Unit OS5: Memory Management

5.5. Lab Slides & Lab Manual

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

Roadmap for Section 5.5.

- Dynamic Link Library (DLL) Usage
- Process Explorer and loaded DLLs
- Using ProcessWalker to inspect address layout
- Viewing the Working Set
- Inspecting the Page Frame Number Database
- Perfmon and memory-related counters
- Monitoring page file consumption

3

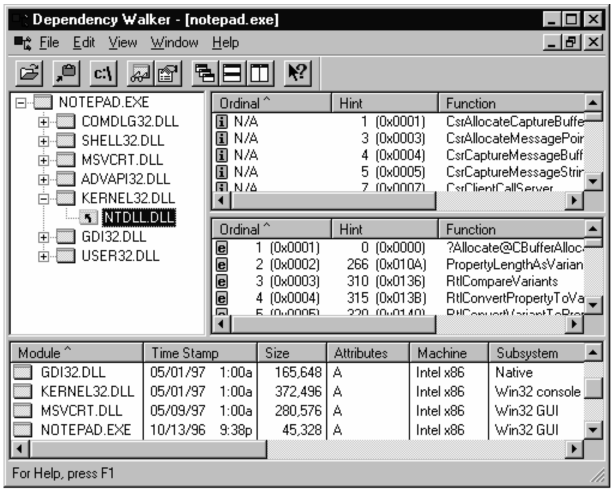
Copyright Notice
© 2000-2005 David A. Solomon and Mark Russinovich

These materials are part of the *Windows Operating System Internals Curriculum Development Kit,* developed by David A. Solomon and Mark E. Russinovich with Andreas Polze

Microsoft has licensed these materials from David Solomon Expert Seminars, Inc. for distribution to academic organizations solely for use in academic environments (and not for commercial use)

DLL Usage: Dependency Walker

- Displays static linkage from EXE to DLLs (doesn't account for dynamically loaded DLLs after process startup)



4

DLL Usage:

To diagnose DLL conflicts, you need to know which DLLs were loaded and from where

- Pviewer & pview & tlist lists the loaded DLLs, but not the path (e.g. type "tlist explorer")
- Dependency Walker can trace DLL loads
- Process Explorer or listdlls from www.sysinternals.com lists full path

```
Command Prompt
C:\NT>listdlls -p outlook

ListDLLs V2.0
Copyright (C) 1997 Mark Russinovich
http://www.ntinternals.com

----------------------------------------------------------------------------
OUTLOOK.EXE pid: 116
  Base        Size       Version          Path
  0x30000000  0xb000     8.05.5104.0000   C:\Program Files\Microsoft Office\Office
\OUTLOOK.EXE
  0x77f60000  0x5c000    4.00.1381.0004   C:\WINNT\System32\ntdll.dll
  0x6e3f0000  0x655000   8.05.5104.0007   C:\Program Files\Microsoft Office\Office
\OUTLLIB.dll
  0x78000000  0x3d000    6.00.8267.0000   C:\WINNT\system32\MSVCRT.dll
  0x77f00000  0x5e000    4.00.1381.0004   C:\WINNT\system32\KERNEL32.dll
  0x77b20000  0xb2000    4.00.1381.0004   C:\WINNT\system32\ole32.dll
  0x77e10000  0x52000    4.00.1381.0004   C:\WINNT\system32\RPCRT4.dll
  0x77dc0000  0x3e000    4.00.1381.0004   C:\WINNT\system32\ADVAPI32.dll
  0x77e70000  0x54000    4.00.1381.0004   C:\WINNT\system32\USER32.dll
  0x77ed0000  0x2c000    4.00.1381.0004   C:\WINNT\system32\GDI32.dll
  0x71030000  0x73000    4.72.3110.0001   C:\WINNT\system32\COMCTL32.dll
  0x306c0000  0x3a2000   8.00.0000.4328   C:\Program Files\Microsoft Office\Office
\MSO97.DLL
  0x6fa90000  0xda000    5.05.2174.0000   C:\WINNT\System32\MAPI32.DLL
```

Lab: Looking at Loaded DLLs

- DLL version mismatches can cause strange application failures
  - Most applications do a poor job of reporting DLL version problems
- To diagnose DLL conflicts, you need to know which DLLs were loaded and from where

