Unit 4: Memory Management

4.2. Windows 2000 Memory Management Internals

Windows 2000
Memory Management Internals

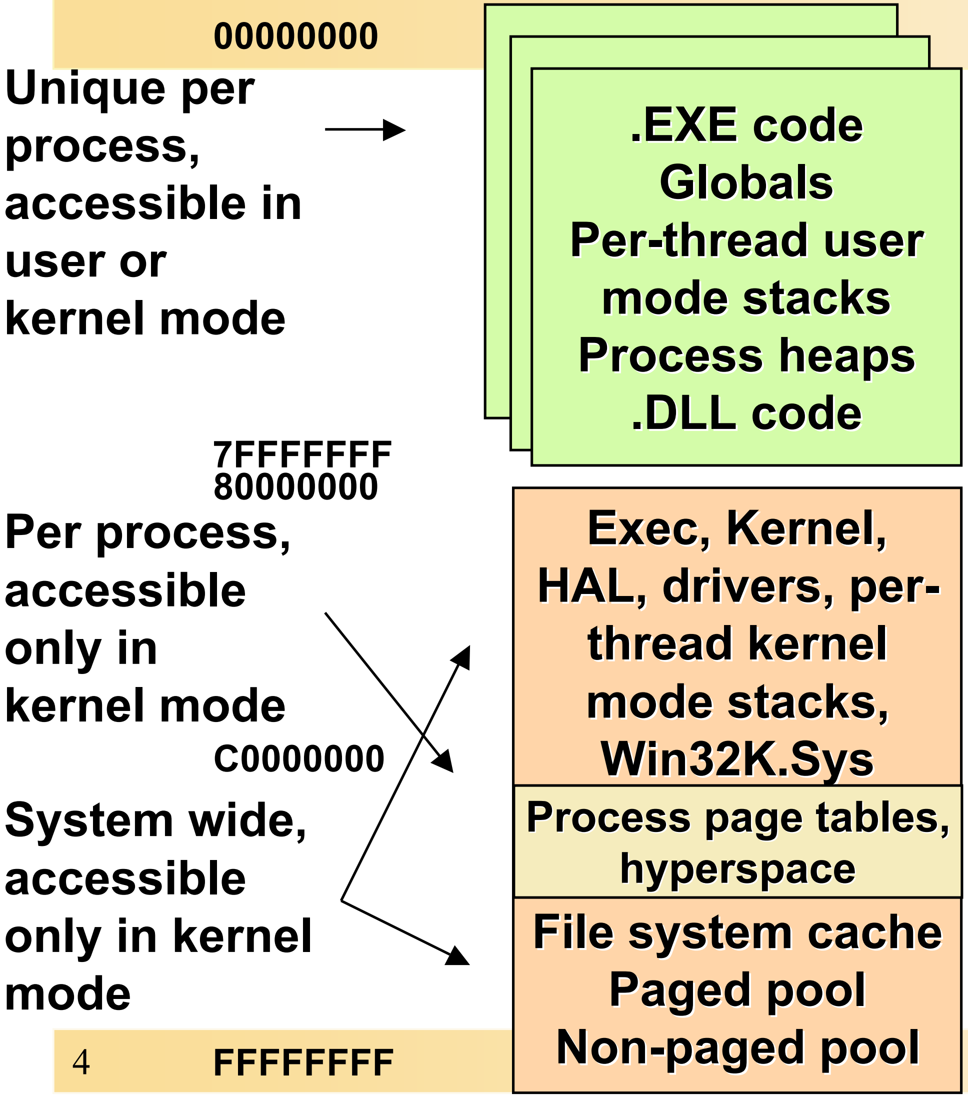
Agenda:

- Introduction
- Process Memory
- Free Memory
- System Memory

Windows 2000 Memory Manager

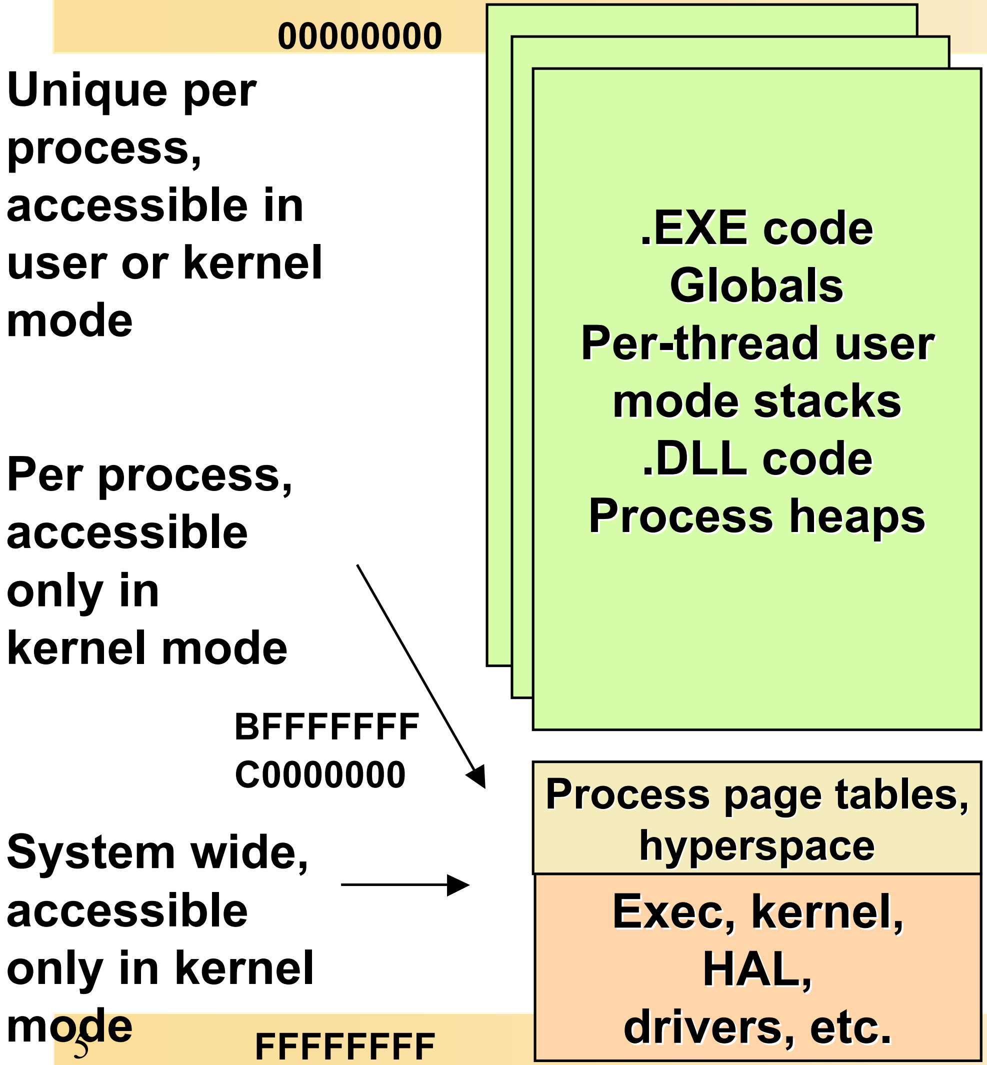
- Provides 4 GB flat virtual address space (32-bit addresses)
- Exports memory-mapped files
- Allows pages shared between processes
- Provides support for file system cache manager
- Windows 2000 enhancements:
  - Integrated support for Terminal Server
  - Ability to use up to 64 GB physical memory
  - Performance and scalability improvements
  - Driver verifier

