Unit OS2:
Operating System Principles

2.5. Lab Slides & Lab Manual

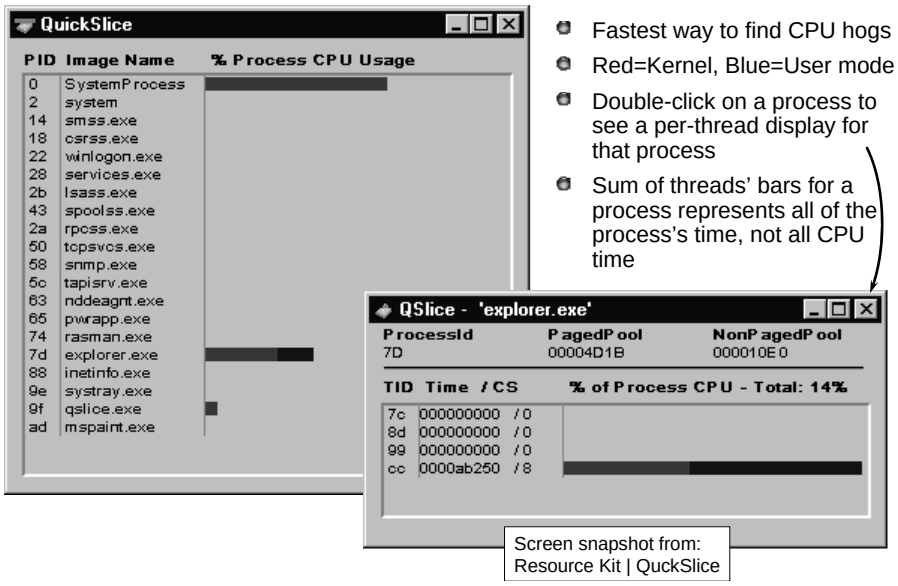
Roadmap for Section 2.6.

Lab experiments investigating:

- Process Execution
- Object Manager & Handles
- Interrupt Handling
- Memory Pools Labs
- System Threads
- System Processes

Kernel-Mode vs User-Mode

QuickSlice (qslice.exe)



4

EXPERIMENT: Viewing Thread Activity with QuickSlice

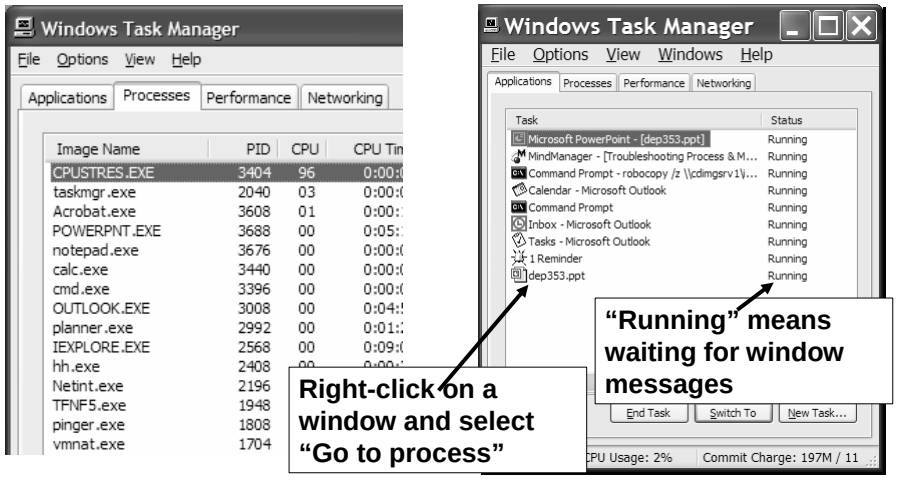
QuickSlice gives a quick, dynamic view of the proportions of system and kernel time that each process currently running on your system is using. On line, the red part of the bar shows the amount of CPU time spent in kernel mode, and the blue part shows the user-mode time.

The total of all bars shown in the QuickSlice window should add up to 100 percent of CPU time. To run QuickSlice, click the Start button, choose Run, and enter Qslice.exe (assuming the Windows 2000 resource kit is in your path). For example, try running a graphics-intensive application such as Paint (Mspaint.exe). Open QuickSlice and Paint side by side, and draw squiggles in the Paint window.

For additional information about the threads in a process, you can also double-click on a process (on either the process name or the colored bar).

Task Manager: Processes vs Applications Tabs

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



7

EXPERIMENT: Viewing Process Information with Task Manager

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:

press Ctrl+Shift+Esc,

right-click on the taskbar and select Task Manager, or

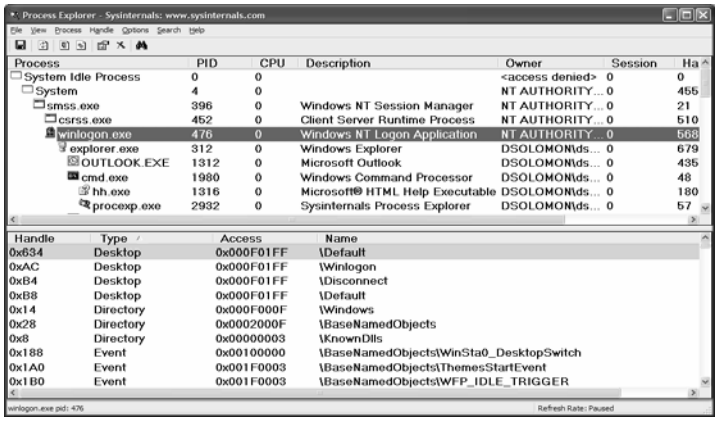
press Ctrl+Alt+Delete and click the Task Manager button. Once Task Manager has started, click the Processes tab to see the list of running processes. Notice that processes are identified by the name of the image of which they are an instance. Unlike some objects in Windows, processes can't be given global names.

To display additional details, choose Select Columns from the View menu and select additional columns to be added.

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).

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



8

EXPERIMENT: Viewing Process Details with Process Explorer

Download the latest version of Process Explorer from www.sysinternals.com and run it. 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 example: In the preceding example, the on-demand symbol server is being used to access symbols and a copy of the symbol files are being stored on the local machine in the c:\symbols folder. For more information on configuring use of the symbol server, see http:// www.microsoft.com/whdc/ddk/debugging/symbols.mspx.

