# Unit OS4: Scheduling and Dispatch

## 4.6. Lab Manual

Windows Operating System Internals - by David A. Solomon and Mark E. Russinovich with Andreas Polze

## Copyright Notice

## Roadmap for Section 4.6.

- Monitoring Processes with TaskManager
- Process Explorer and Thread Monitoring
- PsTools for gathering process information
- Kernel debugger !process and !thread
- Watching the scheduler: CPU boosts
- Monitoring starvation avoidance

3

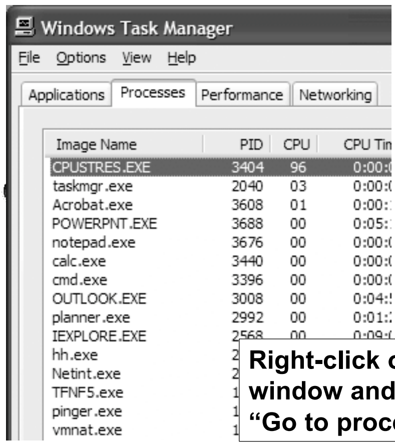
This LabManual includes experiments investigating the algorithms for scheduling and process/thread control implemented inside the Windows operating system. Students are expected to carry out Labs in addition to studying the learning materials in Unit OS4.

A thorough understanding of the concepts presented in Unit OS4: Scheduling and Displatch is a prerequisite for these Labs.

These concepts are explained in Chapter 6 of Windows Internals 4th edition.

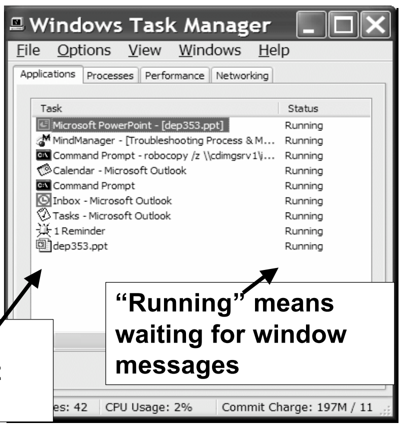
## Task Manager: Processes vs Applications Tabs

- Processes tab: List of processes
- Applications tab: List of top level visible windows



Windows Task Manager

| Image Name | PID | CPU | CPU Tin |
|---|---|---|---|
| CPUSTRES.EXE | 3404 | 96 | 0:00:0 |
| taskmgr.exe | 2040 | 03 | 0:00:0 |
| Acrobat.exe | 3608 | 01 | 0:00:1 |
| POWERPNT.EXE | 3688 | 00 | 0:05:1 |
| notepad.exe | 3676 | 00 | 0:00:0 |
| calc.exe | 3440 | 00 | 0:00:0 |
| cmd.exe | 3396 | 00 | 0:00:0 |
| OUTLOOK.EXE | 3008 | 00 | 0:04:5 |
| planner.exe | 2992 | 00 | 0:01:2 |
| IEXPLORE.EXE | 2568 | 00 | 0:09:0 |
| hh.exe | 2 | | |
| Netint.exe | 2 | | |
| TFNF5.exe | 1 | | |
| pinger.exe | 1 | | |
| vmnat.exe | 1 | | |



Windows Task Manager

| Task | Status |
|---|---|
| Microsoft PowerPoint - [dep353.ppt] | Running |
| MindManager - [Troubleshooting Process & M... | Running |
| Command Prompt - robocopy /z \\cdimgsrv1\j... | Running |
| Calendar - Microsoft Outlook | Running |
| Command Prompt | Running |
| Inbox - Microsoft Outlook | Running |
| Tasks - Microsoft Outlook | Running |
| 1 Reminder | Running |
| dep353.ppt | Running |

es: 42 CPU Usage: 2% Commit Charge: 197M / 11

**Right-click on a window and select "Go to process"**

**"Running" means waiting for window messages**

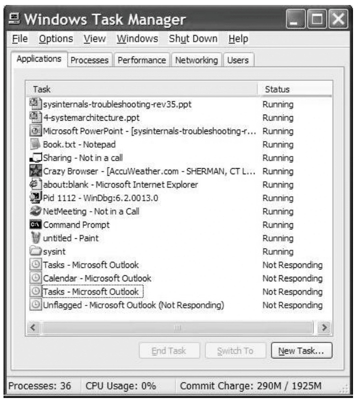
4

Lab objective: Understand Task Manager's Processes and Applications Tabs

The built-in Windows Task Manager provides a quick list of the processes running on the system. You can start Task Manager in one of three ways: (1) press Ctrl+Shift+Esc, (2) right-click on the taskbar and select Task Manager, or (3) press Ctrl+Alt+Delete and click the Task Manager button. Once Task Manager has started, click the Processes tab to see the list of running processes. Notice that processes are identified by the name of the image of which they are an instance. Unlike some objects in Windows, processes can't be given global names. To display additional details, choose Select Columns from the View menu and select additional columns to be added.

## Understand Task Managers “Applications”

- A meaningless term at the OS level
  - Not a list of processes
  - Not a list of “tasks” (another meaningless term)
  - It’s a list of top level visible windows in your session that meet certain criteria
- What does the status column mean?
  - Running:
    - Windows don’t run—threads do
    - Running displayed only when owning thread is waiting for a window message (e.g. not running!)
  - Not Responding: not waiting for window messages
- To map a window to a process, right-click on a window and select “Go to process”



Windows Task Manager
File Options View Windows Shut Down Help
Applications Processes Performance Networking Users
End Task Switch To New Task...

| Task | Status |
|---|---|
| sysinternals-troubleshooting-rev35.ppt | Running |
| 4-systemarchitecture.ppt | Running |
| Microsoft PowerPoint - [sysinternals-troubleshooting-r... | Running |
| Book.txt - Notepad | Running |
| Sharing - Not in a call | Running |
| Crazy Browser - [AccuWeather.com - SHERMAN, CT L... | Running |
| about:blank - Microsoft Internet Explorer | Running |
| Pid 1112 - WinDbg:6.2.0013.0 | Running |
| NetMeeting - Not in a Call | Running |
| Command Prompt | Running |
| untitled - Paint | Running |
| sysint | Running |
| Tasks - Microsoft Outlook | Not Responding |
| Calendar - Microsoft Outlook | Not Responding |
| Tasks - Microsoft Outlook | Not Responding |
| Unflagged - Microsoft Outlook (Not Responding) | Not Responding |

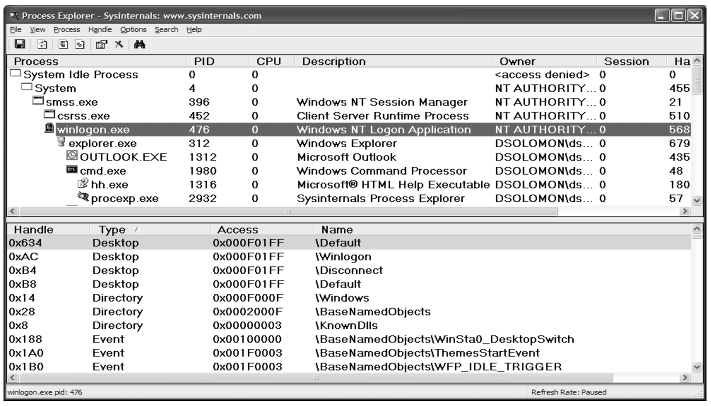
Processes: 36 CPU Usage: 0% Commit Charge: 290M / 1925M

5

Lab objective: Understand Task Managers “Applications”
Although what you see in the Task Manager Processes tab is clearly a list of processes, what the Applications tab displays isn’t as obvious. The Applications tab lists the top-level visible windows on all the desktops in the interactive window station. (By default, there are two desktop objects—you can create more by using the Windows *CreateDesktop* function.) The Status column indicates whether or not the thread that owns the window is in a Windows message wait state. “Running” means the thread is waiting for windowing input; “Not Responding” means the thread isn’t waiting for windowing input (for example, the thread might be running or waiting for I/O or some Windows synchronization object). From the Applications tab, you can match a task to the process that owns the thread that owns the task window by right-clicking on the task name and choosing Go To Process.