4GB Virtual Address Space



- 2 GB per-process
  - Address space of one process is not directly reachable from other processes
- 2 GB systemwide
  - The operating system is loaded here, and appears in every process's address space
  - There is no process for "the operating system" (though there are processes that do things for the OS, more or less in "background")

3GB Process Space Option



- Only available on x86 Windows 2000 Advanced Server
  - Boot with /3GB option in BOOT.INI
  - Chief “loser” in system space is file system cache
- Expands per-process address space
  - But image must be marked as “large address space aware”
- 16GB maximum physical memory
- A stopgap while we wait for 64-bit Windows 2000 (Itanium)

Physical Memory

- Maximum on Windows NT 4.0 is 4 GB
- Maximum on Windows 2000 is 64 GB
  - Alpha: 32 GB
  - x86:

x86:	Professional, Server:	4 GB
	Advanced Server:	8 GB
	Datacenter:	64 GB

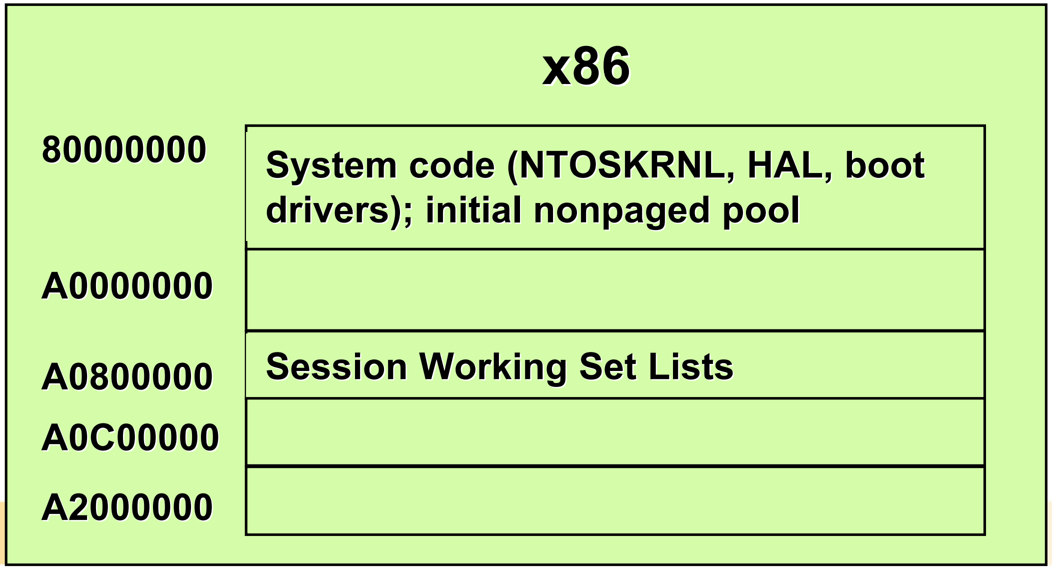
    - Obsoletes PSE driver from Intel that allowed x86 systems with > 4GB to use additional memory as RAM disk
- Virtual address space is still 4 GB,
  so how can you "use" > 4 GB of memory?
  - Mapped (cached) files can remain in physical memory
  - New extended addressing services allow Win32 processes to allocate physical memory and map views or "windows" into 2GB process virtual address space
  - Alpha only: New "very large memory" (VLM) APIs allow Win32 process to allocate up to 28 GB
    - No views necessary, but requires dealing with 64-bit pointers

Address Windowing Extension

- General solution to providing access to large amounts of physical memory
  - Platform independent
- Applications allocate physical memory
  - Then map views of physical memory into their virtual address space (can do I/Os to it)
  - See new Win32 functions AllocateUserPhysicalPages, MapUserPhysicalPages (very fast - 4us)
- Look for server applications to take advantage of this

Sessions

- New memory management object to support Windows 2000 Advanced Server
- All processes in an interactive session share a:
  - Session-specific copy of Win32K.Sys and display drivers
  - Instance of Winlogon and CSRSS
  - Session working set



Agenda

- Introduction
- Process Memory
- Free Memory
- System Memory

All* Committed Virtual Address Space is Mapped To Files

***almost**

- Ranges of virtual address space are mapped to ranges of blocks within disk files
  - These files are the “backing store” for virtual address space
- Commonly-used files are:
  - The system paging file
    - For writeable, nonshareable pages
  - For read-only application-defined code and for shareable data
    - Executable program or DLL
- Can set up additional file/virtual address space relationships at run time (CreateFileMapping API)

Virtual View Of A Process

Process Walker - notepad.exe

Process Sort View Options

Address	State	Prot	Size	BaseAddr	Object	Section	Name
00741000	Free	NA	61440	00000000			
00750000	Commit	RW	4096	00750000			
00751000	Free	NA	61440	00000000			
00760000	Commit	RW	4096	00760000			
00761000	Reserve	NA	126976	00760000			
00780000	Commit	RW	8192	00780000			
00782000	Reserve	NA	57344	00780000			
00790000	Commit	RW	65536	00790000			
007A0000	Reserve	NA	4128768	00790000			
00B90000	Free	NA	16449536	00000000			
01B40000	Commit	RO	4096	01B40000	exe		IMAGE_EXPOR
01B41000	Commit	NA	20480	01B40000	exe	.text	IMAGE_EXPOR
01B46000	Commit	RO	8192	01B40000	exe	.rdata	IMAGE_EXPOR
01B48000	Commit	NA	8192	01B40000	exe	.data	IMAGE_EXPOR
01B4A000	Commit	RO	16384	01B40000	exe	.rsrc	IMAGE_EXPOR
01B4E000	Free	NA	1978277888	00000000			
779F0000	Commit	RO	4096	779F0000	dll		MSVCRT.dll
779F1000	Commit	NA	212992	779F0000	dll	.text	MSVCRT.dll
77A25000	Commit	RO	24576	779F0000	dll	.rdata	MSVCRT.dll
77A2B000	Commit	RW	20480	779F0000	dll	.data	MSVCRT.dll
77A30000	Commit	NA	4096	779F0000	dll		MSVCRT.dll
77A31000	Commit	RO	20480	779F0000	dll	.idata	MSVCRT.dll
77A36000	Free	NA	1810432	00000000			

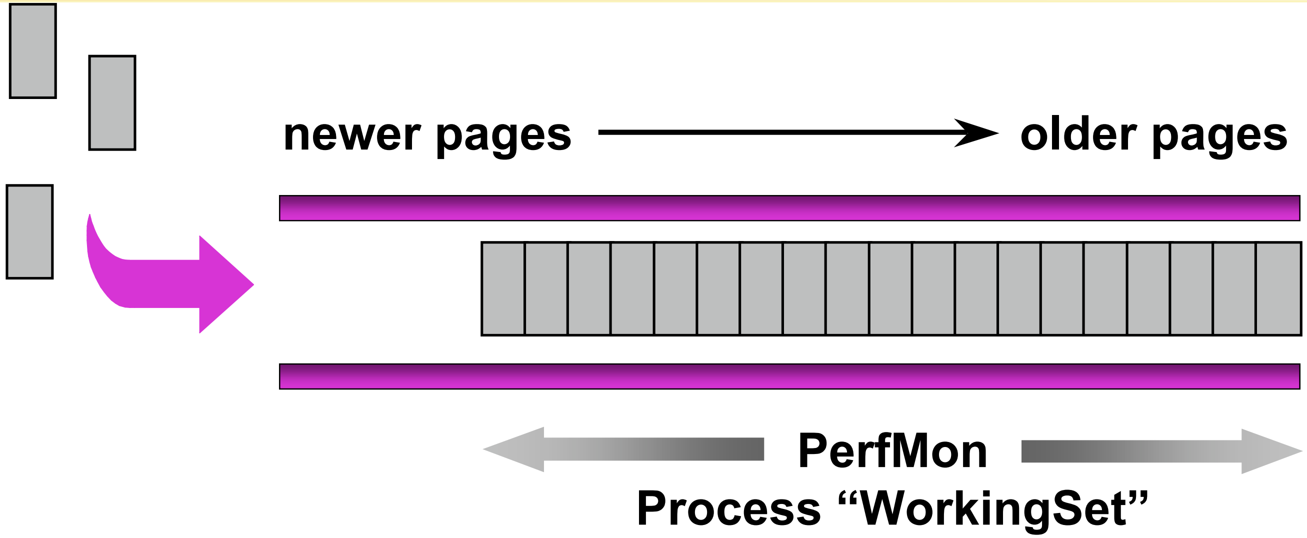
Rewalk

Screen snapshot from:
Programs | SDK Tools | Process Walker
Process | Load Process | notepad