When Process Explorer starts, it shows by default the process list on the top half and the open handles for the currently selected process on the bottom half. It also shows the image description, company name, and full path if you hover the mouse pointer over the process name.

Process Explorer Lab: Process List

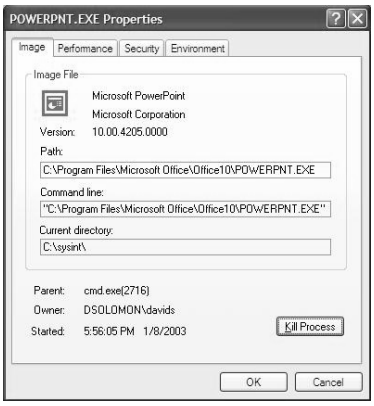
1. Run Process Explorer & maximize window
2. Run Task Manager – click on Processes tab
3. Arrange windows so you can see both
4. Notice process tree vs flat list in Task Manager
   - If parent has exited, process is left justified
5. Sort on first column (“Process”) and note tree view disappears
6. Sort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

Here are a few steps to walk you through some basic capabilities of Process Explorer:

1. Turn off the lower pane by deselecting View, Show Lower Pane.
2. Notice that processes hosting services are highlighted by default in pink. Your own processes are highlighted in blue.
3. Hover your mouse pointer over the image name for processes, and notice the full path displayed by the ToolTip.
4. Click on View, Select Columns, and add the image path.
5. Sort on the process column, and notice the tree view disappears. (You can either display tree view or sort by any of the columns shown.) Click again to sort from Z to A. Then click again and the display returns to tree view.
6. Deselect View, Show Processes From All Users to show only your processes.
7. Go to Options, Difference Highlight Duration, and change the value to 5 seconds. Then launch a new process (anything), and notice the new process highlighted in green for 5 seconds. Exit this new process, and notice the process is highlighted in red for 5 seconds before disappearing from the display. This can be useful to see processes being created and exiting on your system.
8. Finally, double-click on a process and explore the various tabs available from the process properties display.

Process Explorer Lab: Image Information

- Double click on Explorer.exe to bring up process properties
- Image tab:
  - Description, company name, version (from .EXE)
  - Full image path
  - Command line used to start process
  - Current directory
  - Parent process
  - User name
  - Start time



13

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.

Services: talk about troubleshooting them later.

Lab: Viewing Process Activity

1. Task Manager:
   - Applications tab: find the process that owns a window (right mouse click on window title)
   - Process tab: add a few additional columns: Virtual Memory size, Handle count, Thread count
     - Windows 2000: add I/O counters; right click on a process & notice “end process tree” option
2. Look at process hierarchy with TLIST /T
   - Start an NT command prompt, then run Notepad from command prompt, then look at TLIST /T output
   - Exit the command prompt and notice “orphan” process with TLIST /T

25

EXPERIMENT: Viewing the Process Tree

One unique attribute about a process that most tools don’t display is the parent or creator process ID. You can retrieve this value with the Performance tool (or programmatically) by querying the Creating Process ID. The Tlist.exe tool (in the Windows Debugging Tools) can show the process tree by using /t switch.

The list indents each process to show its parent/child relationship. Processes whose parents aren’t alive are left-justified (as is Explorer.exe in the preceding example) because even if a grandparent process exists, there’s no way to find that relationship. Windows maintains only the creator process ID, not a link back to the creator of the creator, and so forth.

To demonstrate the fact that Windows doesn’t keep track of more than just the parent process ID, follow these steps: 1. Open a Command Prompt window. 2. Type start cmd (which starts a second Command Prompt). 3. Bring up Task Manager. 4. Switch to the second Command Prompt. 5. Type mspaint (which runs Microsoft Paint). 6. Click the intermediate (second) Command Prompt window. 7. Type exit. (Notice that Paint remains.) 8. Switch to Task Manager. 9. Click the Applications tab. 10. Right-click on the Command Prompt task, and select Go To Process. 11. Click on the Cmd.exe process highlighted in gray. 12. Right-click on this process, and select End Process Tree. 13. Click Yes in the Task Manager Warning message box.

The first Command Prompt window will disappear, but you should still see the Paintbrush window because it was the grandchild of the Command Prompt process you terminated; and because the intermediate process was terminated, there was no link between the parent and the grandchild.

PS Tools

- Group of 12 process/system control tools
  - Where'd the "Ps" come from?
    - The UNIX process listing tool is named "ps"
    - The first PsTool was a UNIX "ps"-equivalent, PsList
- They all work on NT4/2000/XP/2003
- They all work remotely as well as locally
  - Require admin rights to remote system
    - Can specify credentials with "-u" switch
- None require manual remote software installation

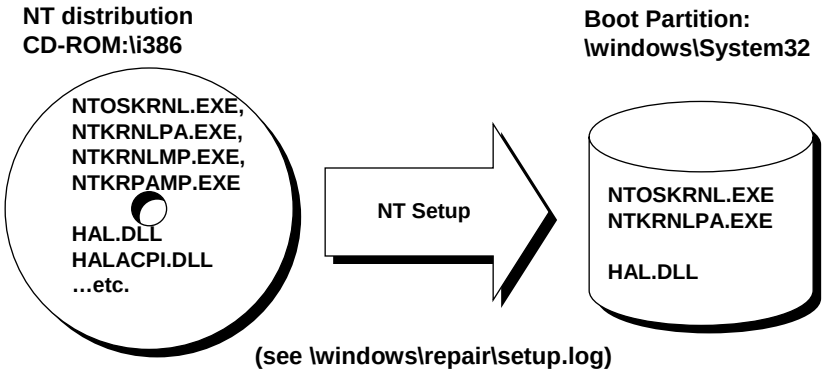
28

All PS Tools

PsFile – lists & closes remote file opens
PsShutdown – remote shutdown, lock workstation, log off user
PsExec – run an app on a remote system
PsList – list processes & threads
PsUptime – system up time
PsInfo – display general system info
PsGetsid – displays computer or user SIDs
PsService – service process control (like SC in XP)
PsLoglist – dumps event log in text
PsSuspend – suspend a process
PsKill – kill processes
PsLoggedon – lists local and remote logon sessions
PsPassword – change local/remote passwords

NTOSKRNL & HAL Selection

- Selected at installation time
  - See \windows\repair\setup.log to find out which one
  - Can select manually at boot time with /HAL= in boot.ini



31

EXPERIMENT:

Looking at Multiprocessor-Specific Support Files on Windows 2000

You can see the files that are different for a 32-bit Windows 2000 multiprocessor system by looking at the driver details for the Computer in Device Manager:

Open the System properties (either by selecting System from Control Panel or by right-clicking the My Computer icon on your desktop and selecting Properties).