## Process Explorer (Sysinternals)

- “Super Task Manager”
  - Shows full image path, command line, environment variables, parent process, security access token, open handles, loaded DLLs & mapped files


Process Explorer - Sysinternals: www.sysinternals.com
File View Process Handle Options Search Help

| Process | PID | CPU | Description | Owner | Session | Ha |
|---|---|---|---|---|---|---|
| System Idle Process | 0 | 0 | | <access denied> | 0 | 0 |
| System | 4 | 0 | | NT AUTHORITY... | 0 | 455 |
| smss.exe | 396 | 0 | Windows NT Session Manager | NT AUTHORITY... | 0 | 21 |
| csrss.exe | 452 | 0 | Client Server Runtime Process | NT AUTHORITY... | 0 | 510 |
| winlogon.exe | 476 | 0 | Windows NT Logon Application | NT AUTHORITY... | 0 | 568 |
| explorer.exe | 312 | 0 | Windows Explorer | DSOLOMON\ds... | 0 | 679 |
| OUTLOOK.EXE | 1312 | 0 | Microsoft Outlook | DSOLOMON\ds... | 0 | 435 |
| cmd.exe | 1980 | 0 | Windows Command Processor | DSOLOMON\ds... | 0 | 48 |
| hh.exe | 1316 | 0 | Microsoft® HTML Help Executable | DSOLOMON\ds... | 0 | 180 |
| procexp.exe | 2932 | 0 | Sysinternals Process Explorer | DSOLOMON\ds... | 0 | 57 |

| Handle | Type | Access | Name |
|---|---|---|---|
| 0x634 | Desktop | 0x000F01FF | \Default |
| 0xAC | Desktop | 0x000F01FF | \Winlogon |
| 0xB4 | Desktop | 0x000F01FF | \Disconnect |
| 0xB8 | Desktop | 0x000F01FF | \Default |
| 0x14 | Directory | 0x000F000F | \Windows |
| 0x28 | Directory | 0x0002000F | \BaseNamedObjects |
| 0x8 | Directory | 0x00000003 | \KnownDlls |
| 0x188 | Event | 0x00100000 | \BaseNamedObjects\WinSta0_DesktopSwitch |
| 0x1A0 | Event | 0x001F0003 | \BaseNamedObjects\ThemesStartEvent |
| 0x1B0 | Event | 0x001F0003 | \BaseNamedObjects\WFP_IDLE_TRIGGER |

winlogon.exe pid: 476    Refresh Rate: Paused

6

Lab objective: Experiment with Process Explorer

Process Explorer, from *www.sysinternals.com*, shows more details about processes and threads than any other available tool, which is why you will see it used in a number of experiments throughout the book. The following are some of the unique things that Process Explorer shows or enables:

- Full path name for the image being executed
- Process security token (list of groups and privileges)
- Highlighting to show changes in the process and thread list
- List of services inside service-hosting processes, including display name and description
- Processes that are part of a job and job details
- Processes running .NET/WinFX applications and.NET-specific details (such as the list of appdomains and CLR performance counters)
- Start time for processes and threads
- Complete list of memory mapped files (not just DLLs)
- Ability to suspend a process
- Ability to kill an individual thread
- Easy identification of which processes were consuming the most CPU time over a period of time (The Performance Tool can display process CPU utilization for a given set of processes, but it won’t automatically show processes created after the performance monitoring session has started.)

## Process Explorer’s Process List

1. Run Process Explorer & maximize window
2. Run Task Manager – click on Processes tab
3. Arrange windows so you can see both
4. Notice process tree vs flat list in Task Manager
   - If parent has exited, process is left justified
5. Sort on first column (“Process”) and note tree view disappears
6. Click on View->Show Process Tree (or CTRL+T) to bring it back
7. Notice description and company name columns
8. Hover mouse over image to see full path of image
9. Right click on a process and choose “Google”

7

Lab objective: Investigate Process Explorer’s Process List

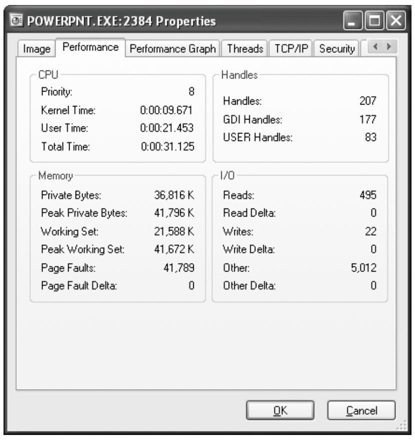
Process Explorer also provides easy access to information available through other tools from one central place, such as:

- Process tree (with ability to collapse parts of the tree)
- Open handles in a process without prior setup (The Microsoft tools to show open handles require the setting of a systemwide flag and a reboot before they can be used.)
- List of DLLs (and memory-mapped files) in a process
- Thread activity within a process
- User-mode thread stacks (including mapping of addresses to names using the debugging tools’ symbol engine)
- Kernel-mode thread stacks for system threads (including mapping of addresses to names using the debugging tools’ symbol engine)
- Context switch delta
  (a better representation of CPU activity, as explained in Chapter 6)
- Kernel memory (paged and nonpaged pool) limits (other tools show only current size)

The first time you run it, you will receive a message that symbols are not currently configured. If properly configured, Process Explorer can access symbol information to display the symbolic name of the thread start function and functions on its call stack (available by double-clicking on a process and clicking on the Threads tab). This is useful for identifying what threads are doing within a process. To access symbols, you must have the Debugging Tools installed (described later in this chapter). Then click on Options, choose Configure Symbols, and fill in the appropriate Symbols path. For more information on configuring use of the symbol server, see *http://www.microsoft.com/whdc/ddk/debugging/symbols.mspx.*

## Process Performance

- Click on Performance Tab of process properties
  - Note: all these numbers can be configured as columns



CPU

| | |
|---|---|
| Priority: | 8 |
| Kernel Time: | 0:00:09.671 |
| User Time: | 0:00:21.453 |
| Total Time: | 0:00:31.125 |

Handles

| | |
|---|---|
| Handles: | 207 |
| GDI Handles: | 177 |
| USER Handles: | 83 |

Memory

| | |
|---|---|
| Private Bytes: | 36,816 K |
| Peak Private Bytes: | 41,796 K |
| Working Set: | 21,588 K |
| Peak Working Set: | 41,672 K |
| Page Faults: | 41,789 |
| Page Fault Delta: | 0 |

I/O

| | |
|---|---|
| Reads: | 495 |
| Read Delta: | 0 |
| Writes: | 22 |
| Write Delta: | 0 |
| Other: | 5,012 |
| Other Delta: | 0 |

8

Lab objective: Use Process Explorer’s performance tab to investigate process behavior
Where is full path.What if process gets same pid as previous process? DO first process’ children get forcibly adopted?  No.