1. Start Notepad
2. Type “tlist notepad” -- EXE & DLLs have no path name
3. Type “listdlls notepad” & see full pathname of EXE & DLLs
4. Run Process Explorer and view DLL list

6

Example Problem: Help failed

The Help command in an application failed on Win95, but worked fine on Win98/ME/NT4/Win2000/WinXP

- Failed with meaningless error message

Solution

Ran Filemon on failing system and working system

- Reduced log to file opens
- Compared logs

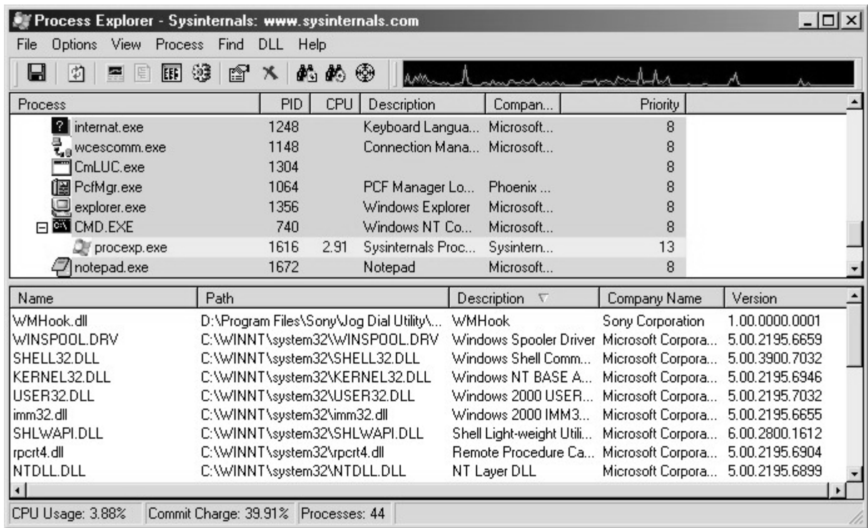
At the point they diverged, looked backwards to last common thing done

- An OLE system DLL was loaded
- Noticed this OLE DLL was loaded from a directory in the user’s PATH on Win95, but from \Windows\System on other versions

Conclusion:

- DLL loaded on Win95 system was not for Win95
- Got proper version for Win95, problem went away

Process Explorer DLL View



7

Click on View->DLL View

- Shows more than just loaded DLLs
- Includes .EXE and any “memory mapped files”

Uses:

- Detect DLL versioning problems
  - Compare the output from a working process with that of a failing one (use File->Save As)
- Find which processes are using a specific DLL (search for it)

Show Relocated DLLs option

- Highlights relocated DLLs in yellow

Process Explorer DLL Lab1: Run Word and Excel

1. In ProcExp, switch to DLL view
2. Look at the DLL list for both Word and Excel and find a common Office DLL loaded in both processes
   - Hint: sort by path
3. Try and delete that DLL with Explorer
   - Should get access denied error (not file locked)
4. In ProcExp, use search to confirm who has this DLL loaded
   - Should show up in both processes

Prefetch Lab

- Lab
  - Run Filemon – set filter as Notepad.exe
  - Make a temporary directory somewhere (e.g. \temp)
  - Run “Notepad \temp\x.y”
  - Exit Notepad
  - Run Notepad again
  - In Filemon log, find creation of .PF file after first run, then use of new .PF in 2nd run

16

EXPERIMENT: Watching Prefetch File Reads and Writes

If you capture a trace of application startup with Filemon from www.sysinternals.com in Windows XP, you can see the prefetcher check for and read the application’s prefetch file (if it exists), and roughly ten seconds after the application started, see the prefetcher write out a new copy of the file. Below is a capture of Notepad startup with an Include filter set to “prefetch” so that Filemon shows only accesses to the \Windows\Prefetch directory:

#	Time	Process	Request	Path	Result	Other
1	4:42:21 PM	notepad.exe:5792	OPEN	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Options: Open Access: All
2	4:42:21 PM	notepad.exe:5792	QUERY INFORMATION	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Length: 12784
3	4:42:21 PM	notepad.exe:5792	READ	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Offset: 0 Length: 12784
4	4:42:31 PM	SVCHOST.EXE:1504	OPEN	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Options: Open Access: All
5	4:42:31 PM	SVCHOST.EXE:1504	QUERY INFORMATION	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Length: 12784
6	4:42:31 PM	SVCHOST.EXE:1504	QUERY INFORMATION	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Length: 12784
7	4:42:31 PM	SVCHOST.EXE:1504	CLOSE	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	
8	4:42:31 PM	SVCHOST.EXE:1504	CREATE	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Options: OverwriteIf Access: All
9	4:42:31 PM	SVCHOST.EXE:1504	WRITE	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	Offset: 0 Length: 12832
10	4:42:31 PM	SVCHOST.EXE:1504	CLOSE	C:\WINDOWS\Prefetch\NOTEPAD.EXE-0275247A.pf	SUCCESS	

Lines 1 through 3 show the Notepad prefetch file being read in the context of the Notepad process during its startup. Lines 4 through 10, which have time stamps 10 seconds later than the first 3 lines, show the Task Scheduler, which is running in the context of a Svchost process, write out the updated prefetch file.

Viewing the Working Set

- Working set size counts shared pages in each working set
- Vadump (Resource Kit) can dump the breakdown of private, shareable, and shared pages

```
C:\> Vadump –o –p 3968
Module Working Set Contributions in pages
    Total   Private Shareable    Shared Module
       14         3        11         0 NOTEPAD.EXE
       46         3         0        43 ntdll.dll
       36         1         0        35 kernel32.dll
        7         2         0         5 comdlg32.dll
       17         2         0        15 SHLWAPI.dll
       44         4         0        40 msvcrt.dll
```

17

EXPERIMENT: Viewing Process Working Set Sizes

You can use the Performance tool to examine process working set sizes by looking at the following performance counters:

Counter	Description
Process: Working Set	Current size of the selected process's working set in bytes
Process: Working Set Peak	Peak size of the selected process's working set in bytes
Process: Page Faults/Sec	Number of page faults for the process that occur each second

Several other process viewer utilities (such as Task Manager, Pview, and Pviewer) also display the process working set size.

You can also get the total of all the process working sets by selecting the _Total process in the instance box in the Performance tool. This process isn't real—it's simply a total of the process-specific counters for all processes currently running on the system. The total you see is misleading, however, because the size of each process working set includes pages being shared by other processes. Thus, if two or more processes share a page, the page is counted in each process's working set.

Process Memory Information

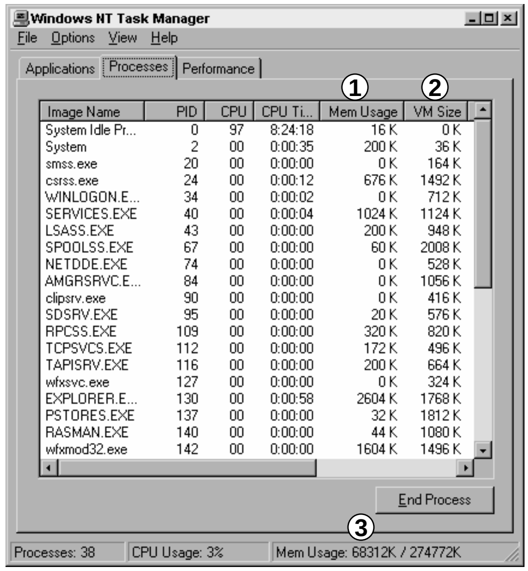
Task Manager Processes tab

① **"Mem Usage" = physical memory used by process (working set <u>size</u>, not working set limit)**

- ◆ **Note: shared pages are counted in each process**

② **"VM Size" = private (not shared) committed virtual space in processes == process's paging file allocation**

③ **"Mem Usage" in status bar is <u>not</u> total of "Mem Usage" column (see later slide)**