Click the Hardware tab.

Click Device Manager.

Expand the Computer object.

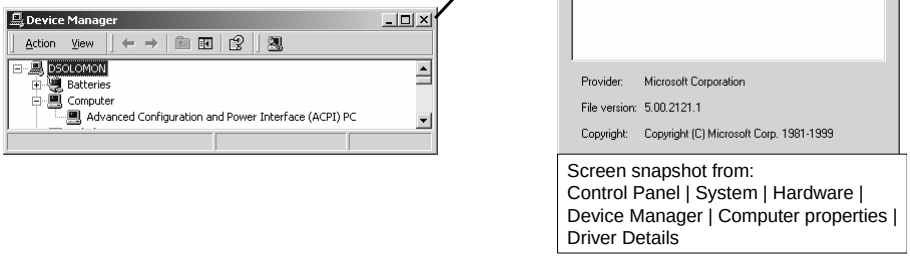
Double-click the child node underneath Computer.

Click the Driver tab.

Click Driver Details.

NTOSKRNL & HAL Selection

- Can also see by viewing the “device drivers” for the Computer
  - Go to Control Panel->System – Hardware tab
  - Click on “Device Manager”
  - Click on “Computer”
  - Right click/Properties on “driver” for PC



Screen snapshot from:
Control Panel | System | Hardware | Device Manager | Computer properties | Driver Details

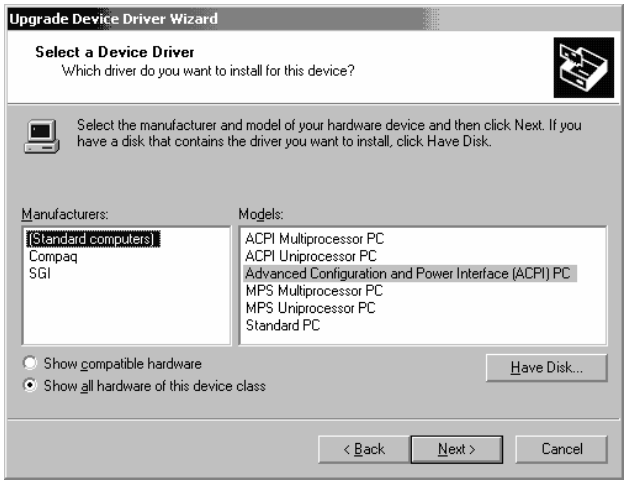
EXPERIMENT: Viewing the Base HALs Included with Windows

To view the HALs included with Windows, open the file Driver.cab in the appropriate architecture-specific folder underneath \Windows\Driver Cache.

(For example, for x86 systems, the file name is \Windows\Driver Cache\i386\Driver.cab.) Scroll down to the files beginning with “Hal” and you will see the HAL DLLs.

HAL Choices

- To see the HAL list, do an “update driver” on the drivers for the “Computer” and specify manual selection from the list;



33

EXPERIMENT: Determining Which HAL You’re Running

There are two ways to determine which HAL you’re running:

Open the file \Windows\Repair\Setup.log, search for Hal.dll, and look at the filename after the equals sign. This is the name of the HAL on the distribution media extracted from Driver.cab.

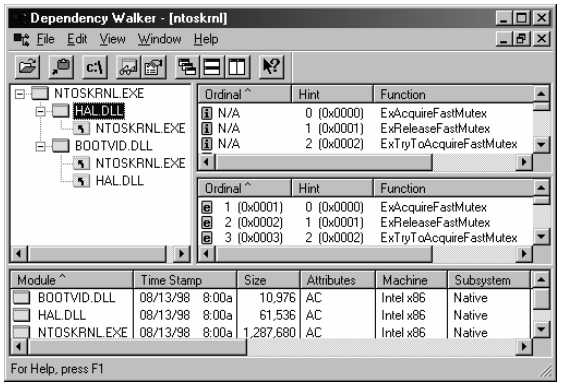
In Device Manager (right-click on the My Computer icon on your desktop, select Properties, click on the Hardware tab, and then click Device Manager), look at the name of the “driver” under the Computer device type.

Examining NTOSKRNL & HAL Linkage

•Tool: Dependency Walker (Depends.Exe in Resource Kit & Platform SDK)

•Allows viewing of image->DLL relationships, imports, and exports

- NTOSKRNL.EXE
  - Executive and Kernel
- HAL.DLL
  - Hardware Abstraction Layer - interface to hardware platform
- BOOTVID.DLL
  - Boot video driver
  - Added in Win2000
- KDCOM.DLL
  - Kernel debugger communication code



34

EXPERIMENT: Viewing NTOSKRNL and HAL Image Dependencies

You can view the relationship of the kernel and HAL images by examining their export and import tables using the Dependency Walker tool (Depends.exe), which is contained in the Windows Support Tools and the Platform SDK. To examine an image in the Dependency Walker, select Open from the File menu to open the desired image file.

Notice that Ntoskrnl is linked against the HAL, which is in turn linked against Ntoskrnl. (They both use functions in each other.) Ntoskrnl is also linked against Bootvid.dll, the boot video driver that is used to implement the GUI startup screen. On Windows XP and later, you will see an additional DLL, Kdcom.dll, in the list. This contains kernel debugger infrastructure code that used to be part of Ntoskrnl.exe. For a detailed description of the information displayed by this tool, see the Dependency Walker help file (Depends.hlp).

Kernel-Mode Device Drivers

- Separate loadable modules (drivername.SYS)
  - Linked like .EXEs
  - Typically linked against NTOSKRNL.EXE and HAL.DLL
  - Only one version of each driver binary for both uniprocessor (UP) and multiprocessor (MP) systems…
  - … but drivers call routines in the kernel that behave differently for UP vs. MP Versions
- Defined in registry
  - Same area as Windows services (t.b.d.) - differentiated by Type value
- Several types:
  - “ordinary”, file system, NDIS miniport, SCSI miniport (linked against port drivers), bus drivers
  - More information in I/O subsystem section
- To view loaded drivers, run drivers.exe
  - Also see list at end of output from pstat.exe – includes addresses of each driver
- To update & control:
  - System properties->Hardware Tab->Device Manager
  - Computer Management->Software Environment->Drivers

35

EXPERIMENT: Viewing the Installed Device Drivers

You can list the installed drivers by running Computer Management. (From the Start menu, select Programs, Administrative Tools, and then Computer Management; or from Control Panel, open Administrative Tools and select Computer Management.) From within Computer Management, expand System Information and then Software Environment, and open Drivers.