Do “notepad fred” to show command-line arguments. Shortcut might pass arguments.

Names used in performance tab much more logical than task manager.

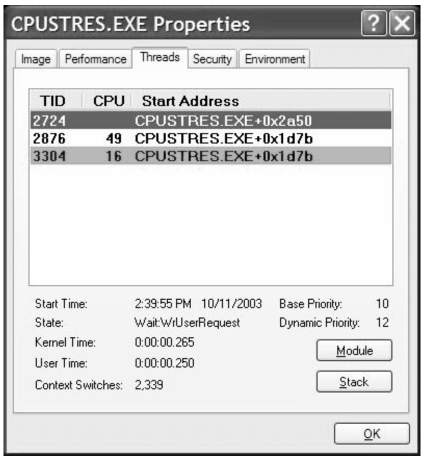
Security: useful for telling what groups the process belongs to. Definitive list.

Explain privileges. Example: change system time. Double click on time applet and look at it in process explorer. See rundll32.exe. If double-click on it and look at command-line, last argument is timedate.cpl. Look at security and find that the time privilege is on.

Environment tab: sometimes input to scripts or apps. PATH is one of the more important. Inherited from parent. Set DAVID=brilliant. Run paintbrush and look at it’s environment. But not in others.

## Thread Details

- Process Explorer "Threads" tab shows which thread(s) are running
  - Start address represents where the thread began running (not where it is now)
  - Click Module to get details on module containing thread start address



| TID | CPU | Start Address |
|---|---|---|
| 2724 | | CPUSTRES.EXE+0x2a50 |
| 2876 | 49 | CPUSTRES.EXE+0x1d7b |
| 3304 | 16 | CPUSTRES.EXE+0x1d7b |

Start Time: 2:39:55 PM 10/11/2003
State: Wait:WrUserRequest
Kernel Time: 0:00:00.265
User Time: 0:00:00.250
Context Switches: 2,339
Base Priority: 10
Dynamic Priority: 12

9

Lab objective: Investigate thread details
Process Explorer provides easy access to thread activity within a process. This is especially important if you are trying to determine why a process is running that is hosting multiple services (such as Svchost.exe, Dllhost.exe, Inetinfo.exe, or the System process) or why a process is hung.

To view the threads in a process, select a process and open the process properties (double-click on the process or click on the Process, Properties menu item). Then click on the Threads tab. This tab shows a list of the threads in the process. For each thread it shows the percentage of CPU consumed (based on the refresh interval configured), the number of context switches to the thread, and the thread start address. You can sort by any of these three columns.

New threads that are created are highlighted in green, and threads that exit are highlighted in red. (The highlight duration can be configured with the Options, Configure Highlighting menu item.) This might be helpful to discover unnecessary thread creation occurring in a process. (In general, threads should be created at process startup, not every time a request is processed inside a process.)

As you select each thread in the list, Process Explorer displays the thread ID, start time, state, CPU time counters, number of context switches, and the base and current priority. There is a Kill button, which will terminate an individual thread, but this should be used with extreme care.

The context switch delta represents the number of times that thread began running in between the refreshes configured for Process Explorer. It provides a different way to determine thread activity than using the percentage of CPU consumed. In some ways it is better because many threads run for such a short amount of time that they are seldom (if ever) the currently running thread when the interval clock timer interrupt occurs, and therefore, are not charged for their CPU time.

## Thread Start Functions

- Process Explorer can map the addresses within a module to the names of functions
  - This can help identify which component within a process is responsible for CPU usage
- Requires access to:
  - Symbol file for that module
  - Proper version of Dbghelp.dll
- By default, Process Explorer looks for:
  - Dbghelp.dll: in the default Windows Debugging Tools install directory
  - Symbols: _NT_SYMBOL_PATH environment variable
  - Can also specify with Options->Configure Symbols

Lab objective: Investigate Thread Start Functions

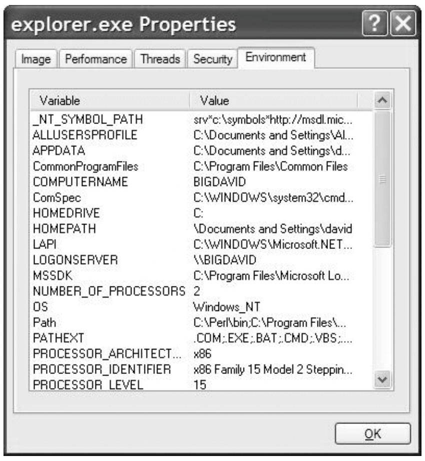
The thread start address is displayed in the form "*module*!*function*", where *module* is the name of the .exe or .dll. The function name relies on access to symbol files for the module. (See "Lab objective: Viewing Process Details with Process Explorer" in Chapter 1.) If you are unsure what the module is, press the Module button. This opens an Explorer file properties window for the module containing the thread's start address (for example, the .exe or .dll).

**Note_**For threads created by the Windows CreateThread function, Process Explorer displays the function passed to *CreateThread*, not the actual thread start function. That is because all Windows threads start at a common process or thread startup wrapper function (*BaseProcessStart* or *BaseThreadStart* in Kernel32.dll). If Process Explorer showed the actual start address, most threads in processes would appear to have started at the same address, which would not be helpful in trying to understand what code the thread was executing.

If properly configured, Process Explorer can access symbol information to display the symbolic name of the thread start function and functions on its call stack (available by double-clicking on a process and clicking on the Threads tab). This is useful for identifying what threads are doing within a process. To access symbols, you must have the Debugging Tools installed (described later in this chapter). Then click on Options, choose Configure Symbols, and fill in the appropriate Symbols path. For more information on configuring use of the symbol server, see *http://www.microsoft.com/whdc/ddk/debugging/symbols.mspx.*

## Process Explorer Lab: Environment Variables

- Click on Environment Tab of process properties


explorer.exe Properties

| Variable | Value |
|---|---|
| _NT_SYMBOL_PATH | srv*c:\symbols*http://msdl.mic... |
| ALLUSERSPROFILE | C:\Documents and Settings\Al... |
| APPDATA | C:\Documents and Settings\d... |
| CommonProgramFiles | C:\Program Files\Common Files |
| COMPUTERNAME | BIGDAVID |
| ComSpec | C:\WINDOWS\system32\cmd... |
| HOMEDRIVE | C: |
| HOMEPATH | \Documents and Settings\david |
| LAPI | C:\WINDOWS\Microsoft.NET... |
| LOGONSERVER | \\BIGDAVID |
| MSSDK | C:\Program Files\Microsoft Lo... |
| NUMBER_OF_PROCESSORS | 2 |
| OS | Windows_NT |
| Path | C:\Perl\bin;C:\Program Files\... |
| PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;... |
| PROCESSOR_ARCHITECT... | x86 |
| PROCESSOR_IDENTIFIER | x86 Family 15 Model 2 Steppin... |
| PROCESSOR_LEVEL | 15 |