Screen snapshot from:
Task Manager | Processes tab

18

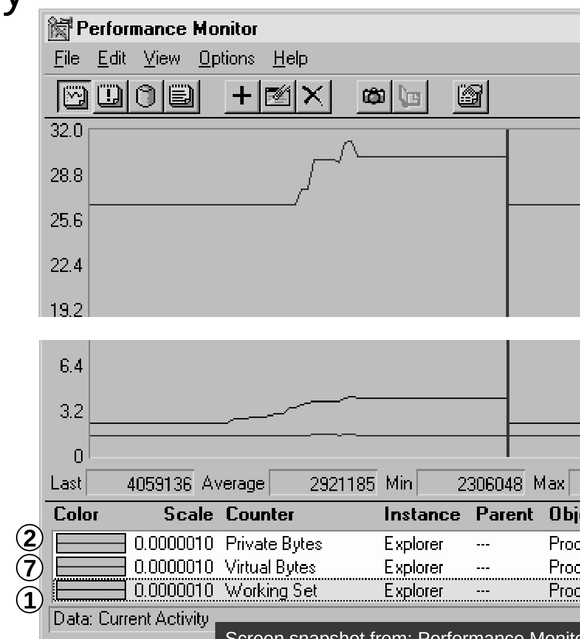
Configuring the Memory Manager

Like most of Windows, the memory manager attempts to automatically provide optimal system performance for varying workloads on systems of varying sizes and types. While there are a limited number of registry values you can add and/or modify under the key HKLM\ SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management to override some of these default performance calculations, in general, the memory manager's default computations will be sufficient for the majority of workloads.

Many of the thresholds and limits that control memory manager policy decisions are computed at system boot time on the basis of memory size and product type. (Windows 2000 Professional and Windows XP Professional and Home editions are optimized for desktop interactive use, and Windows Server systems are optimized for running server applications.)

Process Memory Information
PerfMon - Process Object

- ⑦ "Virtual Bytes" = committed + reserved virtual space, including shared pages
- ① "Working Set" = working set size (not limit) (physical)
- ② "Private Bytes" = private virtual space (same as "VM Size" from Task Manager Processes list)
- Also: In Threads object, look for threads in Transition state - evidence of swapping (usually caused by severe memory pressure)



Screen snapshot from: Performance Monitor counters from Process object

EXPERIMENT: Accounting for Physical Memory Use

By combining information available from performance counters with output from kernel debugger commands, you can come close to accounting for physical memory usage on a machine running Windows. To examine the memory usage information available through performance counters, run the Performance tool and add the counters to view the following information.

Total process working set size

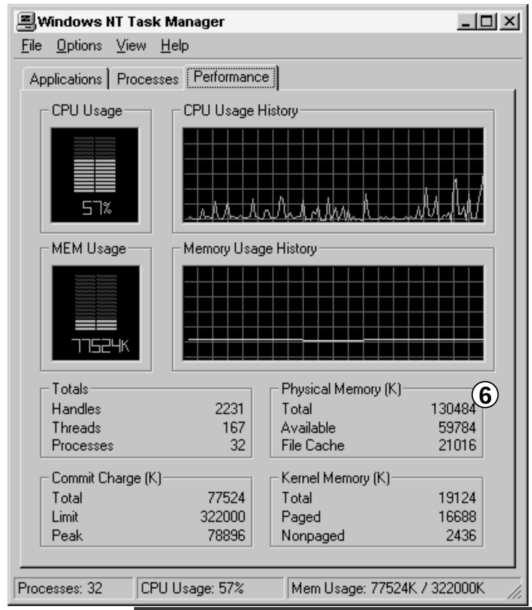
To view this information, select the Process performance object and the Working Set counter for the _Total process instance. This number will be larger than the actual total process memory utilization because shared pages are counted in each process working set. To get a more accurate picture of process memory utilization, subtract free memory (available bytes), operating system memory used (nonpaged pool, resident paged pool, and resident operating system and driver code), and the size of the modified list from the total physical memory on the machine. What you're left with is the memory being used by processes. Comparing this value against the total process working set size as reported by the Performance tool gives you some indication of the amount of sharing occurring between processes. Although examining process physical memory usage is interesting, of more concern is the private committed virtual memory usage by processes, because memory leaks show up as an increasing private virtual size, not an increasing working set size.

Total system working set size

To view this information, select the Memory processor object and the Cache Bytes counter. As explained in the section "System Working Set," the total system working set size includes more than just the cache size— it includes the subset of paged pool, pageable operating system code, and pageable driver code that is resident and in the system working set.

Memory Management Information

Task Manager Performance tab



Screen snapshot from:
Task Manager | Performance tab

- ⑥ "Available" = sum of free, standby, and zero page lists (physical)
  - Majority are likely standby pages
  - Windows 2000/XP/Server 2003: count of shareable pages on standby, modified, and modified nowrite list are included in what was "File Cache" in NT4
    - New name is "System Cache"

EXPERIMENT: Viewing System Memory Information

The Performance tab in the Windows Task Manager displays basic system memory information. This information is a subset of the detailed memory information available through the performance counters.