Alternatively, you list the currently loaded device drivers with the Drivers utility (Drivers.exe in the Windows 2000 resource kits) or the Pstat utility (Pstat.exe in the Windows XP Support Tools, Windows Server 2003 Support Tools, Windows 2000 resource kits, and the Platform SDK).

Digging Into NTOSKRNL.EXE

- Exported symbols
  - Functions and global variables Microsoft wants visible outside the image (e.g. used by device drivers)
  - About 1500 symbols exported
  - Ways to list:
    - Dependency Walker (File->Save As)
    - Visual C++ “link /dump /exports ntoskrnl.exe”
- Global symbols
  - Over 9000 global symbols in XP/Server 2003 (Windows NT 4.0 was 4700)
    - Many variables contain values related to performance and memory policies
  - Ways to list:
    - Visual C++: “dumpbin /symbols /all ntoskrnl.exe” (names only)
    - Kernel debugger: `“x nt!*”`
      - Module name of NTOSKRNL is “NT”

36

Peering into Undocumented Interfaces

Examining the names of the exported or global symbols in key system images (such as Ntoskrnl.exe, Hal.dll, or Ntdll.dll) can be enlightening—you can get an idea of the kinds of things Windows can do versus what happens to be documented and supported today. Of course, just because you know the names of these functions doesn’t mean that you can or should call them—the interfaces are undocumented and are subject to change. We suggest that you look at these functions purely to gain more insight into the kinds of internal functions Windows performs, not to bypass supported interfaces.

For example, looking at the list of functions in Ntdll.dll gives you the list of all the system services that Windows provides to user-mode subsystem DLLs versus the subset that each subsystem exposes. Although many of these functions map clearly to documented and supported Windows functions, several are not exposed via the Windows API. (See the article “Inside the Native API” from www.sysinternals.com.)

Conversely, it’s also interesting to examine the imports of Windows subsystem DLLs (such as Kernel32.dll or Advapi32.dll) and which functions they call in Ntdll.

Another interesting image to dump is Ntoskrnl.exe—although many of the exported routines that kernel-mode device drivers use are documented in the Windows DDK, quite a few are not. You might also find it interesting to take a look at the import table for Ntoskrnl and the HAL; this table shows the list of functions in the HAL that Ntoskrnl uses and vice versa.

Naming Convention for Internal NT Routines

- Two- or three-letter component code in beginning of function name

Executive

Ex	- General executive routine management	Ob	- Object management
Exp	- Executive private (not exported)	Io	- I/O subsystem
Cc	- Cache manager	Se	- Security
Mm	- Memory management	Ps	- Process structure
Rtl	- Run-Time Library	Lsa	- Security Authentication
FsRtl	- File System Run-Time Lib	Zw	- File access, etc.

Kernel

Ke	- Kernel
Ki	- Kernel internal (not available outside the kernel)

HAL

Hal	- Hardware Abstraction Layer
READ_, WRITE_	- I/O port and register access

37

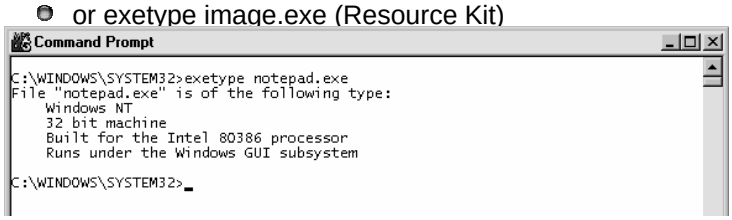
Table 2-7 in Windows Internals, 4th edition lists most of the commonly used function name prefixes for the executive components. Each of these major executive components also uses a variation of the prefix to denote internal functions—either the first letter of the prefix followed by an i (for internal) or the full prefix followed by a p (for private). For example, Ki represents internal kernel functions, and Psp refers to internal process support functions.

Header of Executable File Specifies Subsystem Type

- Subsystem for each .exe specified in image header
  - see winnt.h

```
IMAGE_SUBSYSTEM_UNKNOWN     0   // Unknown subsystem
IMAGE_SUBSYSTEM_NATIVE      1   // Image doesn't require a subsystem
IMAGE_SUBSYSTEM_WINDOWS_GUI 2   // Windows subsystem (graphical app)
IMAGE_SUBSYSTEM_WINDOWS_CUI 3   // Windows subsystem (character cell)
IMAGE_SUBSYSTEM_OS2_CUI     5   // OS/2 subsystem
IMAGE_SUBSYSTEM_POSIX_CUI   7   // Posix subsystem
```

  - see Explorer / QuickView (right-click on .exe or .dll file)
  - or exetype image.exe (Resource Kit)



39

LAB:

5 different examples:

- exetype on ntoskrnl.exe (native),
- exetype on notepad.exe (Win32 GUI),
- exetype on cmd.exe (Win32 CUI),
- exetype on write.exe (Win 3.1),
- exetype on qbasic.exe (DOS)

Lab: Subsytems & Images

- Look at subsystem startup information in registry
- Using EXETYPE, look at subsystem types for:
  - \windows\system32\notepad.exe,
  - cmd.exe, csrss.exe

40

EXPERIMENT: Viewing the Image Subsystem Type

You can see the image subsystem type by using either the Exetype tool in the Windows resource kits or the Dependency Walker tool (Depends.exe) in the Windows Support Tools and Platform SDK. For example, notice the image types for two different Windows images, Notepad.exe (the simple text editor) and Cmd.exe (the Windows command prompt):