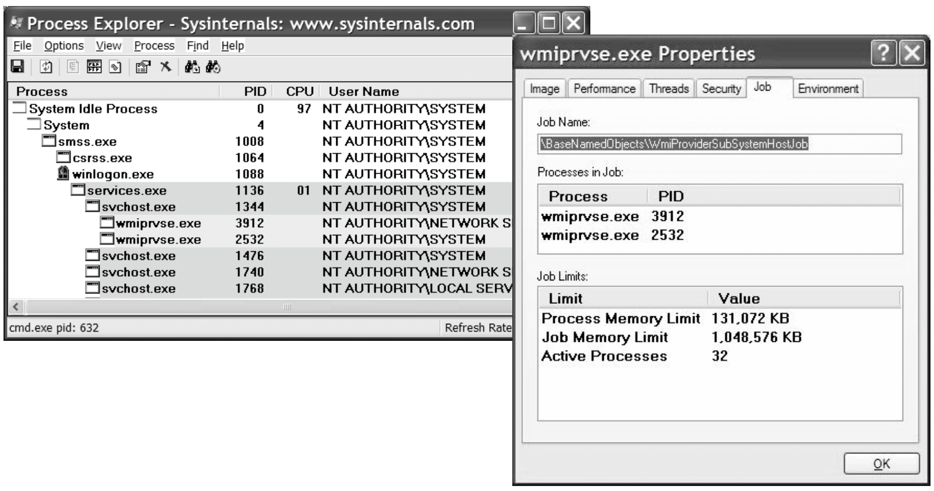
11

Lab objective: Monitor Environment Variables with Process Explorer

- Open a command prompt
- Run Notepadexe from command prompt
- Type “set  abc=xyz”
- In ProcExp, hit F5 and examine environment variables for Cmd.exe and Notepad.exe
  - Notice Notepad.exe does not know about the environment variable abc

## Identify Jobs used by WMI

- Jobs are used by WMI
  - Example: run Psinfo (Sysinternals) and pause output



Process Explorer - Sysinternals: www.sysinternals.com

| Process | PID | CPU | User Name |
|---|---|---|---|
| System Idle Process | 0 | 97 | NT AUTHORITY\SYSTEM |
| System | 4 | | NT AUTHORITY\SYSTEM |
| smss.exe | 1008 | | NT AUTHORITY\SYSTEM |
| csrss.exe | 1064 | | NT AUTHORITY\SYSTEM |
| winlogon.exe | 1088 | | NT AUTHORITY\SYSTEM |
| services.exe | 1136 | 01 | NT AUTHORITY\SYSTEM |
| svchost.exe | 1344 | | NT AUTHORITY\SYSTEM |
| wmiprvse.exe | 3912 | | NT AUTHORITY\NETWORK S |
| wmiprvse.exe | 2532 | | NT AUTHORITY\SYSTEM |
| svchost.exe | 1476 | | NT AUTHORITY\SYSTEM |
| svchost.exe | 1740 | | NT AUTHORITY\NETWORK S |
| svchost.exe | 1768 | | NT AUTHORITY\LOCAL SERV |

cmd.exe pid: 632 Refresh Rate

wmiprvse.exe Properties

Job Name: \BaseNamedObjects\WmiProviderSubSystemHostJob

Processes in Job:

| Process | PID |
|---|---|
| wmiprvse.exe | 3912 |
| wmiprvse.exe | 2532 |

Job Limits:

| Limit | Value |
|---|---|
| Process Memory Limit | 131,072 KB |
| Job Memory Limit | 1,048,576 KB |
| Active Processes | 32 |

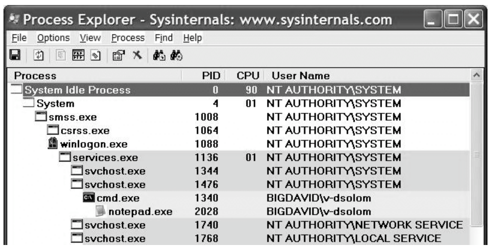
12

Lab objective: Identify Jobs used by WMI (Windows Management Instrumentation)

1. From a command prompt, run Psinfo (from www.sysinternals.com)
2. Notice in Process Explorer two WMI (Windows Management Instrumentation) provider processes that are part of a job object (highlighted above)
   (for a description of WMI, see Windows Internals, 4th edition p.237)
3. Double click on either Wmiprvse.exe process and click on the Job tab.
4. Notice the limits set for the job (per-process and job-wide private virtual memory and total active process count)

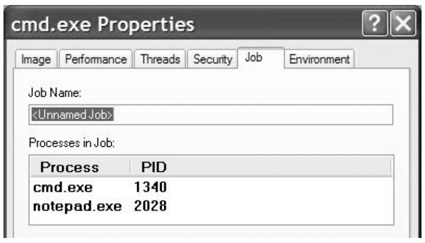
## Jobs created by RUNAS

1. In a command prompt:
   RUNAS /USER:xxx CMD
   (where xxx is some other local account)
2. In ProcExp, find newly created cmd.exe process
   - Who is the father?
3. Run Notepad from new CMD window
4. Double click on newly highlighted process & click on Job tab



```
Process Explorer - Sysinternals: www.sysinternals.com
File  Options  View  Process  Find  Help

Process                     PID   CPU  User Name
System Idle Process           0    90  NT AUTHORITY\SYSTEM
  System                      4    01  NT AUTHORITY\SYSTEM
    smss.exe               1008        NT AUTHORITY\SYSTEM
      csrss.exe            1064        NT AUTHORITY\SYSTEM
      winlogon.exe         1088        NT AUTHORITY\SYSTEM
        services.exe       1136    01  NT AUTHORITY\SYSTEM
          svchost.exe      1344        NT AUTHORITY\SYSTEM
          svchost.exe      1476        NT AUTHORITY\SYSTEM
            cmd.exe        1340        BIGDAVID\v-dsolom
              notepad.exe  2028        BIGDAVID\v-dsolom
          svchost.exe      1740        NT AUTHORITY\NETWORK SERVICE
          svchost.exe      1768        NT AUTHORITY\LOCAL SERVICE
```



```
cmd.exe Properties
Image | Performance | Threads | Security | Job | Environment

Job Name:
<Unnamed Job>

Processes in Job:
Process      PID
cmd.exe      1340
notepad.exe  2028
```