Both Pmon.exe (in the Windows Support Tools) and Pstat.exe (in the Platform SDK) display system and process memory information.

Finally, the !vm command in the kernel debugger shows the basic memory management information available through the memory-related performance counters. This command can be useful if you're looking at a crash dump or hung system. Here's an example of its output:

```
kd> !vm
    *** VirtualMemory Usage ***
        PhysicalMemory: 32620 ( 130480Kb)
        PageFile: \??\C:\pagefile.sys
            Current: 204800Kb Free Space: 101052Kb
            Minimum: 204800Kb Maximum: 204800Kb
        Available Pages: 3604 ( 14416Kb)
        ResAvailPages: 24004 ( 96016Kb)
        ModifiedPages: 768 ( 3072Kb)
        NonPagedPoolUsage: 1436 ( 5744Kb)
        NonPagedPoolMax: 12940 ( 51760Kb)
        PagedPool 0Usage: 6817 ( 27268Kb)
        PagedPool 1Usage: 982 ( 3928Kb)
        PagedPool 2Usage: 984 ( 3936Kb)
        PagedPool Usage: 8783 ( 35132Kb)
        PagedPool Maximum: 26624 ( 106496Kb)
        …..
```

PFN Database

- PFN = Page Frame Number
  = Physical Page Number
- PFN Database keeps track of the state of each physical page
  - An array of structures, one element per physical page
  - Maintains reference and share counts for pages in working sets
  - Structure elements implement forward and backward links for free, modified, standby, zero, and bad page lists
  - Does not reflect memory not managed by NT (e.g. adapter ram)

```
kd> !pte ff709348
!pte ff709348
FF709348  - PDE at C0300FF4    PTE at C03FDC24
          contains 00410063  contains 0049E063
          pfn 00410 DA--KWV  pfn 0049E DA--KWV
kd> !pfn 410
!pfn 410
    PFN address FFBCC180
    flink       00000000  blink / share count 000000B0  pteaddress C0300FF4
    reference count 0001                                 color 0
    restore pte 00000000  containing page        00030  Active
```

Screen snapshot from: kernel debugger !pte command
use resulting displayed PFN on !pfn command

21

EXPERIMENT: Viewing PFN Entries

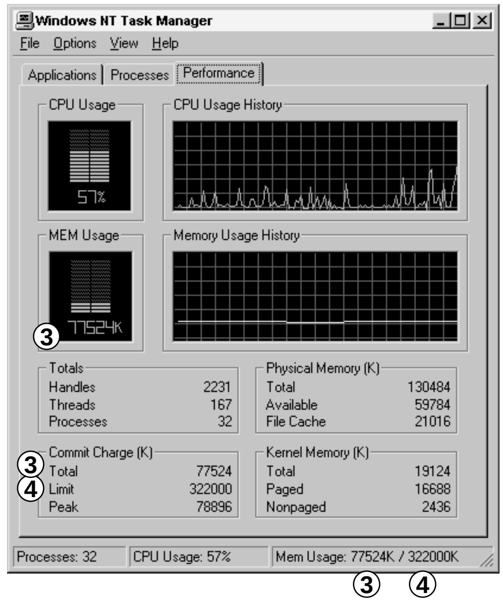
You can examine individual PFN entries with the kernel debugger !pfn command. You first need to supply the PFN as an argument. (For example, !pfn 0 shows the first entry, !pfn 1 shows the second, and so on.) In the following example, the PTE for virtual address 0x50000 is displayed, followed by the PFN that contains the page directory, and then the actual page:

```
kd> !pte 50000
00050000 - PDE at C0300000          PTE at C0000140
            contains 00700067       contains 00DAA047
            pfn00700 --DA--UWV      pfn00DAA--D---UWV

kd> !pfn700
    PFN00000700 at address 827CD800
    flink 00000004 blink/ share count00000010 pteaddress C0300000
    reference count 0001                        color 0
    restore pte 00000080 containing page 00030 Active M
    Modified

kd> !pfn daa
    PFN00000DAA at address 827D77F0
```