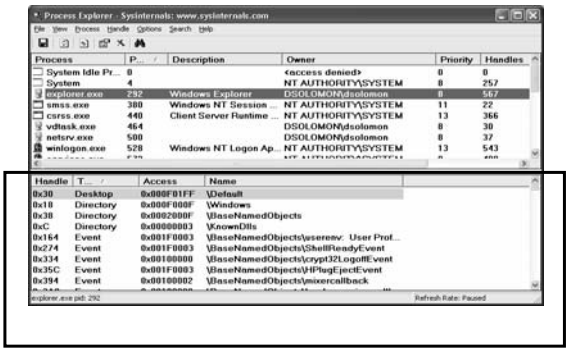
```
C:\>exetype\Windows\system32\notepad.exe
File "\Windows\system32\notepad.exe" is of the following type:
Windows NT 32 bit machine Built for the Intel 80386 processor
Runs under the Windows GUI subsystem

C:\>exetype\Windows\system32\cmd.exe
File "\Windows\system32\cmd.exe" is ofthe followingtype:
Windows NT 32 bit machine Built for the Intel 80386 processor
Runs under the Windows character-based subsystem
```

This shows that Notepad is a GUI program while Cmd is a console or character-based program. And although the output of the Exetype tool implies there are two different subsystems for GUI and character-based programs, there is just one Windows subsystem. Also, Windows isn’t supported on the Intel 386 processor (or the 486 for that matter)—the text output by the Exetype program hasn’t been updated.

Examining Open Handles: Sysinternals Tools

- Process Explorer (GUI version) or handle (character cell version) from www.sysinternals.com
  - Uses a device driver to walk handle table, so doesn't need Global Flag set



EXPERIMENT: Viewing Open Handles

Run Process Explorer, and make sure the lower pane is enabled and configured to show open handles. (Click on View, Lower Pane View, and then Handles). Then open a command prompt and view the handle table for the new Cmd.exe process. You should see an open file handle to the current directory.

If you then change the current directory with the CD command, you will see in Process Explorer that the handle to the previous current directory is closed and a new handle is opened to the new current directory. The previous handle is highlighted briefly in red, and the new handle is highlighted in green. The duration of the highlight can be adjusted by clicking Options and then Difference Highlight Duration.

Process Explorer's differences highlighting feature makes it easy to see changes in the handle table. For example, if a process is leaking handles, viewing the handle table with Process Explorer can quickly show what handle or handles are being opened but not closed. This information can assist the programmer to find the handle leak.

Viewing Open Handles

- Handle View
  - Suggestion: sort by type or path column
  - Objects of type “File” and “Key” are most interesting for general troubleshooting
  - By default, shows named objects
    - Click on Options->Show Unnamed Objects
- Solve file locked errors
  - Use the search feature to determine what process is holding a file or directory open
  - Can even close an open files (be careful!)
- Understand resources used by an application
  - Files
  - Registry keys
- Detect handle leaks using refresh difference highlighting
  - Can also view the state of synchronization objects (mutexes, semaphores, events)

44

You can also display the open handle table by using the command line Handle tool from www.sysinternals.com. For example, note the following partial output of Handle examining the handle table for a Cmd.exe process before and after changing the directory:

```
C:\>handle -p cmd.exe

Handlev 2.2  Copyright(C)1997-2004 MarkRussinovich
Sysinternals - www.sysinternals.com
------------------------------------------------------------
cmd.exe pid:3184BIGDAVID\dsolomon
  b0:File        C:\

C:\>cd windows

C:\WINDOWS>handle -p cmd.exe

Handlev 2.2  Copyright(C)1997-2004 MarkRussinovich
Sysinternals - www.sysinternals.com
------------------------------------------------------------
cmd.exe pid:3184BIGDAVID\dsolomon
  b4:File C:\WINDOWS
```

Process Explorer Lab: Handles

1. Run Microsoft Word
2. In Word, type some text and save file as "c:\test.doc" (don't close it – leave Word open)
3. From a command prompt type:
   "del c:\test.doc"
   (should get file locked)
4. In ProcExp, hit F5 and then use Search to find new handle to test.doc
5. In Word, close the file (leave Word running)
6. In ProcExp, hit F5 and see handle closed to file (in red)

EXPERIMENT: Creating the Maximum Number of Handles

The test program Testlimit from www.sysinternals.com/windowsinternals.shtml has an option to open handles to an object until it cannot open any more handles. You can use this to see how many handles can be created in a single process on your system. Because handle tables are allocated from paged pool, you might run out of paged pool before you hit the maximum number of handles that can be created in a single process.

To see how many handles you can create on your system, follow these steps:

1. Download the Testlimit zip file from the link just mentioned, and unzip it into a directory.
2. Run Process Explorer, and click View and then System Information. Notice the current and maximum size of paged pool. (To display the maximum pool size values, Process Explorer must be configured properly to access the symbols for the kernel image, Ntoskrnl.exe.) Leave this system information display running so that you can see pool utilization when you run the Testlimit program.
3. Open a command prompt.
4. Run the Testlimit program with the "-h" switch (do this by typing testlimit –h). When Testlimit fails to open a new handle, it will display the total number of handles it was able to create. If the number is less than approximately 16 million, you are probably running out of paged pool before hitting the theoretical per-process handle limit.
5. Close the command-prompt window; doing this will kill the Testlimit process, thus closing all the open handles.

Viewing Open Handles with Kernel Debugger

- If looking at a dump, use !handle in Kernel Debugger (see help for options)

```
lkd> !handle 0 f 9e8 file

processor number 0
Searching for Process with Cid == 9e8
Searching for handles of type file

PROCESS 82ce72d0  SessionId: 0  Cid: 09e8    Peb: 7ffdf000  ParentCid: 06ec

   DirBase: 06602000  ObjectTable: e1c879c8  HandleCount: 430.
   Image: POWERPNT.EXE
...

0280: Object: 82c5e230  GrantedAccess: 00120089

Object: 82c5e230  Type: (82fdde70) File
   ObjectHeader: 82c5e218

      HandleCount: 1  PointerCount: 1

      Directory Object: 00000000  Name:
            \slides\ntint\new\4-systemarchitecture.ppt {HarddiskVolume1}
```


EXPERIMENT: Viewing the Handle Table with the Kernel Debugger

The !handle command in the kernel debugger takes three arguments:

!handle <handleindex><flags><processid>

The handle index identifies the handle entry in the handle table. (Zero means display all handles.) The first handle is index 4, the second 8, and so on. For example, typing !handle 4 will show the first handle for the current process. The flags you can specify are a bitmask, where bit 0 means display only the information in the handle entry, bit 1 means display free handles (not just used handles), and bit 2 means display information about the object that the handle refers to.