Working Set

- Working set: The subset of the virtual address space in physical memory
  - Essentially, all the pages the process can reference without incurring a page fault
  - Upper limit on size for each process
  - When limit is reached, a page must be released for every page that's brought in ("working set replacement")
- Working set limit: The maximum pages the process can own
  - Default value for new processes
  - System-wide maximum computed at boot time (see MmMaximumWorkingSetSize)

Working Set List
A FIFO list for each process

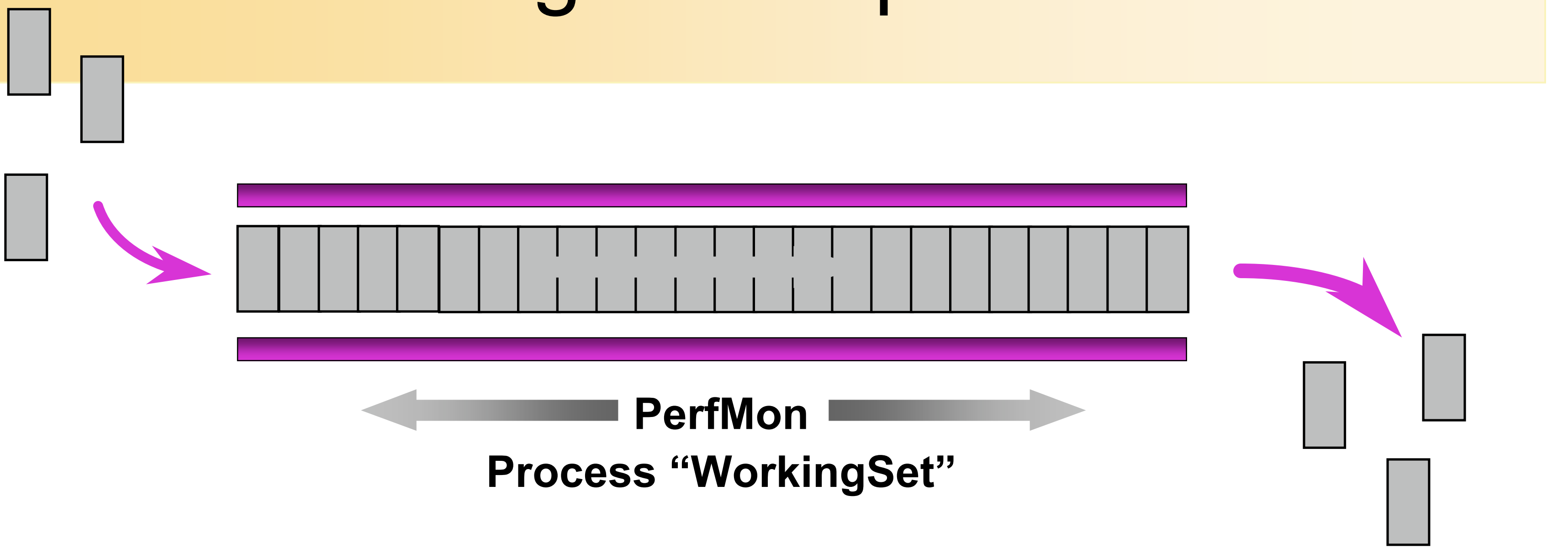


- A process always starts with an empty working set
  - Pages itself into existence
  - Many page faults may be resolved from memory (to be described later)

Soft Versus Hard Page Faults

- Hard page faults involve a disk read
  - Some hard page faults are unavoidable
    - Code is brought into physical memory (from .EXEs and .DLLs) via page faults
    - The file system cache reads data from cached files in response to page faults
- Soft page faults are satisfied in memory
  - A shared page that's valid for one process can be faulted into other processes
  - Pages can be faulted back into a process from the standby and modified page list (described later)
- Performance counters:
  - "Page faults/sec" versus "page reads/sec"
  - "Demand zero" faults/second
  - See chapter "Detecting Memory Bottlenecks" in Windows NT 4.0 Workstation Resource Guide

Working Set Replacement



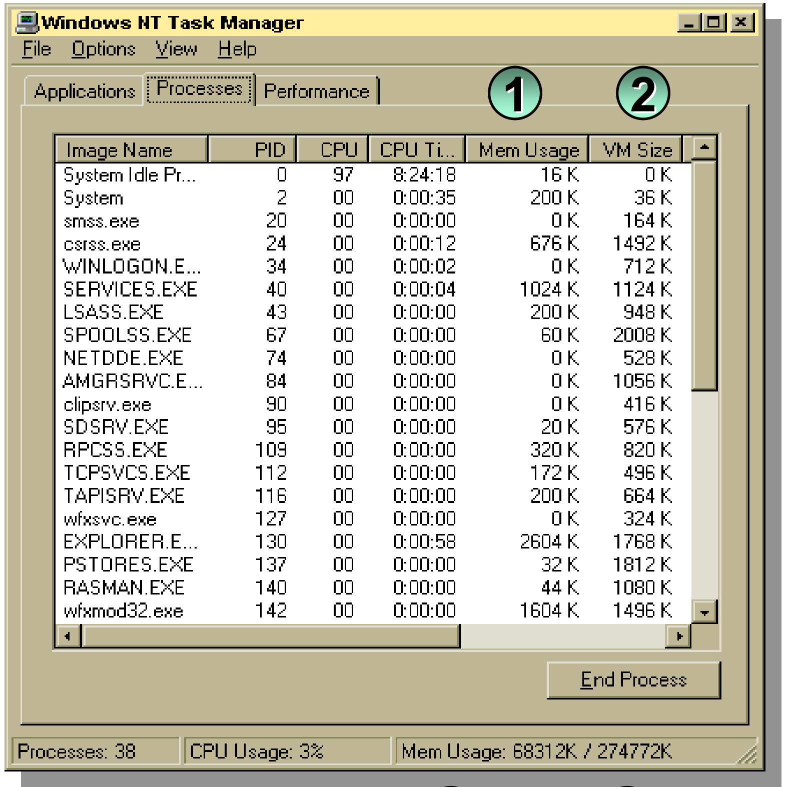
- When working set "count" = working set size, must give up pages to make room for new pages
- Page replacement is "modified FIFO"
  - Windows 2000 on uniprocessor x86 implements "least recently accessed"

