### Using a command-line interface

To view the DLLs being used by a process, use the following command:

```
> listdlls <ProcessName>
```

To view the processes using a specific DLL, use the following command:

```
> listdlls -d <DLLName>
```

### Using VBScript

There are no scripting interfaces available to get this information. To get it programmatically, you must use the Win32 API or .NET Framework, or shell out to the *listdlls* utility.

## Discussion

Ever visited *DLL hell*? Things aren't as bad as they once were in the early days of Windows NT, but keeping track of DLL versions for certain applications can still be a pain. DLL hell was the term given to the problem where applications would overwrite DLLs with older or incompatible versions. This would cause applications to fail in unexpected ways. Starting with Windows 2000, this problem was reduced with the introduction of the Windows File Protection (WFP). Now, applications can't replace system DLLs—only system updates can, such as when you install a service pack or hotfix.

If you really want to dig down into a process and see what it is doing, check out Recipe 6.10 where I talk about viewing the APIs that a process calls. I'll also show you another way to view the DLLs loaded by a process.

## See Also

MS KB 222193 (Description of the Windows File Protection Feature) and MS KB 247957 (SAMPLE: Using DUPPS.exe to Resolve DLL Compatibility Problems)

## 6.10 Viewing the APIs Called by a Process

### Problem

You want to view the application programming interfaces (APIs) that a particular process invokes while running. This is useful for application developers who need to troubleshoot programs, but it can also be useful to system administrators who want to dig into the internals of a troublesome application.

### Solution

#### Using a graphical user interface

1. Open the API Monitoring tool (*apimon.exe*) from the Windows 2000 Resource Kit.
2. From the menu, select **File** → **Open** (or click the open folder icon).
3. Browse to the executable you want to monitor and click **OK**.
4. From the menu, select **Tools** → **Start Monitor** (or click the play button). This launches the selected application. Within the API Monitor screen there will be a **DLLs In Use** window and **API Counters** window. The API information will continue to be collected until you stop it.
5. From the menu, select **Tools** → **Stop Monitor** (or click the stop button).

### Discussion

If you have any Unix experience, you may be familiar with the *truss* utility. With *truss*, you can view all the system calls and APIs that a particular process is calling. Having this capability can be extremely beneficial when you are trying to debug why a particular application is failing or not behaving correctly. API Monitor provides similar functionality on the Windows platform. It displays the DLLs loaded by the target process, each function it calls, the number of times it calls a function, and the amount of time it took for each function call to complete.

If the API Monitor is the kind of thing that gets you excited, then you'll want to take a look at the Dependency Walker (*depends.exe*) in the Support Tools, which provides even more information.

## 6.11 Viewing the Handles a Process Has Open

### Problem

You want to view all the handles a process has open. This is handy if you want to find out all of the files and registry keys a particular process is using.

### Solution

#### Using a graphical user interface

1. Open the Sysinternals Process Explorer tool (*procxp.exe*).
2. From the menu, select **View** → **View Handles**.
3. Click on the process you want to view. In the bottom window, the list of handles being used by that process will be displayed.

#### Using a command-line interface

To view all of the handles a process has open, use the following command:

```
> handle -a -p <ProcessName>
```

You can also search for a specific handle using the following command:

```
> handle <HandleName>
```

For example, if you want to find all processes that have the *c:\test* directory open, you would replace *<HandleName>* with *c:\test*.

#### Using VBScript

There are no scripting interfaces to get this information. To get it programmatically, you must use the Win32 API or .NET Framework, or shell out to the *handle* utility.

### Discussion

Have you ever wanted to see all of the resources a particular process is using? Perhaps you have a new application or service and you want to see what files it touches, what registry keys it has open, what Windows stations it uses, etc. The lovely Process Explorer (and *handle.exe* command-line equivalent) can give you this and much more. The cool thing about Process Explorer is that it even lets you close a particular handle if you want to; from the bottom window simply right-click on the handle and select **Close Handle**. This is helpful if you are trying to delete a file or Registry key, but a process has a lock on it.

## 6.12 Viewing the Network Ports a Process Has Open

### Problem

You want to view the network ports on which a process is communicating. This is useful if you want to see the type of traffic a particular process is generating.

### Solution

#### Using a graphical user interface

1. Open the Sysinternals TCPView tool (*tcpview.exe*).
2. The complete list of processes and associated ports are displayed by default. New connections show up in green and terminating connections show up in red.

#### Using a command-line interface

The following command displays the open ports and the process ID of the process associated with the port. The `-o` option is new to *netstat.exe* in Windows XP and Windows Server 2003:

```
> netstat -o
```

The Sysinternals *netstatp.exe* command is similar to *netstat.exe*, except it displays the process name associated with each port:

```
> netstatp
```

And for yet another extremely useful port querying tool, check out *portqry.exe* (see MS KB 310099 for more information). With *portqry* you can get even more information than *netstatp*. Run this command to output all of the ports and their associated processes:

```
> portqry -local
```

That command also breaks port usage down by service (e.g., DnsCache). You can watch the port usage for a particular PID and log it to a file. The following command does this for PID 1234:

```
> portqry -wpid 1234 -wt 5 -l portoutput.txt -v
```

The `-wt` defines the watch time, which is how long *portqry* waits before examining the process again (the default is 60 seconds). The `-v` option is for verbose output.