13

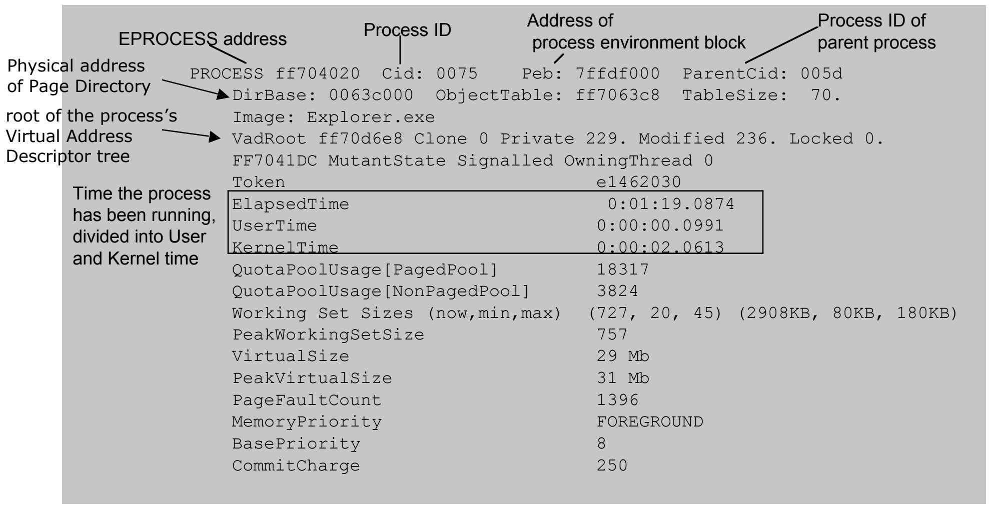
Lab objective: Investigate Jobs created by RUNAS

The RUNAS command permits launching processes under alternate credentials. The service behind the RUNAS command (called the Secondary Logon service) uses a job object to contain the process(es) it creates. This is so that at logoff, the service can terminate all processes that were created by RUNAS and any processes created by these processes, even if the parent/child relationships have been broken.

To view the job object created when RUNAS is used, perform the following steps:

1. From the command prompt, use the *runas* command to create a process running the command prompt (Cmd.exe). For example, type **runas /user:<domain>\<username> cmd**. You'll be prompted for your password. Enter your password, and a command prompt window will appear. The Windows service that executes runas commands creates an unnamed job to contain all processes (so that it can terminate these processes at logoff time).
2. From the command prompt, run Notepad.exe.
3. Then run Process Explorer and notice that the Cmd.exe and Notepad.exe processes are highlighted as part of a job.
4. Double-click either the Cmd.exe or Notepad.exe process to bring up the process properties. You will see a Job tab on the process properties dialog box.
5. Click the Job tab to view the details about the job. In this case, there are no quotas associated with the job, but there are two member processes.

## Process Block (!process)


EPROCESS address
Process ID
Address of process environment block
Process ID of parent process
Physical address of Page Directory
root of the process's Virtual Address Descriptor tree
Time the process has been running, divided into User and Kernel time

```
PROCESS ff704020  Cid: 0075    Peb: 7ffdf000  ParentCid: 005d
    DirBase: 0063c000  ObjectTable: ff7063c8  TableSize:  70.
    Image: Explorer.exe
    VadRoot ff70d6e8 Clone 0 Private 229. Modified 236. Locked 0.
    FF7041DC MutantState Signalled OwningThread 0
    Token                              e1462030
    ElapsedTime                         0:01:19.0874
    UserTime                           0:00:00.0991
    KernelTime                         0:00:02.0613
    QuotaPoolUsage[PagedPool]          18317
    QuotaPoolUsage[NonPagedPool]       3824
    Working Set Sizes (now,min,max)  (727, 20, 45) (2908KB, 80KB, 180KB)
    PeakWorkingSetSize                 757
    VirtualSize                        29 Mb
    PeakVirtualSize                    31 Mb
    PageFaultCount                     1396
    MemoryPriority                     FOREGROUND
    BasePriority                       8
    CommitCharge                       250
```

14

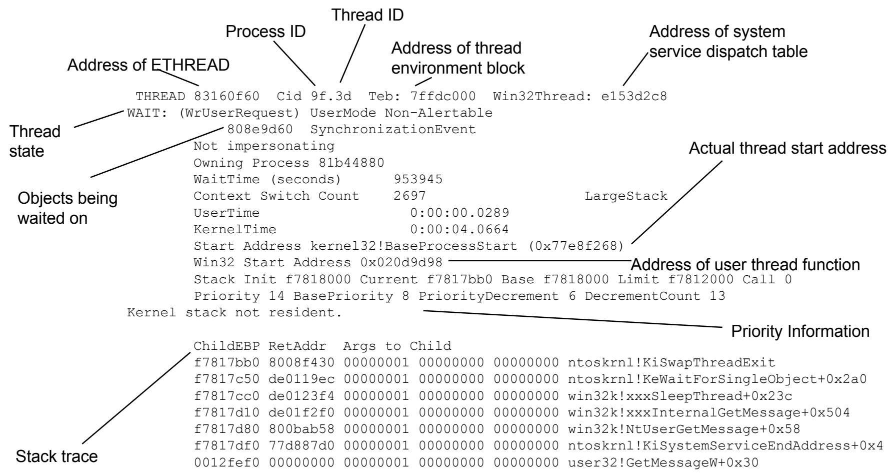
Lab objective: Using the Kernel Debugger !process Command

The kernel debugger !process command displays a subset of the information in an EPROCESS block. This output is arranged in two parts for each process. First you see the information about the process, as shown here (when you don't specify a process address or ID, !process lists information for the active process on the current CPU):

```
lkd> !process
PROCESS 8575f030 SessionId:0 Cid: 08d0 Peb:7ffdf000 ParentCid:0360
DirBase:1a81b000 ObjectTable:e12bd418 HandleCount: 65.
             Image:windbg.exe
             VadRoot 857f05e0Vads 71Clone0 Private 1152. Modified98.Locked1.
             DeviceMape1e96c88
             Token e1f5b8a8
             ElapsedTime 1:23:06.0219
             UserTime 0:00:11.0897
             KernelTime 0:00:07.0450
             QuotaPoolUsage[PagedPool] 38068
             QuotaPoolUsage[NonPagedPool] 2840
             Working Set Sizes (now,min,max) (2552, 50, 345)(10208KB,200KB, 1380KB)
             PeakWorkingSetSize 2715
             VirtualSize 41 Mb
             PeakVirtualSize 41 Mb
             PageFaultCount 3658
             MemoryPriority BACKGROUND
             BasePriority 8
             CommitCharge 1566
```

After the basic process output comes a list of the threads in the process. Other commands that display process information include !handle, which dumps the process handle table .

## Thread Block (!thread)


Thread ID
Process ID
Address of system service dispatch table
Address of thread environment block
Address of ETHREAD
Thread state
Objects being waited on
Actual thread start address
Address of user thread function
Priority Information
Stack trace

```
  THREAD 83160f60  Cid 9f.3d  Teb: 7ffdc000  Win32Thread: e153d2c8
 WAIT: (WrUserRequest) UserMode Non-Alertable
            808e9d60  SynchronizationEvent
        Not impersonating
        Owning Process 81b44880
        WaitTime (seconds)      953945
        Context Switch Count    2697                  LargeStack
        UserTime                  0:00:00.0289
        KernelTime                0:00:04.0664
        Start Address kernel32!BaseProcessStart (0x77e8f268)
        Win32 Start Address 0x020d9d98
        Stack Init f7818000 Current f7817bb0 Base f7818000 Limit f7812000 Call 0
        Priority 14 BasePriority 8 PriorityDecrement 6 DecrementCount 13
 Kernel stack not resident.

        ChildEBP RetAddr  Args to Child
        f7817bb0 8008f430 00000001 00000000 00000000 ntoskrnl!KiSwapThreadExit
        f7817c50 de0119ec 00000001 00000000 00000000 ntoskrnl!KeWaitForSingleObject+0x2a0
        f7817cc0 de0123f4 00000001 00000000 00000000 win32k!xxxSleepThread+0x23c
        f7817d10 de01f2f0 00000001 00000000 00000000 win32k!xxxInternalGetMessage+0x504
        f7817d80 800bab58 00000001 00000000 00000000 win32k!NtUserGetMessage+0x58
        f7817df0 77d887d0 00000001 00000000 00000000 ntoskrnl!KiSystemServiceEndAddress+0x4
        0012fef0 00000000 00000001 00000000 00000000 user32!GetMessageW+0x30
```