Balance Set Manager

- Nearest thing Windows 2000 has to a “swapper”
  - Balance set = sum of all inswapped working sets
- Balance Set Manager is a system thread
  - Wakes up every second. If paging activity high or memory needed:
    - Trims working sets of processes
    - If thread in a long user-mode wait, marks kernel stack pages as pageable
    - If process has no nonpageable kernel stacks, “outswaps” process
    - Triggers a separate thread to do the “outswap” by gradually reducing target process’s working set limit to zero
- Evidence: Look for threads in “Transition” state in PerfMon
  - Means that kernel stack has been paged out, and thread is waiting for memory to be allocated so it can be paged back in
- This thread also performs a scheduling-related function
  - Priority inversion avoidance

Memory Management Information
Task manager processes tab

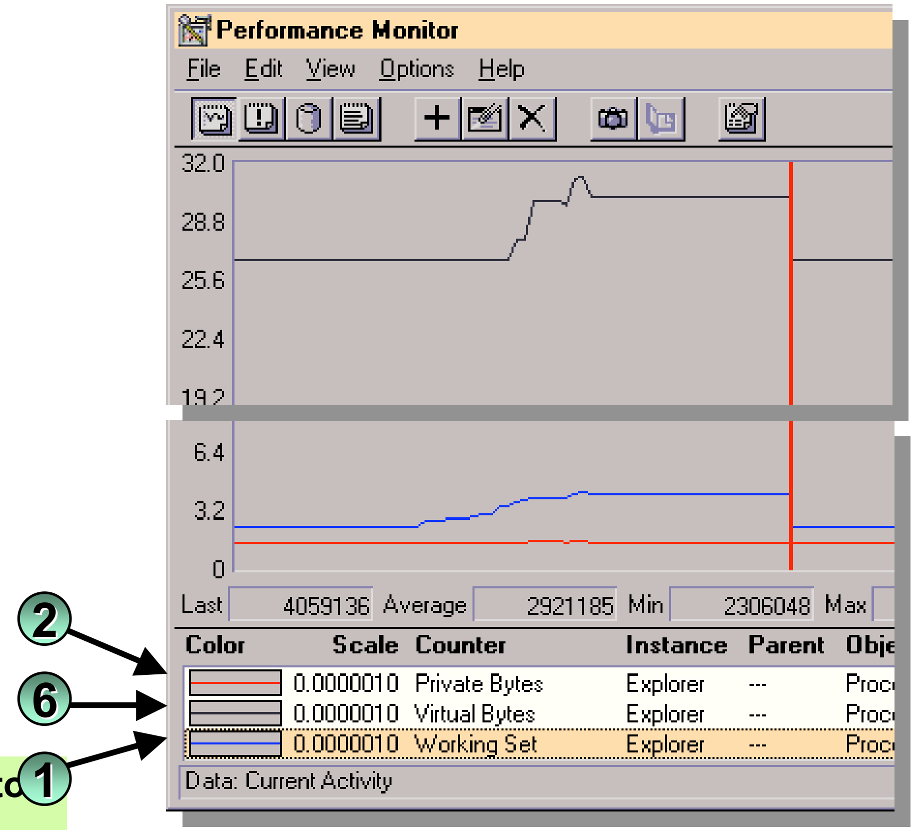
- 1 "Mem Usage" = physical memory used by process (working set size, not working set limit)
- 2 "VM Size" = private (not shared) committed virtual space in processes
- 3 4 "Mem Usage" in status bar is same as "commit charge/commit limit" in "Performance" tab (see next slide) - not same as "Mem Usage" column here!



Screen snapshot from: Task Manager | Processes tab 3 4

Memory Management Information
PerfMon - process object

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



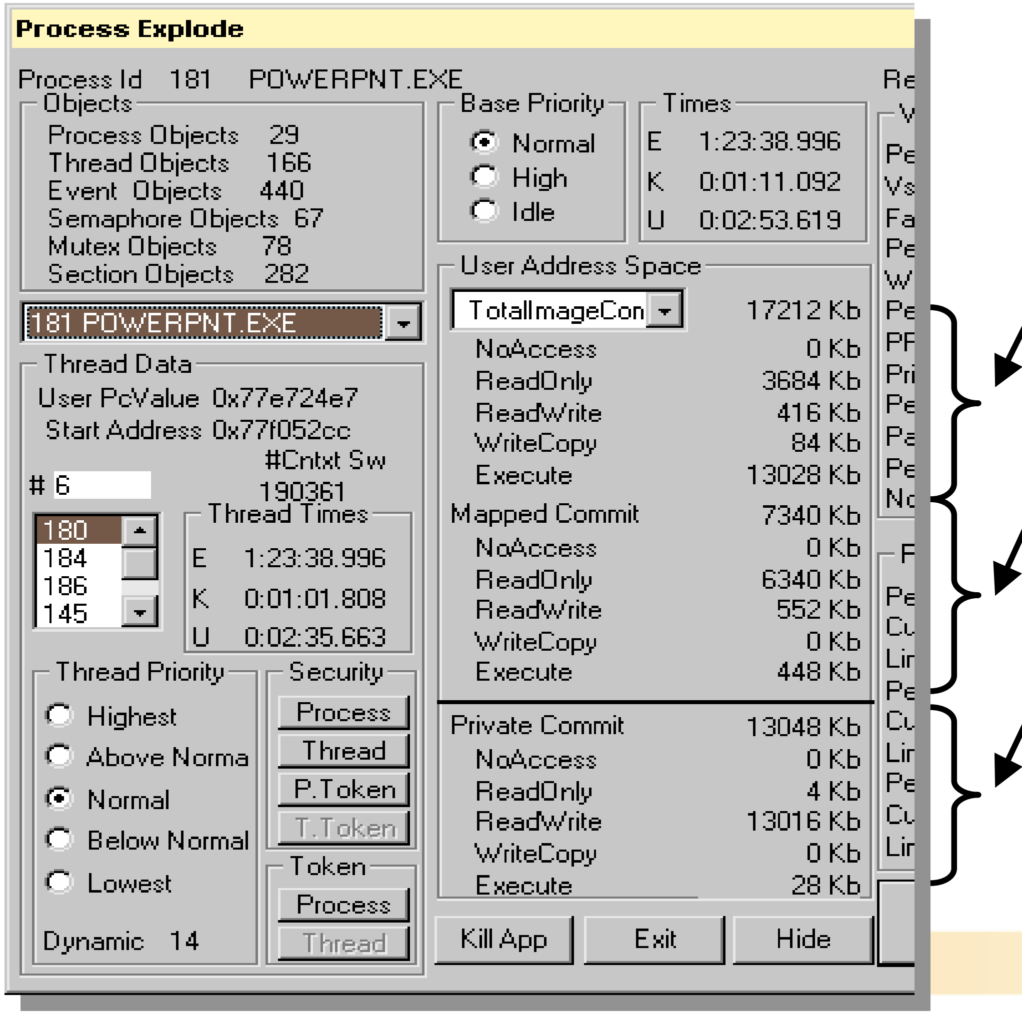
Screen snapshot from: Performance Monitor counters from Process object

Process Memory Used

- To get total of all process working sets:
  - In Perfmon, look at “working set size” of “_Total” process (not a real process)
- This will be higher than actual, because shared pages are counted in
  each process
- To get exact total:
  - Process memory really used =
    Total physical memory - OS memory
    used - Available (free) memory
  - (see end of presentation)

Memory information for a process

Resource Kit pview.exe



Virtual sizes of committed sections of image and DLLs or total of all (total selected)