The following command displays full details about the handle table for process ID 0x408:

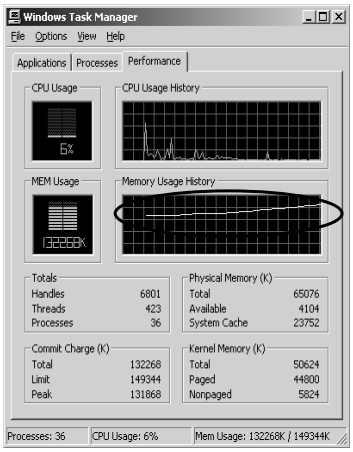
```
kd> !handle0 7408
processor number0
Searching for Process with Cid==408
PROCESS 865f0790 SessionId:0Cid:0408 Peb:7ffdf000 ParentCid: 01dc
   DirBase:04fd3000 ObjectTable: 856ca888 TableSize: 21.
   Image: i386kd.exe
Handle Table at e2125000 with 21 Entries in use
0000:free handle
0004:Object: e20da2e0 GrantedAccess:000f001f
Object: e20da2e0 Type:(81491b80)Section
   ObjectHeader: e20da2c8
   HandleCount:1 PointerCount:1
0008:Object: 80b13330 GrantedAccess:00100003
Object: 80b13330 Type:(81495100)Event
   ObjectHeader: 80b13318
   HandleCount:1 PointerCount:1
```

Lab: Causing a Pool Leak

- Run NotMyFault and select "Leak Pool"

  (available from http://www.sysinternals.com/files/notmyfault.zip)

  - Allocates paged pool buffers and doesn't free them
  - Stops leaking when you select "Stop Leaking"



EXPERIMENT: Troubleshooting a Pool Leak

In this experiment, you will fix a real paged pool leak on your system so that you can put to use the techniques described in the previous section to track down the leak. The leak will be generated by the NotMyFault tool, which you can download from www.sysinternals.com/windowsinternals.shtml.

When you run NotMyFault.exe, it loads a device driver Myfault.sys and presents the following dialog box:

1. Click the Leak Pool button. This causes NotMyFault to begin sending requests to the Myfault device driver to allocate paged pool. NotMyFault will continue to do this until you click the Stop Leaking button. Note that the paged pool is not released even when you close the program; the pool is permanently leaked until you reboot the system.
2. While the pool is leaking, first open Task Manager and click on the Performance tab. You should notice Paged Pool climbing. You can also check this with Process Explorer's System Information display. (Click on Show and then System Information.)
3. To determine the pool tag that is leaking, run Poolmon and press the "b" key to sort by the number of bytes. Press "p" twice so that Poolmon is showing only paged pool. You should notice the pool tag "Leak" climbing to the top of the list. -- Press Stop Leaking !
4. Run Strings (from www.sysinternals.com) to look for driver binaries that contain the pool tag "Leak":

   Strings \windows\system32\drivers*.sys | findstrLeak

Understanding Pool Usage

- Three options:
  1. Poolmon (in Support Tools and Device Driver Kit)
  2. Kernel Debugger *!Poolused* command
  3. Driver Verifier (in Windows 2000 and later)

49

EXPERIMENT: Determining the Maximum Pool Sizes

Because paged and nonpaged pool represent a critical system resource, it is important to know when you're nearing the maximum size computed for your system so that you can determine whether you need to override the default maximum with the appropriate registry values. The pool-size performance counters report only the current size, however, not the maximum size.

So you don't know when you're reaching the limit until you've exhausted pool. (As noted earlier, future versions of Windows might support dynamically sized pools. Therefore, the need to check the pool maximums might no longer be necessary in the future.) You can obtain the pool maximums by using either Process Explorer or live kernel debugging. To view pool maximums with Process Explorer, click on View, System Information.

Troubleshooting Pool Leaks With Poolmon

- Once you find pool tag that is leaking, need to find which driver is creating the tag
  1. Try looking up in Windows Debugging Tools subfolder \triage\pooltag.txt
     - May not be there if 3rd party driver
  2. If not, run Strings (from Sysinternals) on all drivers:

```
strings \windows\system32\drivers\*.sys | findstr
  Xyzz
```

- Or, use new Poolmon in 2003 DDK to generate local pool tags
  - Poolmon -c will create a “localtag.txt” (if not present)
  - Scans binaries of loaded drivers for pool tags
    - Still missed drivers that are loaded but deleted

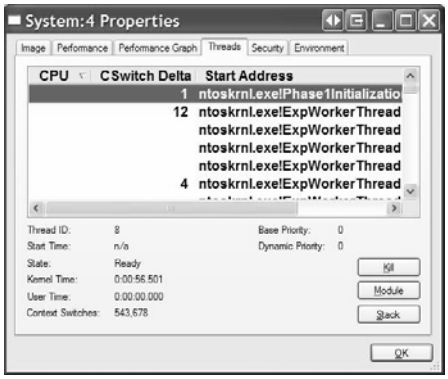
51

To determine the pool tag that is leaking, run Poolmon and press the “b” key to sort by the number of bytes.

Press “p” twice so that Poolmon is showing only paged pool. You should notice the pool tag “Leak” (our victim) climbing to the top of the list. (Poolmon shows changes to pool allocations by highlighting the lines that change.)

Identifying System Threads: Process Explorer

- With Process Explorer:
  - Double click on System process
  - Go to Threads tab – sort by CPU time
- As explained before, threads run between clock ticks (or at high IRQL) and thus don't appear to run
  - Sort by context switch delta column



EXPERIMENT: Mapping a System Thread to a Device Driver

In this experiment, we'll see how to map CPU activity in the System process to the responsible system thread (and the driver it falls in) generating the activity. This is important because when the System process is running, you must go to the thread granularity to really understand what's going on. For this experiment, we will generate system thread activity by generating file server activity on your machine. (The file server driver, Srv.sys, creates system threads to handle inbound requests for file I/O. See Chapter 13 for more information on this component.)

1. Open a command prompt.
2. Do a directory listing of your entire C drive using a network path to access your C drive. For example, if your computer name is COMPUTER1, type "dir \\computer1\c$ /s". (The /s switch lists all subdirectories.)
3. Run Process Explorer, and double-click on the System process.
4. Click on the Threads tab.
5. Sort by the CSwitch Delta (context switch delta) column. You should see one or more threads in Srv.sys running.