Lab objective: Using the Kernel Debugger !thread Command

The kernel debugger !thread command dumps a subset of the information in the thread data structures. Some key elements of the information the kernel debugger displays can't be displayed by any utility: internal structure addresses; priority details; stack information; the pending I/O request list; and, for threads in a wait state, the list of objects the thread is waiting for. To display thread information, use either the !process command (which displays all the thread blocks after displaying the process block) or the !thread command to dump a specific thread. The output of the thread information, along with some annotations of key fields, is shown above.

## Process Block Layout

```
lkd> dt nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER
   +0x078 ExitTime         : _LARGE_INTEGER
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : Ptr32 Void
   +0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x090 QuotaUsage       : [3] Uint4B
   +0x09c QuotaPeak        : [3] Uint4B
   +0x0a8 CommitCharge     : Uint4B
   +0x0ac PeakVirtualSize  : Uint4B
   +0x0b0 VirtualSize      : Uint4B
                           .
                           .
```

- **NOTE: Add “-r” to recurse through substructures**

16

Lab objective: Displaying the Format of an EPROCESS Block

For a list of the fields that make up an EPROCESS block and their offsets in hexadecimal, type dt _eprocess in the kernel debugger. The output (truncated for the sake of space) looks like this:

```
lkd> dt _eprocess
nt!_EPROCESS
            +0x000Pcb :_KPROCESS
            +0x06cProcessLock :_EX_PUSH_LOCK
            +0x070CreateTime :_LARGE_INTEGER
            +0x078ExitTime :_LARGE_INTEGER
            +0x080RundownProtect :_EX_RUNDOWN_REF
            +0x084UniqueProcessId :Ptr32Void
            +0x088ActiveProcessLinks:_LIST_ENTRY
            +0x090QuotaUsage :[3] Uint4B
            +0x09cQuotaPeak :[3] Uint4B
            +0x0a8CommitCharge :Uint4B
            +0x0acPeakVirtualSize :Uint4B
            +0x0b0VirtualSize :Uint4B
            +0x0b4SessionProcessLinks :_LIST_ENTRY
            +0x0bcDebugPort :Ptr32Void
            +0x0c0ExceptionPort :Ptr32Void
            +0x0c4ObjectTable :Ptr32_HANDLE_TABLE
            +0x0c8Token : _EX_FAST_REF
            +0x0ccWorkingSetLock : _FAST_MUTEX
            +0x0ecWorkingSetPage : Uint4B
            +0x0f0AddressCreationLock: _FAST_MUTEX
            +0x110HyperSpaceLock : Uint4B
            +0x114ForkInProgress : Ptr32_ETHREAD
            +0x118HardwareTrigger : Uint4B
```

Note that the first field (Pcb) is actually a substructure, the kernel process block (KPROCESS), which is where scheduling-related information is stored. To display the format of the kernel process block, type kt_kprocess.

## Thread Block (!strct ethread)

```
lkd> dt nt!_ETHREAD
  +0x000 Tcb            : _KTHREAD
  +0x1c0 CreateTime       : _LARGE_INTEGER
  +0x1c0 NestedFaultCount : Pos 0, 2 Bits
  +0x1c0 ApcNeeded       : Pos 2, 1 Bit
  +0x1c8 ExitTime        : _LARGE_INTEGER
  +0x1c8 LpcReplyChain    : _LIST_ENTRY
  +0x1c8 KeyedWaitChain   : _LIST_ENTRY
  +0x1d0 ExitStatus       : Int4B
  +0x1d0 OfsChain         : Ptr32 Void
  +0x1d4 PostBlockList    : _LIST_ENTRY
  +0x1dc TerminationPort  : Ptr32 _TERMINATION_PORT
  +0x1dc ReaperLink       : Ptr32 _ETHREAD
```

➢ **NOTE: Add "-r" to recurse through substructures**

17

Lab objective: Displaying ETHREAD and KTHREAD Structures

The ETHREAD and KTHREAD structures can be displayed with the dt command in the  kernel debugger. The following output shows the format of an ETHREAD:

```
lkd>dt nt!_ethread
nt!_ETHREAD
             +0x000Tcb : _KTHREAD
             +0x1c0CreateTime : _LARGE_INTEGER
             +0x1c0NestedFaultCount: Pos 0, 2Bits
             +0x1c0ApcNeeded : Pos 2, 1Bit
             +0x1c8ExitTime : _LARGE_INTEGER
             +0x1c8LpcReplyChain : _LIST_ENTRY
             +0x1c8KeyedWaitChain : _LIST_ENTRY
             +0x1d0ExitStatus : Int4B
             +0x1d0OfsChain : Ptr32Void
             +0x1d4PostBlockList : _LIST_ENTRY
             +0x1dcTerminationPort : Ptr32_TERMINATION_PORT
             +0x1dcReaperLink : Ptr32_ETHREAD
             +0x1dcKeyedWaitValue : Ptr32Void
             +0x1e0ActiveTimerListLock: Uint4B
             +0x1e4ActiveTimerListHead: _LIST_ENTRY
             +0x1ecCid : _CLIENT_ID
             +0x1f4LpcReplySemaphore:_KSEMAPHORE
             +0x1f4KeyedWaitSemaphore :_KSEMAPHORE
             +0x208LpcReplyMessage : Ptr32Void
             +0x208LpcWaitingOnPort: Ptr32Void
             +0x20cImpersonationInfo:Ptr32 _PS_IMPERSONATION_INFORMATION
             +0x210IrpList : _LIST_ENTRY
             +0x218TopLevelIrp : Uint4B
             +0x21cDeviceToVerify : Ptr32_DEVICE_OBJECT
             +0x220ThreadsProcess : Ptr32_EPROCESS .........
```

The KTHREAD can be displayed with the similar command nt !_kthread.