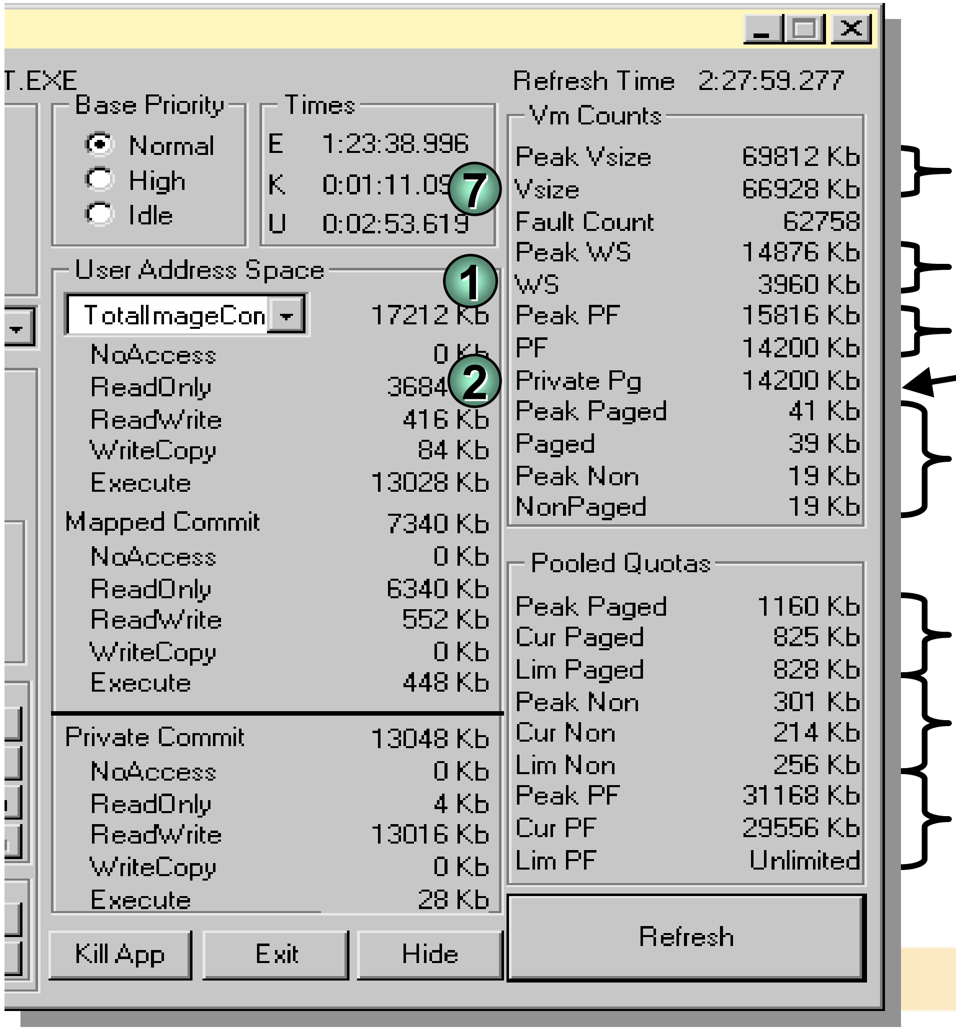
Virtual sizes of sections mapped after image startup (including DLLs loaded with LoadLibrary)

Process-private committed virtual address space (i.e. paging file allocation)

note, “writecopy” = “writeable, but not written to yet”. Windows NT has yet to create process-private pages for these; they are still shared; they become “private commit” when written to

Some, but not all, of this info is also shown by Process Viewer’s “memory detail” button

Memory information for a process
Resource Kit pview.exe



Total virtual address space (committed PLUS reserved, private and shared)

WS = working set (physical)

PF = paging file space allocated (not necessarily written to!)

Same as PerfMon "private bytes", TaskMan "VM size"

Systemwide paged pool (virtual) and nonpaged pool used by this process

Systemwide paged pool

Systemwide nonpaged pool

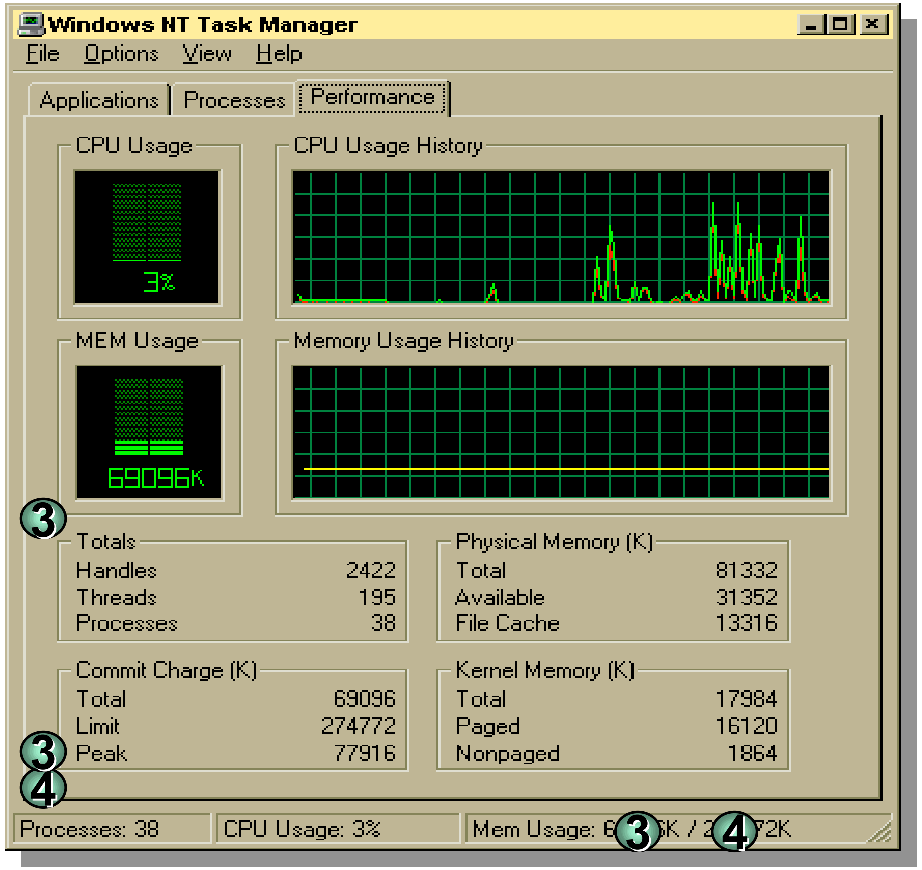
Paging file space allocated by all processes + OS

Note, "limits" in the last three groups are per-process limits; i.e., how much each process can use of these

Memory Management Information
Task manager performance tab

- “Commit charge total” =
  ③ total of private (not shared) committed virtual space in all processes; i.e., total of “VM Size” from processes display, + Kernel Memory paged

④ “Commit charge limit” = sum of available physical memory for processes + free space in paging file



Screen snapshot from: Task Manager | Performance tab

Page Files

- Contiguous page files help!
  - Will be contiguous when created if space available
  - Or, can defrag with full Diskeeper or “CONTIG” (www.sysinternals.com)
- Size depends on virtual memory requirements of applications and drivers
  - Min size should be “max” of normal VM usage
    - Hard disk space is cheap
    - Thus no pagefile fragmentation
  - Max size could be much larger if infrequent demands for large amounts of pagefile space
    - Pagefile extension is deleted on reboot, thus returning to a contiguous pagefile

Page Files

- When page file space runs low
  - 1. “System running low on virtual memory”
    - First time: Before pagefile expansion
    - Second time: When committed bytes reaching commit limit
  - 2. “System out of virtual memory”
    - Page files are full
- Look for who is consuming pagefile space:
  - Process memory leak: Check VM Size (Perfmon “private bytes”)
  - Paged pool leak: Check paged pool size
    - Run poolmon to see what object(s) are filling pool
    - Could be a result of processes not closing handles - check process “handle count”