#### Using VBScript

None of the scripting interfaces provide a way to access information about the ports a process has open. However, the *netstatp* tool comes with complete source that

shows how to do it via IP Helper functions that can be directly accessed with a non-scripting language.

## Discussion

Each connection to and from a client computer is associated with a process. This connection is also associated with a particular port. Most of the open ports you'll see will be numbered above 1024. This is because well-known ports use port numbers lower than 1024 so Windows dynamically allocates ports above that.

A connection has a state, which you'll see when running any of the utilities described in the solution. This state indicates the type of activity going on over the connection. Most of the connections you'll see are in the ESTABLISHED state, which simply means the connection is open and prepared to send or receive traffic. For a list of all the states, see Table 6-4.

Table 6-4. TCP connection states

State	Description
SYN_SEND	Indicates an active open of a connection
SYN_RECEIVED	Server received SYN from the client
ESTABLISHED	Client received server's SYN and session is established
LISTEN	Server is ready to accept a connection
FIN_WAIT_1	Indicates an active close of the connection
TIMED_WAIT	Client enters this state after an active close
CLOSE_WAIT	Indicates a passive close of the connection. Server just received first FIN from a client
FIN_WAIT_2	Client just received acknowledgment of its first FIN from the server
LAST_ACK	Server has sent its own FIN
CLOSED	Server received acknowledgment from client and closed the connection

## See Also

MS KB 137984 (TCP Connection States and Netstat Output) and MS KB 310099 (Description of the Portqry.exe Command-Line Utility)

## 6.13 Script: Process Doctor

Have you ever wanted to know when a particular process terminates on a system? Perhaps an application is failing mysteriously and you find out about it only after a user complains. Or maybe you have an application that fails periodically and you want to start it up immediately after it fails. This is pretty easy to accomplish using both VBScript and the command line.

## Using VBScript

With WMI and event handlers, process monitoring is straightforward. The following code monitors the *calc.exe* process, and as soon as it recognizes that it is no longer running, restarts it:

```
' ----- SCRIPT CONFIGURATION -----
strProcess = "calc.exe" ' Image name of the process you want to monitor
strComputer = "."
' ----- END CONFIGURATION -----
set objWMI = GetObject("winmgmts:\\\" & strComputer & "\root\cimv2")
set colProcesses = objWMI.ExecNotificationQuery(_
    "select * from __instanceDeletionEvent " _
    & " within 2 where TargetInstance isa 'Win32_Process' " _
    & " and TargetInstance.Name = '" & strProcess & "'"")
do
    set objProcess = colProcesses.NextEvent
    WScript.Echo "Process " & strProcess & _
        " (" & objProcess.TargetInstance.ProcessID & ") terminated"
    intRC = objWMI.Get("Win32_Process").Create(strProcess, , , intProcessID)
    if intRC = 0 Then
        Wscript.Echo strProcess & " started. PID: " & intProcessID
    else
        Wscript.Echo strProcess & " did not start. Error code: " & intRC
    end if
loop
```

The main method to note in this script is `ExecNotificationQuery`, which executes a WQL query and receives events that result from it. Let's break the query down. The `select` statement pulls all `__instanceDeletionEvent` objects:

```
"select * from __instanceDeletionEvent "
```

Anytime a process terminates, or a file is deleted, or a service is removed, that event is registered as a deletion event in WMI (`__instanceDeletionEvent`).\* This is incredibly powerful. The `select` statement I used matches ALL deletion events, which isn't quite what we want. This is why the next statement is needed:

```
" within 2 where TargetInstance isa 'Win32_Process' "
```

`TargetInstance isa 'Win32_Process'` filters out all deletion events except those of type `Win32_Process`. The `within 2` statement informs WMI how often to check the system (in seconds) to see if anything new matches this query.

At this point, the query would match all terminating processes, so I restrict it to just the one we are interested in:

```
" and TargetInstance.Name = '" & strProcess & "'"")
```

\* Likewise, new instances of processes, files, services, etc., are captured as `__instanceCreationEvent` objects and anytime an instance is modified, it is registered as an `__instanceModificationEvent`.

In the case of Win32\_Process instanceDeletionEvent objects, the TargetInstance.Name property equals the name of the process. So here I set that equal to the strProcess variable, which contains *calc.exe*.