If you see a system thread running and you are not sure what the driver is, press the Module button, which will bring up the file properties.

System Threads Lab

1. Generate network file access activity, for example:
   "dir \\computername\c$ /s"
   - System process should be consuming CPU time
2. Open System process process properties
3. Go to Threads tab
4. Sort by CPU time and find thread(s) running
5. Determine what driver these are in

EXPERIMENT: Identifying System Threads in the System Process

You can see that the threads inside the System process must be kernel-mode system threads because the start address for each thread is greater than the start address of system space (which by default begins at 0x80000000, unless the system was booted with the /3GB Boot.ini switch). Also, if you look at the CPU time for these threads, you'll see that those that have accumulated any CPU time have run only in kernel mode.

To find out which driver created the system thread, look up the start address of the thread (which you can display with Pviewer.exe) and look for the driver whose base address is closest to (but before) the start address of the thread. Both the Pstat utility (at the end of its output) as well as the !drivers kernel debugger command list the base address of each loaded device driver.

To quickly find the current address of the thread, use the !stacks 0 command in the kernel debugger. Here is sample output from a live system (using LiveKd):

Identifying System Threads: User-Mode Tools

- To really understand what's going on, must find which driver a thread "belongs to"
- With standard user-mode tools:
  1. PerfMon: monitor %Processor time for each thread in System process & determine which thread(s) are running
  2. Pviewer: get "Start address" (address of thread function) of running thread(s)
  3. Pstat: find which driver thread start address falls in
     - Look for what driver starts near the thread start address

58

EXPERIMENT: Viewing the Real User Start Address for Windows Threads

The fact that each Windows thread begins execution in a system-supplied function (and not the user-supplied function) explains why the start address for thread 0 is the same for every Windows process in the system (and why the start addresses for secondary threads are also the same).

The start address for thread 0 in Windows processes is the Windows start-of-process function; the start address for any other threads would be the Windows start-of-thread function. To see the user-supplied function address, use the Tlist utility in the Windows Support Tools. Type tlist process-name or tlist process-id to get the detailed process output that includes this information. For example, compare the thread start addresses for the Windows Explorer process as reported by Pstat (in the Platform SDK) and Tlist:

```
C:\> pstat
pid:3f8 pri:8 Hnd: 329Pf: 80043Ws: 4620K explorer.exe
  tid pri CtxSwtch StrtAddr UserTime KernelTime State
  7c 9 16442 77E878C10:00:01.241 0:00:01.251 Wait:UserRequest
  42c 11 157888 77E92C500:00:07.110 0:00:34.309 Wait:UserRequest
  44c 8 6357 77E92C500:00:00.070 0:00:00.140 Wait:UserRequest
  1cc 8 3318 77E92C500:00:00.030 0:00:00.070 Wait:DelayExecution
```

The start address of thread 0 reported by Pstat is the internal Windows start-of-process function; the start addresses for threads 1 through 3 are the internal Windows start-ofthread functions. Tlist, on the other hand, shows the user-supplied Windows start address (the user function called by the internal Windows start function).

Identifying System Threads: Kernel Debugger

- With Kernel Debugger:
  - ln (“List Near”) <startaddress> will give name of driver and function
  - Use !process or !thread to see kernel stack

```
lkd> ln 8061adb8
(8061adb8)   nt!MiModifiedPageWriter   |  (8061af38)
lkd> !process 4
            …
 THREAD 816113e0  Cid 8.50  WAIT: (Executive) KernelMode Non-Alertable
          f5c67d70  NotificationTimer
          80482540  SynchronizationEvent
      Start Address nt!KeBalanceSetManager (0x804634e0)
      Stack Init f5c68000 Current f5c67cc0 Base f5c68000 Limit f5c65000 Call 0
ChildEBP RetAddr  Args to Child
f5c67cd8 8042d5a3 ffffffff ff676980 00000000 nt!KiSwapThread+0xc5
f5c67d0c 8046355e 00000002 f5c67d98 00000001 nt!KeWaitForMultipleObjects+0x266
f5c67da8 80454faf 00000000 00000000 00000000 nt!KeBalanceSetManager+0x7e
f5c67ddc 80468ec2 804634e0 00000000 00000000 nt!PspSystemThreadStartup+0x69
00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x16
```

Lab: Solitaire as a Service

- Create a service to run Sol.exe

  Sc create dumbservice binpath= c:\windows\system32\sol.exe
- Start the service
  - Use the GUI, or type “sc start dumbservice”, or “net start..”
- Quickly run Process Explorer and look at handle table for sol.exe
  - Notice name of Windowstation object
- Open services.msc; mark service “Allow Service to Interact with Desktop”
- Start the service again and in Process Explorer, look at handle table for sol.exe
  - Notice name of Windowstation object

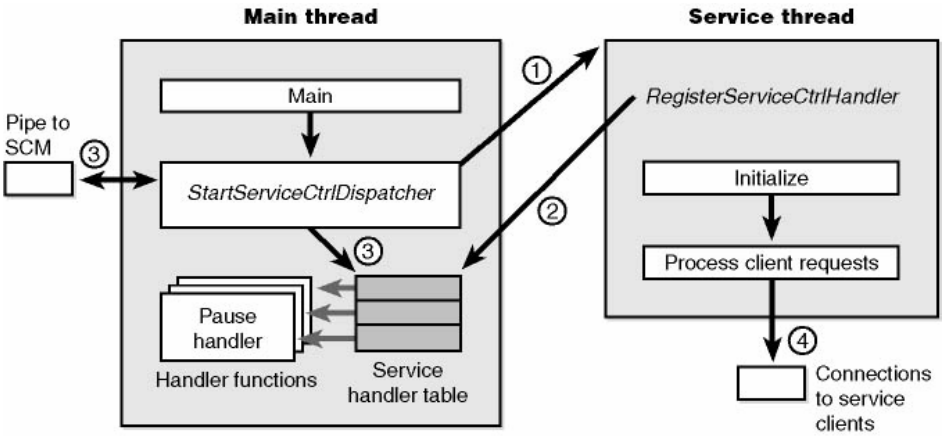
60

SrvAny Tool

If you have a program that you want to run as a service, you need to modify the startup code to conform to the requirements for services outlined in this section. If you don’t have the source code, you can use the SrvAny tool in the Windows resource kits. SrvAny enables you to run any application as a service. It reads the path of the service file that it must load from the Parameters subkey of the service’s registry key.