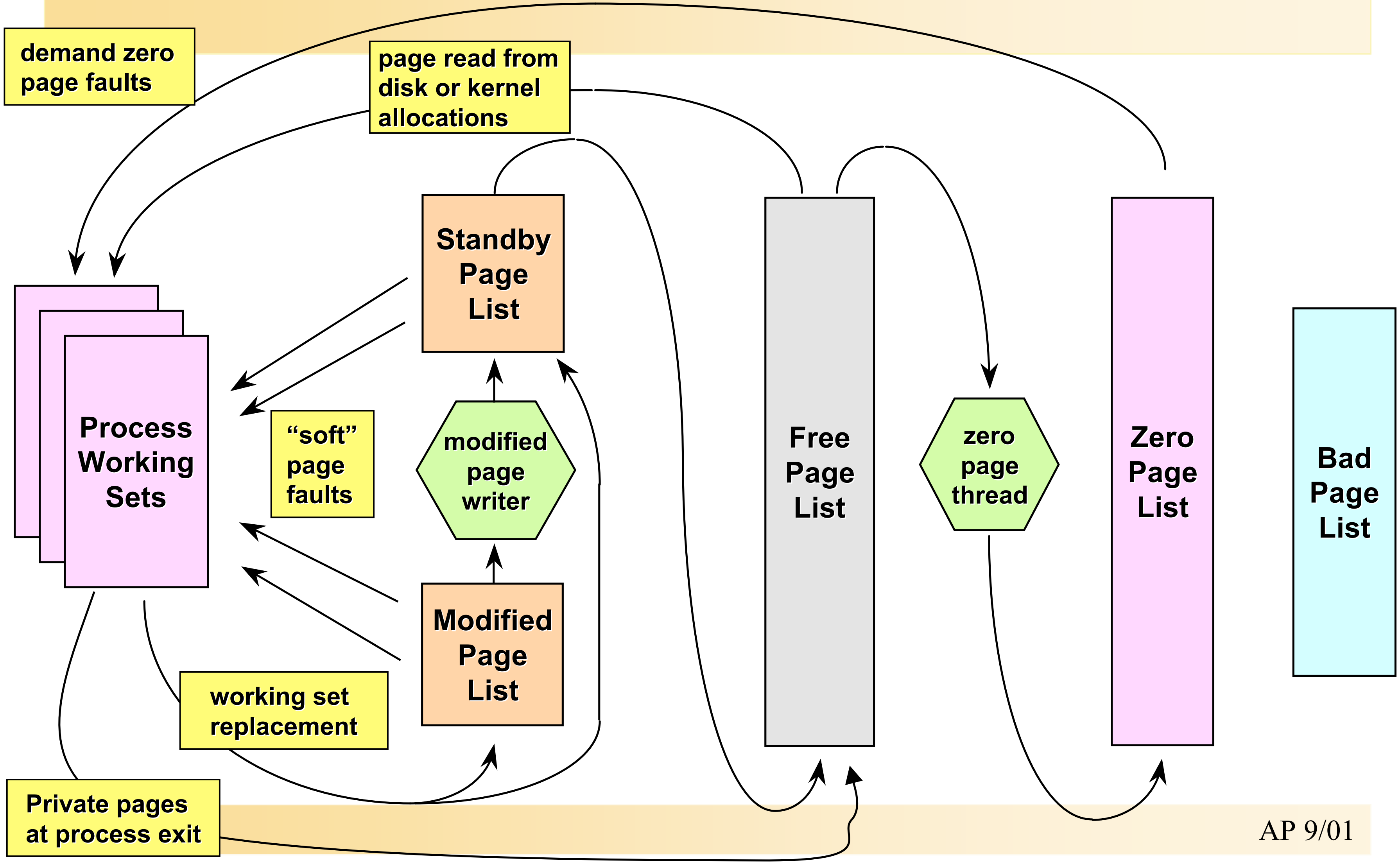
Agenda

- Introduction
- Process Memory
- Free Memory
- System Memory

Unassigned Physical Memory

- System keeps unassigned (available) physical pages on one of several lists:
  - Free page list
  - Modified page list
  - Standby page list
  - Zero page list
  - Bad page list - pages that failed memory test at system startup
- Lists are implemented by entries in the "PFN database"
  - Maintained as FIFO lists or queues

Paging Dynamics



Standby And Modified Page Lists

- Used to:
  - Avoid writing pages back to disk too soon
  - Avoid releasing pages to the free list too soon
- The system can replenish the free page list by taking pages from the top of the standby page list
  - This breaks the association between the process and the physical page
  - I.e., the system no longer knows if the page still contains the process's info
- Pages move from the modified list to the standby list
  - Modified pages' contents are copied to the pages' backing stores (usually the paging file) by the modified page writer (see next slide)
  - The pages are then placed at the bottom of the standby page list
- Pages can be faulted back into a process from the standby and modified page list
  - The SPL and MPL form a system-wide cache of "pages likely to be needed again"

Modified Page Writer

- Moves pages from modified to standby list, and copies their contents to disk
  - I.e., this is what writes the paging file and updates mapped files (including the file system cache)
- Two system threads
  - One for mapped files, one for the paging file
- Triggered when
  - Memory is overcomitted (too few free pages)
  - Or modified page threshold is reached
  - Does not flush entire modified page list

for memory size	modified page threshold	retain modified pages
small (<13 MB)	100	40
medium (13-19)	150	80
large (19-32)	300	150
huge (over 32 M)	600	256

Zero Page List

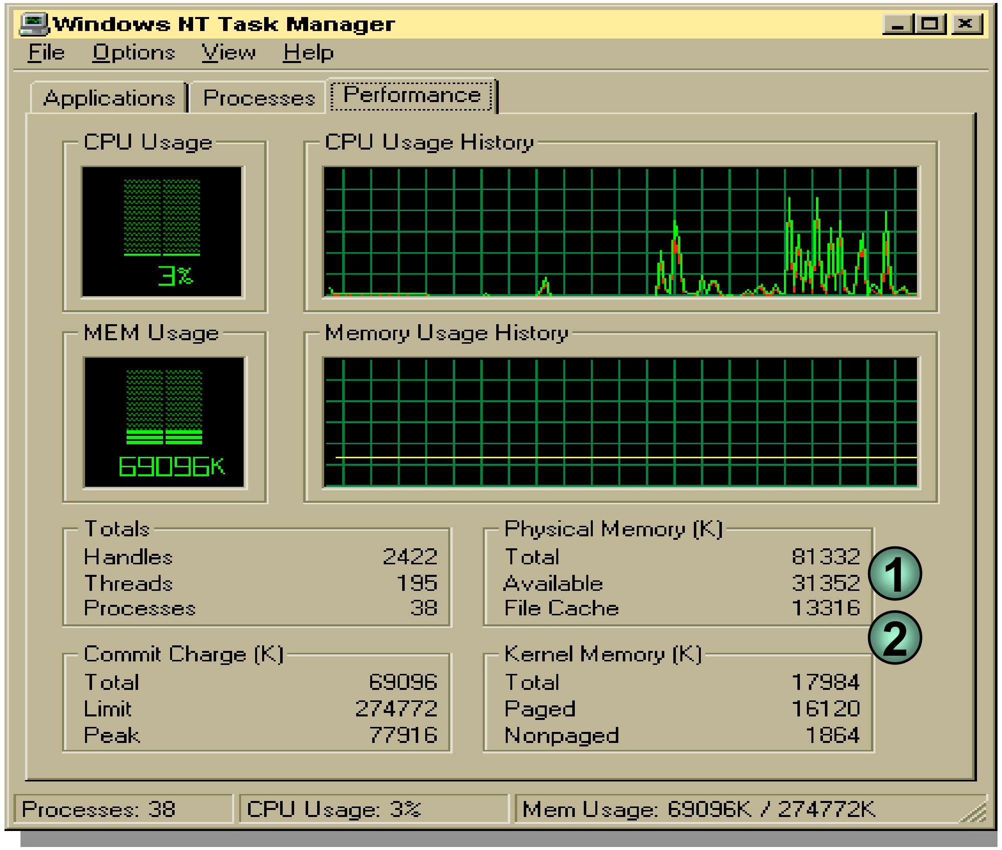
- Large uninitialized data regions are mapped to demand zero pages
- On first reference to such a page, a page is allocated from the zero page list
  - No need to read zeroes from disk to provide the “data”
  - After modification, these pages are mapped to the paging file
- Zero page list is replenished by the “zero page thread”
  - Thread 0 in “System” process (routine name is Phase1Initialization)
  - Runs at priority 0
    (lower than can be reached by Win32 applications, but above idle threads)
  - One per system (even on SMP)
  - Takes pages from the free page list, fills them with zeroes, and puts them on the zero page list while the CPU is otherwise idle
  - Usually is waiting on an event - which is set when, after resolving a fault, system notices that zero page list is too small