## Watching the Scheduler
### Performance Monitor - Options | Chart

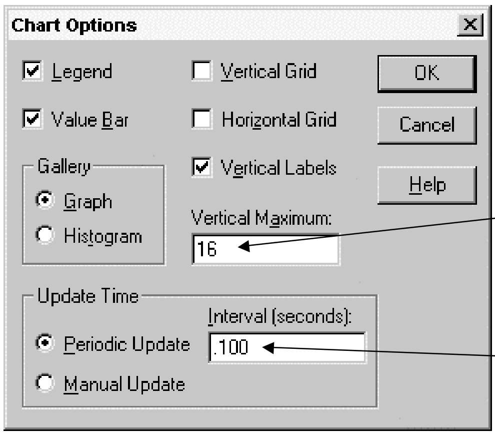


Chart Options

☑ Legend ☐ Vertical Grid

☑ Value Bar ☐ Horizontal Grid

☑ Vertical Labels

Gallery: [✓ Graph] Histogram

Vertical Maximum: 16

Update Time: [✓ Periodic Update] Manual Update

Interval (seconds): .100

**Set chart maximum vertical scale to 16**

**Set update interval to 0.1 seconds or less**

Screen snapshot from: Performance Monitor Options menu | Chart command

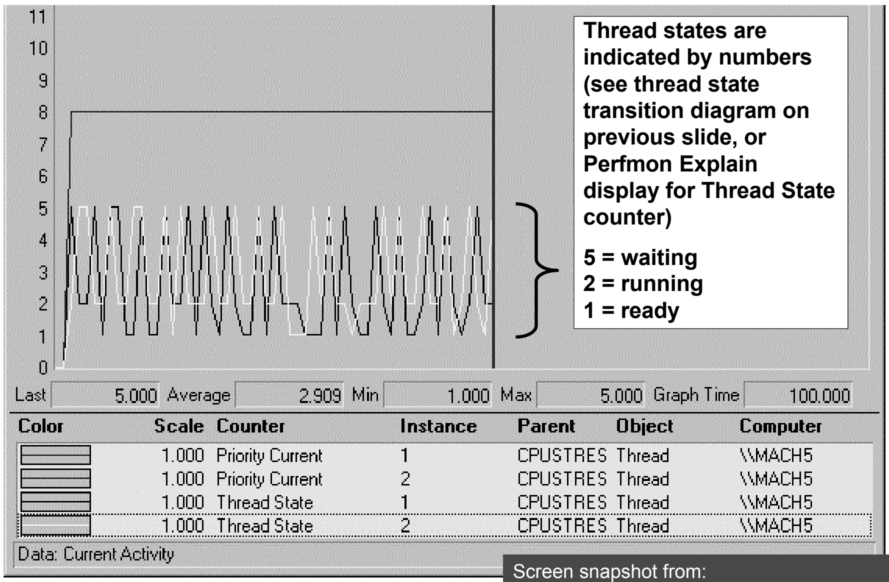
18

Lab objective: Thread-Scheduling State Changes

You can watch thread-scheduling state changes with the Performance tool in Windows. This utility can be useful when you're debugging a multithreaded application if you're unsure about the state of the threads running in the process. To watch thread-scheduling state changes by using the Performance tool, follow these steps:

1. Run the Microsoft Notepad utility (Notepad.exe).
2. Start the Performance tool by selecting Programs from the Start menu and then selecting Performance from the Adminstrative Tools menu.
3. Select chart view if you're in some other view.
4. Right-click on the graph, and choose Properties.
5. Click the Graph tab, and change the chart vertical scale maximum to 7. (As you'll see from the explanation text for the performance counter, thread states are numbered from 0 through 7.) Click OK.
6. Click the Add button on the toolbar to bring up the Add Counters dialog box.
7. Select the Thread performance object, and then select the Thread State counter. Click the Explain button to see the definition of the values.
8. In the Instances box, scroll down until you see the Notepad process (notepad/0); select it, and click the Add button.

## Watching the Scheduler (contd.)
### Performance Monitor


11
10
9
8
7
6
5
4
3
2
1
0
Thread states are indicated by numbers (see thread state transition diagram on previous slide, or Perfmon Explain display for Thread State counter)
5 = waiting
2 = running
1 = ready

Last 5.000 Average 2.909 Min 1.000 Max 5.000 Graph Time 100.000

| Color | Scale | Counter | Instance | Parent | Object | Computer |
|---|---|---|---|---|---|---|
| ▭ | 1.000 | Priority Current | 1 | CPUSTRES | Thread | \\MACH5 |
| ▭ | 1.000 | Priority Current | 2 | CPUSTRES | Thread | \\MACH5 |
| ▭ | 1.000 | Thread State | 1 | CPUSTRES | Thread | \\MACH5 |
| ▭ | 1.000 | Thread State | 2 | CPUSTRES | Thread | \\MACH5 |

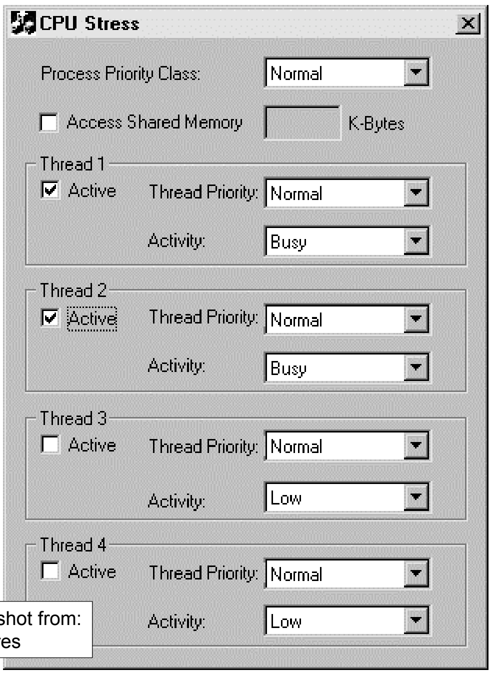
Data: Current Activity

Screen snapshot from:
PerfMon main window, setup from previous slide

9. Scroll back up in the Instances box to the Mmc process (the Microsoft Management Console process running the System Monitor), select all the threads (mmc/ 0, mmc/1, and so on), and add them to the chart by clicking the Add button.
10. Now close the Add Counters dialog box by clicking Close.
11. You should see the state of the Notepad thread (the very top line in the following figure) as a 5, which, as shown in the explanation text you saw under step 5, represents the waiting state (because the thread is waiting for GUI input):
12. Notice that one thread in the Mmc process (running the Performance tool snapin) is in the running state (number 2). This is the thread that's querying the thread states, so it's always displayed in the running state.
13. You'll never see Notepad in the running state (unless you're on a multiprocessor system) because Mmc is always in the running state when it gathers the state of the threads you're monitoring.

## Watching Forground Priority Boosts

- Run: cpustres.exe (Resource Kit)


CPU Stress

Process Priority Class: Normal

☐ Access Shared Memory K-Bytes

Thread 1: ☑ Active — Thread Priority: Normal — Activity: Busy

Thread 2: ☑ Active — Thread Priority: Normal — Activity: Busy

Thread 3: ☐ Active — Thread Priority: Normal — Activity: Low

Thread 4: ☐ Active — Thread Priority: Normal — Activity: Low

Screen snapshot from: Run… cpustres

20

Lab objective: Watching Foreground Priority Boosts and Decays

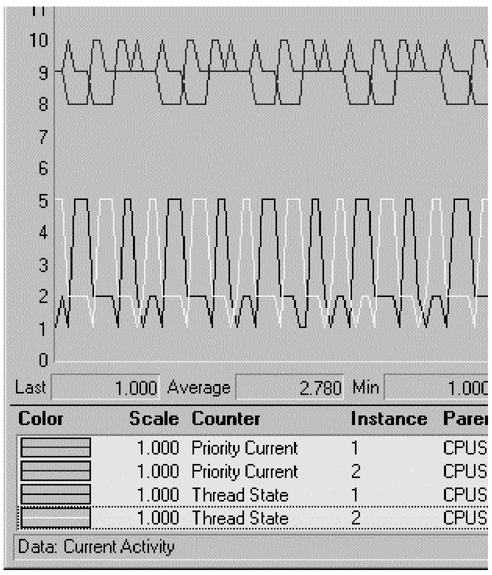
Using the CPU Stress tool (in the resource kit and the Platform SDK), you can watch priority boosts in action. Take the following steps:

1. Open the System utility in Control Panel (or right-click My Computer and select Properties), click the Advanced tab, and click the Performance Options button. Select the Applications option. This causes PsPrioritySeparation to get a value of 2.
2. Run Cpustres.exe.
3. Run the Windows NT 4 Performance Monitor (Perfmon4.exe in the Windows 2000 resource kits). This older version of the Performance tool is needed for this experiment because it can query performance counter values at a frequency faster than the Windows Performance tool (which has a maximum interval of once per second).
4. Click the Add Counter toolbar button (or press Ctrl+I) to bring up the Add To Chart dialog box.
5. Select the Thread object, and then select the Priority Current counter.
6. In the Instance box, scroll down the list until you see the cpustres process. Select the second thread (thread 1). (The first thread is the GUI thread.)
7. Click the Add button, and then click the Done button.
8. Select Chart from the Options menu. Change the Vertical Maximum to 16 and the Interval to 0.010, as follows, and click OK.

## Priority Boost and Decay (contd.)

### Demo with CpuStres and PerfMon

- CpuStres settings:
  - two active threads
  - activity level = busy (about 25% wait time)
  - normal process priority class, normal thread priorities
- Usually only visible in PerfMon if target app owns foreground window (hence longer quantum)
- These are showing +2 boost (from 8 to 10) for foreground apps after wait completion



Last 1.000 Average 2.780 Min 1.00C

| Color | Scale | Counter | Instance | Parei |
|---|---|---|---|---|
| ▭ | 1.000 | Priority Current | 1 | CPUS |
| ▭ | 1.000 | Priority Current | 2 | CPUS |
| ▭ | 1.000 | Thread State | 1 | CPUS |
| ▭ | 1.000 | Thread State | 2 | CPUS |

Data: Current Activity

21

9. Now bring the Cpustres process to the foreground. You should see the priority of the Cpustres thread being boosted by 2 and then decaying back to the base priority.
10. The reason Cpustres receives a boost of 2 periodically is because the thread you're monitoring is sleeping about 75 percent of the time and then waking up—the boost is applied when the thread wakes up. To see the thread get boosted more frequently, increase the Activity level from Low to Medium to Busy. If you set the Activity level to Maximum, you won't see any boosts because Maximum in Cpustres puts the thread into an infinite loop. Therefore, the thread doesn't invoke any wait functions and therefore doesn't receive any boosts.
11. When you've finished, exit Performance Monitor and CPU Stress.

## Priority Boosts on GUI Threads

- Threads that own windows receive an additional boost of 2 when they wake up because of windowing activity, such as the arrival of window messages.
- The windowing system (Win32k.sys) applies this boost when it calls KeSetEvent to set an event used to wake up a GUI thread.
- The reason for this boost is similar to the previous one—to favor interactive applications.

22

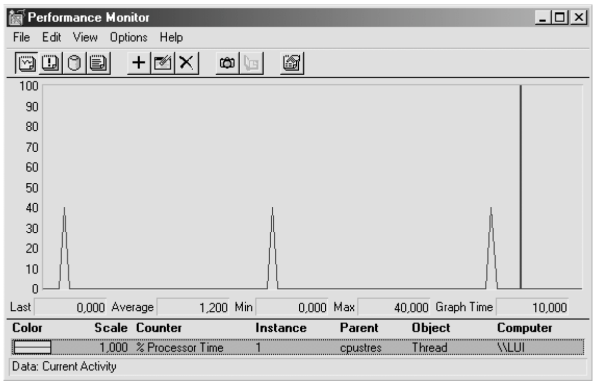
Lab objective: Watching Priority Boosts on GUI Threads

You can also see the windowing system apply its boost of 2 for GUI threads that wake up to process window messages by monitoring the current priority of a GUI application and moving the mouse across the window. Just follow these steps:

1. Open the System utility in Control Panel, click the Advanced tab, and click the Performance Options button. If you're running Windows XP or Windows Server 2003 select the Advanced tab and ensure that the Programs option is selected; if you're running Windows 2000 ensure that the Applications option is selected. This causes PsPrioritySeparation to get a value of 2.
2. Run Notepad from the Start menu by selecting Programs/Accessories/Notepad.
3. Run the Windows NT 4 Performance Monitor (Perfmon4.exe in the Windows 2000 resource kits). This older version of the Performance tool is needed for this experiment because it can query performance counter values at a faster frequency. (The Windows Performance tool has a maximum interval of once per second.) 4
4. Click the Add Counter toolbar button to bring up the Add To Chart dialog box.
5. Select the Thread object, and then select the Priority Current counter.
6. In the Instance box, scroll down the list until you see Notepad thread 0. Click it, click the Add button, and then click the Done button.
7. As in the previous experiment, select Chart from the Options menu. Change the Vertical Maximum to 16 and the Interval to 0.010, and click OK.
8. You should see the priority of thread 0 in Notepad at 8, 9, or 10. Because Notepad entered a wait state shortly after it received the boost of 2 that threads in the foreground process receive, it might not yet have decayed from 10 to 9 and to 8.

## CPU Starvation Resolution

- CpuStres with two compute-bound threads (“maximum” activity level)
- One is at lower priority than the other



Last 0,000 Average 1,200 Min 0,000 Max 40,000 Graph Time 10,000

| Color | Scale | Counter | Instance | Parent | Object | Computer |
|---|---|---|---|---|---|---|
| ▭ | 1,000 | % Processor Time | 1 | cpustres | Thread | \\LUI |

Data: Current Activity

23

Lab objective: Watching Priority Boosts for CPU Starvation

Using the CPU Stress tool (in the resource kit and the Platform SDK), you can watch priority boosts in action. In this experiment, we’ll see CPU usage change when a thread’s priority is boosted. Take the following steps:

1. Run Cpustres.exe. Change the activity level of the active thread (by default, Thread 1) from Low to Maximum. Change the thread priority from Normal to Below Normal.
2. Run the Windows NT 4 Performance Monitor (Perfmon4.exe in the Windows 2000 resource kits). Again, you need the older version for this experiment because it can query performance counter values at a frequency faster than once per second.
3. Click the Add Counter toolbar button (or press Ctrl+I) to bring up the Add To Chart dialog box.
4. Select the Thread object, and then select the % Processor Time counter.
5. In the Instance box, scroll down the list until you see the cpustres process. Select the second thread (thread 1). (The first thread is the GUI thread.)
6. Click the Add button, and then click the Done button.
7. Raise the priority of Performance Monitor to real-time by running Task Manager, clicking the Processes tab, and selecting the Perfmon4.exe process. Right-click the process, select Set Priority, and then select Realtime.
8. Run another copy of CPU Stress. In this copy, change the activity level of Thread 1 from Low to Maximum.
9. Now switch back to Performance Monitor. You should see CPU activity every 4 or so seconds because the thread is boosted to priority 15.