When SrvAny starts, it notifies the SCM that it is hosting a particular service, and when it receives a start command, it launches the service executable as a child process. The child process receives a copy of the SrvAny process’s access token and a reference to the same window station, so the executable runs within the same security account and with the same interactivity setting as you specified when configuring the SrvAny process. SrvAny services don’t have the share-process Type value, so each application you install as a service with SrvAny runs in a separate process with a different instance of the SrvAny host program.

Service Processes



1. *StartServiceCtrlDispatcher* launches service thread.
2. Service thread registers service handlers.
3. *StartServiceCtrlDispatcher* calls handlers in response to SCM commands.
4. Service thread processes client requests.

61

Services:

How do services interact with the system?

- Must register with service control manager when started (otherwise process is killed)
- Get startup configuration parameters from Registry
- Log errors to Windows 2000 Event Log
- Use some form of IPC mechanism for client communication and control
- Likely make use of Win2K security impersonation

Service implementation

- One .EXE may have >1 service (type code in Registry indicates)

Examples of services installed by default

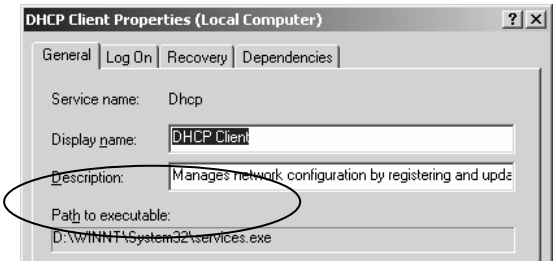
- Event Log, Task Scheduler

Examples of add-on services

- DNS, DHCP, RAS, Web server

Mapping Services to Service Processes

- Not always a 1-to-1 mapping
  - Some service processes contain more than one service
    - Conserves virtual memory, reduces boot time
  - This is up to the developer of the service
- Service properties displayed through Control Panel (services.msc) show name of .EXE
  - But not which process the services is running in



63

EXPERIMENT: Listing Installed Services

To list the installed services, select Administrative Tools from Control Panel, and then select Services. To see the detailed properties about a service, right-click on a service and select Properties. Notice that the Path To Executable field identifies the program that contains this service. Remember that some services share a process with other services—mapping isn't always one to one.

Mapping Services to Service Processes

- Tlist /S (Debugging Tools) or Tasklist /svc (XP/2003) list internal name of services inside service processes
- Process Explorer shows more: external display name and description

64

EXPERIMENT: Viewing Service Details Inside Service Processes

Process Explorer highlights processes hosting one service or more. (You can configure this by selecting the Configure Highlighting entry in the Options menu.) If you double-click on a service-hosting process, you will see a Services tab that lists the services inside the process: the name of the registry key that defines the service, the display name seen by the administrator, and the description text for that service (if present).

Lab: Viewing Service Processes

1. Open a command prompt
2. Type “tasklist /svc”
3. Find the Svchost.exe process with the most services inside it
4. In Process Explorer, double click on that Svchost.exe process
5. Click on Services tab
6. Notice extra details about each service displayed by Process Explorer

EXPERIMENT: Viewing Services Running Inside Processes

The Process Explorer utility that you can download from www.sysinternals.com shows detailed information about the services running with processes. Run Process Explorer and view Services tabs on the process properties dialog box for the following processes: Services.exe, Lsass.exe, and Svchost.exe. Several instances of SvcHost will be running on your system, and you can see the account in which each is running by adding the Username column to the Process Explorer display or by looking at the Username field on the Image tab of a process’s Process Properties dialog box.

The information displayed by Process Explorer includes the service name, display name, and service description, if it has one, which Process Explorer shows beneath the service list when you select a service.

Services Infrastructure

- Windows 2000 introduced generic Svchost.exe
  - Groups services into fewer processes
    - Improves system startup time
    - Conserves system virtual memory
  - Not user-configurable as to which services go in which processes
  - 3rd parties cannot add services to Svchost.exe processes
- Windows XP/2003 have more Svchost processes due to two new less privileged accounts for built-in services
  - LOCAL SERVICE, NETWORK SERVICE
  - Less rights than SYSTEM account
    - Reduces possibility of damage if system compromised
- On XP/2003, four Svchost processes (at least):
  - SYSTEM
  - SYSTEM (2nd instance – for RPC)
  - LOCAL SERVICE
  - NETWORK SERVICE

67

Service Control Tools

Net start/stop – local system only

Sc.exe (built in to XP/2003; also in Win2000 Resource Kit)

Command line interface to all service control/configuration functions

Works on local or remote systems

Psservice (Sysinternals) – similar to SC

Other tools in Resource Kit

Instsrv.exe – install/remove services (command line)

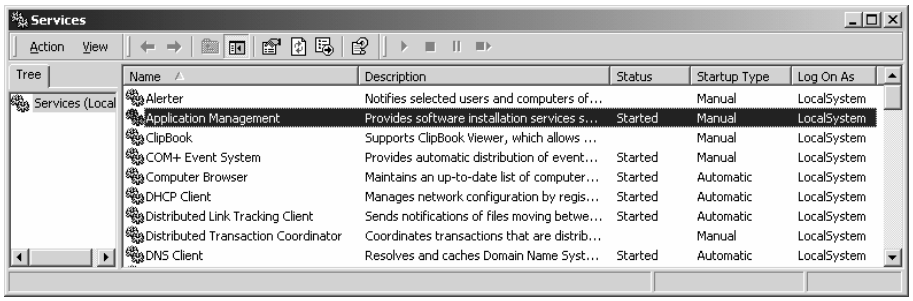
Srvinstw.exe – install/remove services (GUI)

Why are service creation tools included in Reskit?

Because Reskit comes with several services that are not installed as services when you install the Reskit

Service Configuration & Control Tools

- To view & control services:
  - Control Panel->Administrative Tools->Services



- No option to add/remove – done at install/uninstall time

Understanding Svchost.exe CPU Time Consumption

If a multi-service process or other multi-component process such Inetinfo.exe (IIS) or Dllhost.exe (COM) is consuming CPU time, how do you determine which service is responsible?

Need to drill down to thread granularity

Go to Threads tab in Process Explorer and sort by CPU usage