Memory Management Information
Task manager performance tab

1. "Available" memory = total of free, zero, and standby lists (majority usually are standby pages)

2. Windows 2000: System cache = total of cache, paged pool, system code + size of standby list

   (displayed instead of file cache which did not include size of standby list)



Screen snapshot from: Task Manager | Performance tab

Examining Sizes of Page Lists

- Must use Kernel Debugger

```
kd> !memusage
!memusage
 loading PFN database..........................
            Zeroed:      0 (      0 kb)
              Free:    322 (   1288 kb)
           Standby:   1032 (   4128 kb)
          Modified:    119 (    476 kb)
   ModifiedNoWrite:      0 (      0 kb)
      Active/Valid:   2623 (  10492 kb)
        Transition:      0 (      0 kb)
           Unknown:      0 (      0 kb)
             TOTAL:   4096 (  16384 kb)
```

Screen snapshot from: Kernel debugger !memusage command

Agenda

- Introduction
- Process Memory
- Free Memory
- System Memory

System Memory Usage

- Windows 2000 OS and driver memory usage breaks down into:
  - Nonpageable code
  - Pageable code
  - File system cache
  - Nonpaged pool
  - Paged pool
- Let’s start with the memory pools

System Memory Pools

- Windows 2000 provides two system memory pools for the OS and drivers:
  - Nonpaged pool (always in physical memory)
  - Paged pool (may be paged out)
- Pool sizes are a function of memory size and system type (Server versus Workstation)
  - Can be overidden in Registry:
    - HKLM\System\CurrentControlSet\Control\Session Manager\Executive
  - See TechNet articles (search for “nonpaged”)
    - http://technet.microsoft.com/cdonline/content/complete/boes/bo/winntas/technote/planning/ntdomsiz.htm

System Memory Pools

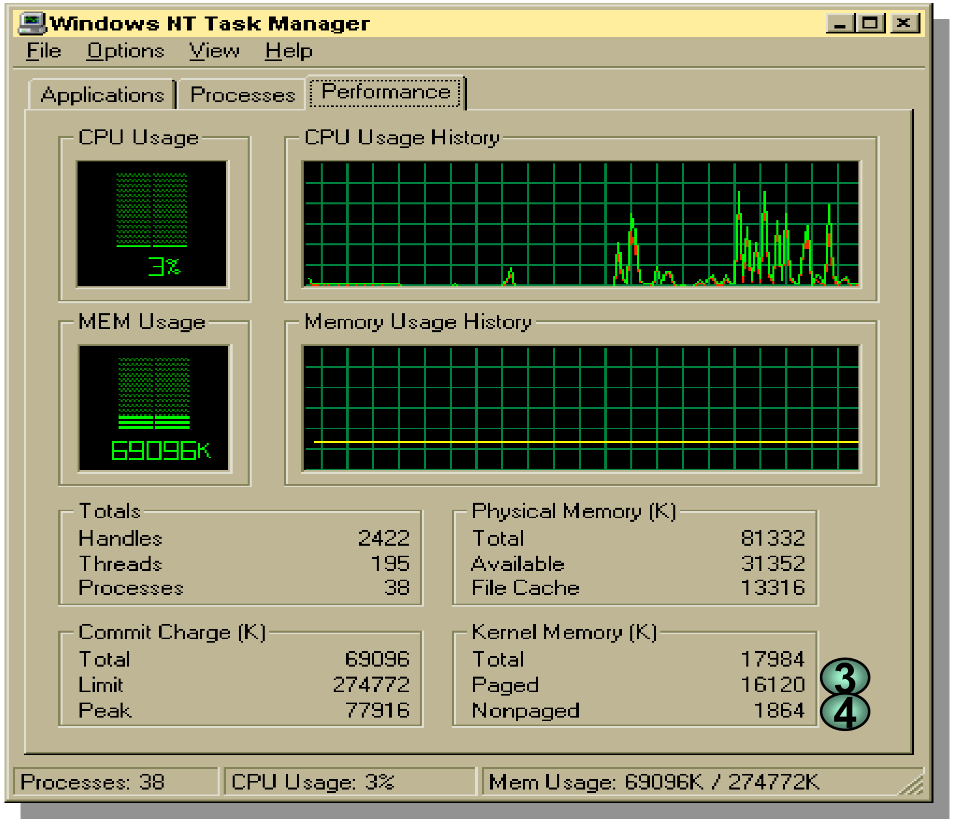
- Nonpaged pool has initial size and upper max
  - Upper limit: 256 MB on x86 (128MB on Windows NT 4.0)
    - 128MB for /3GB systems
  - Note: Performance counter displays current size
    - Maximum size stored in kernel variable MmMaximumNonPagedPoolInBytes
    - Therefore cannot easily tell when approaching max
- Paged pool limited by pagefile size
  - Upper limit: 192MB on x86, 240MB on Alpha
- System cache can be up to 960MB virtual (512MB in Windows NT 4.0)

Memory Management Information

Task manager performance tab

③ "Kernel Memory Paged" = physically resident size of paged pool

④ "Kernel Memory Nonpaged" = physical size of nonpaged pool



Screen snapshot from: Task Manager | Performance tab

Monitoring Pool Usage

- Poolmon.exe in in \support\tools on Windows 2000 CD
- Must first turn on “Pool tagging” with GFLAGS (ResKit) and reboot
- Shows paged and nonpaged pool consumption by data structure “tag” (no official list - many are self-explanatory)

Command Prompt - poolmon

```
Memory:  130484K Avail:   63296K  PageFlts:     0   InRam Krnl: 2816K P:12908K
Commit:   56740K Limit: 322000K Peak:   57028K           Pool N: 2464K P:15072K
Tag  Type      Allocs             Frees             Diff   Bytes        Per Alloc

Key  Paged     33275 (   0)       33013 (   0)       262   16800 (    0)       64
CMkb Paged     33275 (   0)       33155 (   0)       120   23104 (    0)      192
ObSq Paged     31597 (   0)       31597 (   0)         0       0 (    0)        0
Io   Nonp      29991 (   2)       29915 (   2)        76   16480 (    0)      216
IoNm Paged      9968 (   0)        9056 (   0)       912  129984 (    0)      142
CM   Paged      7050 (   0)        6519 (   0)       531 9335104 (    0)    17580
File Nonp       5477 (   0)        3932 (   0)      1545  296640 (    0)      192
NtFC Paged      5039 (   0)        5011 (   0)        28    1792 (    0)       64
Gh 5 Paged      3572 (   0)        3368 (   0)       204  264320 (    0)     1295
Gh 4 Paged      3498 (   0)        3477 (   0)        21    4256 (    0)      202
Sect Paged      2862 (   0)        2596 (   0)       266   34048 (    0)      128
SeSd Paged      2839 (   0)        2651 (   0)       188   33536 (    0)      178
Vad  Nonp       2660 (   0)        1629 (   0)      1031   65984 (    0)       64
MmCa Nonp       2517 (   0)        1515 (   0)      1002   96160 (    0)       95
Npfs Nonp       2305 (   0)        2192 (   0)       113   14880 (    0)      131
```