After I've called ExecNotificationQuery, I am ready to enter a loop to start retrieving events. I used a do loop, which makes this script run until you kill it. The first line in the do loop invokes the NextEvent method. This is a synchronous call that waits until an event occurs before the script continues. If the *calc.exe* process never dies, the script never moves beyond this step. If it does catch the process terminating, it prints out that process's PID and creates a new instance of the process.

## Using a Command-Line Interface

You can do the exact same thing as the VBScript solution using a command line. Here is the complete command:

```
> for /L %v in (1,1,40) do (tasklist /FI "IMAGENAME eq calc.exe" /FO CSV /NH |  
findstr "calc.exe" || calc.exe) & sleep 15
```

Let's break this long command line down. The first part is a for loop that starts at 1 and goes to 40 by 1 (i.e., 1, 2, 3, 4 . . . 40). The loop counter is placed in the %v variable.

```
for /L %v in (1,1,40)
```

Next is the command to execute within the loop. The first part of the command simply runs the *tasklist.exe* utility and performs a search for the *calc.exe* process. The last two options tell *tasklist* to print the results (/FO) in a comma-separated list (CSV) and not to print any headers (/NH).

```
do (tasklist /FI "IMAGENAME eq calc.exe" /FO CSV /NH
```

If *calc.exe* is running, this command would produce this output:

```
"calc.exe", "4156", "Console", "0", "2,804 K"
```

Whereas if it is not running, it would produce this:

```
INFO: No tasks running with the specified criteria.
```

The second part of the command pipes the output from *tasklist* to the *findstr.exe* utility:

```
| findstr "calc.exe"
```

I do this because we need a way to determine if *calc.exe* is running. All we really want is for *tasklist* to produce some output if it is running and not to generate anything if it isn't. So this *findstr* command helps out by removing the INFO: No tasks . . . output if *calc.exe* isn't running.

Now, if the *tasklist* and *findstr* commands do not generate any output, the next command executes:

```
|| calc.exe)
```

If they generate output, this command won't execute, which is exactly what we want. It will run *calc.exe* only if *calc.exe* isn't running.

After we've done all this work to determine if *calc.exe* is running and to start it otherwise, we need to take a breather. Remember, this whole thing is in a for loop that will run 40 times. If we didn't put some sort of break in it, it would run 40 consecutive times without a pause. This is where the next command comes into play:

```
& sleep 15
```

This pauses the whole command for 15 seconds. And since we are using a loop with 40 iterations, the whole command will run over a 10-minute period ( $15 \times 40 / 60 = 10$  minutes). If you want this monitor to run for longer, simply change the number of seconds after the *sleep* command or the number of iterations in the for loop.

## 6.14 Script: Process Terminator

Have you ever wanted to prevent a process from running? Perhaps the process keeps starting and you haven't been able to find what is starting it. Or maybe you know what is causing the process to start, but you can't prevent it from happening. I called the script the Process Doctor because it tries to revive processes that die. In this recipe, I'll describe the opposite. The Process Terminator kills a certain process every time it tries to run.

### Using VBScript

The code in this script is very similar to that of the Process Doctor. The primary difference is that instead of looking at *instanceDeletionEvent* objects, we are looking for *instanceCreationEvent* objects, that is, new instances of the *calc.exe* process. Here is the script:

```
' ----- SCRIPT CONFIGURATION -----
strProcess = "calc.exe"
strComputer = "."
' ----- END CONFIGURATION -----

set objWMI = GetObject("winmgmts:\\." & strComputer & "\root\cimv2")
set colProcesses = objWMI.ExecNotificationQuery(_
    "select * from __instanceCreationEvent " _
    & " within 1 where TargetInstance isa 'Win32_Process' " _
    & " and TargetInstance.Name = '" & strProcess & "'")
do
    set objProcess = colProcesses.NextEvent
    WScript.Echo "Terminating process " & strProcess & _
        " (" & objProcess.TargetInstance.ProcessID & ")"
    objProcess.TargetInstance.Terminate
loop
```

Also, instead of creating a new instance of the process when a creation event is found, I terminate the process. For more on terminating process, see Recipe 6.3.

## Using a Command-Line Interface

Just as with the Process Doctor, you can perform similar functionality to the Process Terminator using a command line. But fortunately, the command line in this case isn't quite as complicated. Here is the command-line version of the Process Terminator using the *taskkill.exe* command:

```
> for /L %v in (1,1,10) do taskkill /IM calc.exe /F /T & sleep 60
```

The first part is very similar to the Process Doctor command line. This is a for loop that iterates from 1 to 10 by 1's.

```
for /L %v in (1,1,10)
```

The *taskkill* command matches any process with an image name of *calc.exe* and forcefully terminates its process tree (/F and /T):

```
do taskkill /IM calc.exe /F /T
```

Lastly, we sleep for 60 seconds and start the next iteration of the loop:

```
& sleep 60
```

You can also do the same thing using the Sysinternals *pskill.exe* command:

```
> for /L %v in (1,1,10) do pskill calc.exe & sleep 60
```