Memory Management Information

Task Manager Performance tab

- ③ Total committed private virtual memory (total of “VM Size” in process tab + Kernel Memory Paged)
  - not all of this space has actually been used in the paging files; it is “how much would be used if it was all paged out”
  - “Commit charge limit” = sum of physical memory available for processes + current total size of paging file(s)
- ④ does not reflect true maximum page file sizes (expansion)
  - when “total” reaches “limit”, further VirtualAlloc attempts by any process will fail



23

EXPERIMENT: Viewing System Page Files

To view the list of page files, look in the registry at HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PagingFiles. This contains the paging file configuration settings modified through the System utility in Control Panel. In Windows 2000, click the Performance Options button on the Advanced tab, and then click the Change button. In Windows XP and Windows Server 2003, click the Advanced tab, click the Settings button in the Performance section, click the Advanced tab, and finally, click the Change button in the Virtual Memory section.

Lab: Memory Leaks

- Run Leakyapp.exe (Resource Kit)
- In Task Manager Process tab, watch Mem Usage & VM Size grow (also look at Performance tab Commit limit/peak)
  - Mem Usage will eventually reach an upper limit
  - VM Size will grow until no more page file space

Low and High Memory Notification

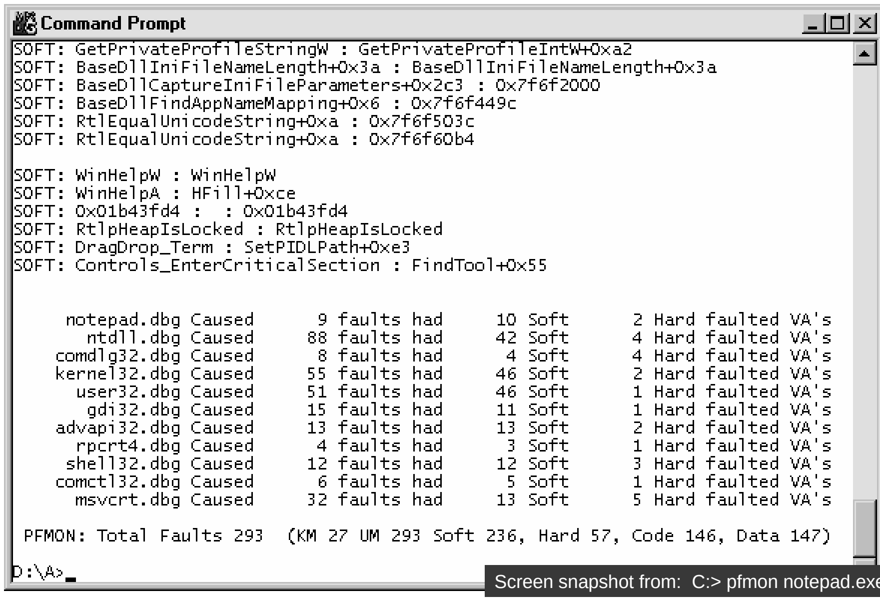
Windows XP and Windows Server 2003 provide a way for user mode processes to be notified when physical memory is low and/or plentiful. This information can be used to determine memory usage as appropriate. For example, if available memory is low, the application can reduce memory consumption. If available memory is high, the application can allocate more memory.

To be notified of low or high memory conditions, call the *CreateMemoryResourceNotification* function, specifying whether low or high memory notification is desired. A handle can be provided to any of the wait functions. When memory is low (or high), the wait completes, thus notifying the thread of the condition. Alternatively, the *QueryMemoryResourceNotification* can be used to query the system memory condition at any time.

Notification is implemented by the memory manager signaling a globally named event object *LowMemoryCondition* or *HighMemoryCondition*. These named objects are not in the normal *BaseNamedObjects* object manager directory, but in a special directory called *KernelObjects*. When low (or high) memory condition is detected, the appropriate event is signaled, thus waking up any waiting threads.

The default level of available memory that signals a low-memory-resource notification event is approximately 32 MB per 4 GB, to a maximum of 64 MB. The default level that signals a highmemory-resource notification event is three times the default low-memory value. These values can be overridden by adding a DWORD registry value *LowMemoryThreshold* or *HighMemoryThreshold* under HKLM\System\CurrentControlSet\Session Manager\Memory Management that specifies the number of megabytes to use as the low or high threshold.