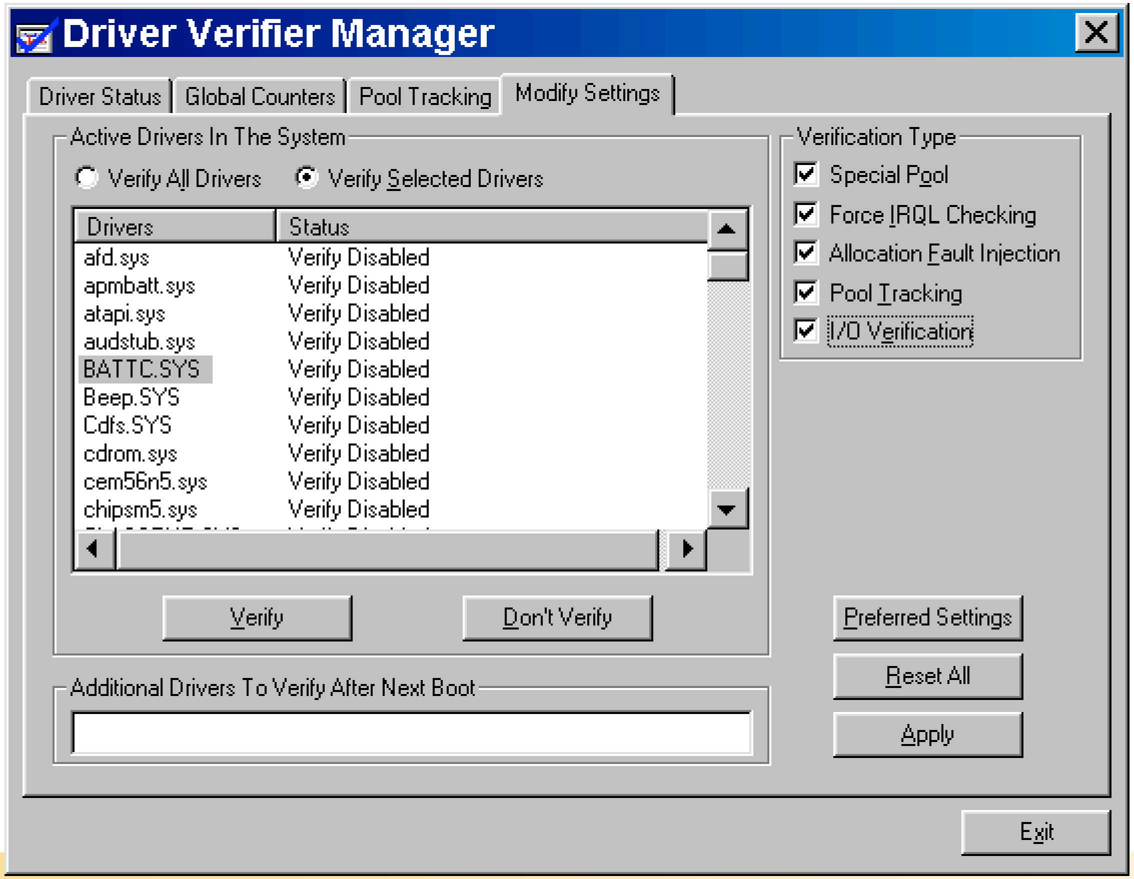
- ? Displays help, p toggles between nonpaged, paged pool, or both
- b Sorts by total # of bytes

Driver Verifier

- Additional driver integrity checking features in Windows 2000
  - Pool integrity checking (special pool)
  - Unmap pageable memory at high IRQL
  - Simulating low resource conditions
  - API verification
  - Memory leak detection
  - I/O packet memory verification
- GUI utility to enable (verifier.exe)
- For more info:
  - http://www.microsoft.com/hwdev/driver/driververify.htm

Driver Verifier

- Verifier.exe



Special Pool

- One of the many features in the Driver Verifier is available on Window NT 4.0 SP4
- Helps catch driver and OS memory corruptions
  - Puts read only page before and after each allocation
  - Each allocation goes in its own page
  - Front of a page (underrun checking)/end of page (overrun checking)
- To enable on NT4, add special registry keys under:
  HKEY_LOCAL_MACHINE\CurrentControlSet\Control
  \Session Manager\Memory Management
- To enable on Windows 2000, use Verifier.exe
- See article Q192486 for details

Nonpageable System Code

- Most drivers + parts of NTOSKRNL.EXE are nonpaged
- No performance counter to get total size
- To get size of nonpageable system code, run \ntreskit\pstat.exe and add columns 1 and 2
  - (7) non-paged code
  - (8) non-paged data
  - (9) pageable code+data
    - output of “drivers” (\ntreskit\drivers.exe) is similar
    - Win32K.Sys is paged, even though it shows up as nonpaged - must subtract from list

Command Prompt

```
D:\A>pstat
Pstat version 0.3:  memory: 81332 kb  uptime:  0  0:
        .
        .
        .                          (7)        (8)       (9)
  ModuleName Load Addr   Code       Data     Paged
-----------------------------------------------------------
ntoskrnl.exe 80100000  264192     39488    431936 Fri Au
     hal.dll 80010000   20320      2752      9344 Thu Ju
  Pcmcia.sys 80001000   15648       672         0 Fri Ju
   atapi.sys 8000b000   14720        32         0 Wed Ju
SCSIPORT.SYS 801d3000    9184        32     14368 Tue Ju
 sparrow.sys 801db000   15168        96         0 Wed Ju
  amsint.sys 801e0000    9856         0         0 Wed Ju
  Atdisk.sys 801e4000   12384        64         0 Tue Ju
    Disk.sys 801eb000    2368         0      7744 Wed Ju
  CLASS2.SYS 801ef000    6912         0      1504 Tue Ju
    Ntfs.sys 801f3000   67392      5376    267072 Thu Ju
    TAPE.SYS f887c000    7872         0      4192 Tue Ju
   Cdrom.SYS f8710000   12608        32      3072 Tue Ju
        .
        .
        .
CANON800.DLL fd7a5000       0         0         0
   ntdll.dll 77f60000  233472     20480         0 Mon Ju
-----------------------------------------------------------
       Total          2478400    142016   1663840

D:\A>
```

System Working Set

- Just as processes have working sets, pageable system code and data lives in a working set
- Pageable components:
  - Paged pool
  - Pageable code and data in the exec
  - Pageable code and data in kernel-mode drivers, Win32K.Sys, graphics drivers, etc.
  - Global file system data cache
- To get physical (resident) size of these with PerfMon, look at:
  - Memory | Pool Paged Resident Bytes
  - Memory | System Code Resident Bytes
  - Memory | System Driver Resident Bytes
  - Memory | System Cache Resident Bytes
- NOTE: Memory | Cache bytes counter is really total of these four "resident" (physical) counters