Page Fault Monitor (pfmon)



```
Command Prompt
SOFT: GetPrivateProfileStringW : GetPrivateProfileIntW+0xa2
SOFT: BaseDllIniFileNameLength+0x3a : BaseDllIniFileNameLength+0x3a
SOFT: BaseDllCaptureIniFileParameters+0x2c3 : 0x7f6f2000
SOFT: BaseDllFindAppNameMapping+0x6 : 0x7f6f449c
SOFT: RtlEqualUnicodeString+0xa : 0x7f6f503c
SOFT: RtlEqualUnicodeString+0xa : 0x7f6f60b4

SOFT: WinHelpW : WinHelpW
SOFT: WinHelpA : HFill+0xce
SOFT: 0x01b43fd4 :  : 0x01b43fd4
SOFT: RtlpHeapIsLocked : RtlpHeapIsLocked
SOFT: DragDrop_Term : SetPIDLPath+0xe3
SOFT: Controls_EnterCriticalSection : FindTool+0x55


    notepad.dbg Caused     9 faults had    10 Soft     2 Hard faulted VA's
      ntdll.dbg Caused    88 faults had    42 Soft     4 Hard faulted VA's
   comdlg32.dbg Caused     8 faults had     4 Soft     4 Hard faulted VA's
   kernel32.dbg Caused    55 faults had    46 Soft     2 Hard faulted VA's
     user32.dbg Caused    51 faults had    46 Soft     1 Hard faulted VA's
      gdi32.dbg Caused    15 faults had    11 Soft     1 Hard faulted VA's
   advapi32.dbg Caused    13 faults had    13 Soft     2 Hard faulted VA's
     rpcrt4.dbg Caused     4 faults had     3 Soft     1 Hard faulted VA's
    shell32.dbg Caused    12 faults had    12 Soft     3 Hard faulted VA's
   comctl32.dbg Caused     6 faults had     5 Soft     1 Hard faulted VA's
     msvcrt.dbg Caused    32 faults had    13 Soft     5 Hard faulted VA's

 PFMON: Total Faults 293  (KM 27 UM 293 Soft 236, Hard 57, Code 146, Data 147)

D:\A>_
```

Screen snapshot from: C:> pfmon notepad.exe

EXPERIMENT: Viewing Page Fault Behavior

With the Pfmon tool (in the Windows 2000 and 2003 resource kits, as well as in the Windows XP Support Tools), you can watch page fault behavior as it occurs. A soft fault refers to a page fault satisfied from one of the transition lists. Hard faults refer to a diskread. The following example is a portion of output you'll see if you start Notepad with Pfmon and then exit. Be sure to notice the summary of page fault activity at the end.

```
C:\>pfmon notepad
SOFT:KiUserApcDispatcher :KiUserApcDispatcher
SOFT:LdrInitializeThunk :LdrInitializeThunk  SOFT:0x77f61016: : 0x77f61016
SOFT:0x77f6105b: : fltused+0xe00  HARD:0x77f6105b: : fltused+0xe00
SOFT:LdrQueryImageFileExecutionOptions :  LdrQueryImageFileExecutionOptions
SOFT:RtlAppendUnicodeToString: RtlAppendUnicodeToString
SOFT:RtlInitUnicodeString: RtlInitUnicodeString

notepad     Caused 8faultshad 9 Soft 5 Hardfaulted VA’s
ntdll       Caused 94faultshad 42 Soft 8 Hardfaulted VA’s
comdlg32    Caused 3faultshad 0 Soft 3 Hardfaulted VA’s
shlwapi     Caused 2faultshad 2 Soft 2 Hardfaulted VA’s
gdi32       Caused 18faultshad 10 Soft 2 Hardfaulted VA’s
kernel32    Caused 48faultshad 36 Soft 3 Hardfaulted VA’s
user32      Caused 38faultshad 26 Soft 6 Hardfaulted VA’s
advapi32    Caused 7faultshad 6 Soft 3 Hardfaulted VA’s
rpcrt4      Caused 6faultshad 4 Soft 2 Hardfaulted VA’s
comctl32    Caused 6faultshad 5 Soft 2 Hardfaulted VA’s
shell32     Caused 6faultshad 5 Soft 2 Hardfaulted VA’s
            Caused 10faultshad 9 Soft 5 Hardfaulted VA’s
winspool    Caused 4faultshad 2 Soft 2 Hardfaulted VA’s
PFMON: Total Faults250   (KM 74 UM250Soft 204,Hard 46, Code121, Data129)
```