# Unit OS5: Memory Management

## 5.4. Windows Memory Management Internals



# 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)

# Roadmap for Section 5.4.

- From working sets to paging dynamics
- Birth of a process working set
- Working set trimming, heuristics
- Paging, paging dynamics
- Hard vs. soft page faults
- Page files

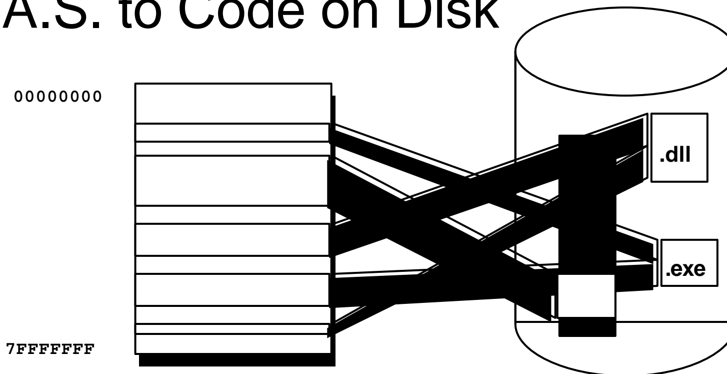


# *All** Committed Virtual Address Space is Mapped to Files

- Ranges of virtual address space are mapped to ranges of blocks w ithin disk files
  - Each such range is defined by a “section object” (visible in WinObj if named)
  - These files are the “backing store” for virtual address space
- Commonly-used files are:
  - The system paging file
    - For writeable, nonshareable (copy-on-write) pages
    - i.e. most writeable data
  - 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)

***almost - exceptions include nonpaged pool & system code mapped into each user process and/or physical memory allocated through new Windows 2000 extended addressing services**

# Application Startup Maps V.A.S. to Code on Disk

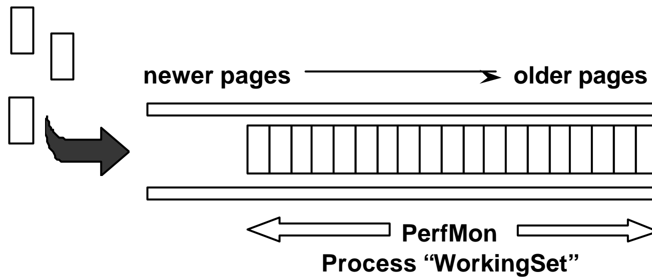



- See link/dump/header, or QuickView for .exe's and .dll's
- CreateFileMapping, MapViewOfFile simply make the mechanism available to application-level code
- All of these files may simultaneously be mapped by other processes

# Working Set

- Working set: All the physical pages "owned" by a process
  - Essentially, all the pages the process can reference without incurring a page fault
- Working set limit: The maximum pages the process *can* own
  - When limit is reached, a page must be released for every page that's brought in ("working set replacement")
  - Default upper limit on size for each process
  - System-wide maximum calculated & stored in MmMaximumWorkingSetSize
    - approximately RAM minus 512 pages (2 MB on x86) minus min size of system working set (1.5 MB on x86)
    - Interesting to view (gives you an idea how much memory you've "lost" to the OS)
  - True upper limit: 2 GB minus 64 MB

# Working Set List




- A process always starts with an empty working set
  - It then incurs *page faults* when referencing a page that isn't in its working set
  - Many page faults may be resolved from memory (to be described later)

# Birth of a Working Set

- Pages are brought into memory as a result of page faults
  - Prior to XP, no pre-fetching at image startup
  - But readahead is performed after a fault
    - See MmCodeClusterSize, MmDataClusterSize, MmReadClusterSize
- If the page is not in memory, the appropriate block in the associated file is read in
  - Physical page is allocated
  - Block is read into the physical page
  - Page table entry is filled in
  - Exception is dismissed
  - Processor re-executes the instruction that caused the page fault (and this time, it succeeds)
- The page has now been "faulted into" the process "working set"

# Prefetch Mechanism

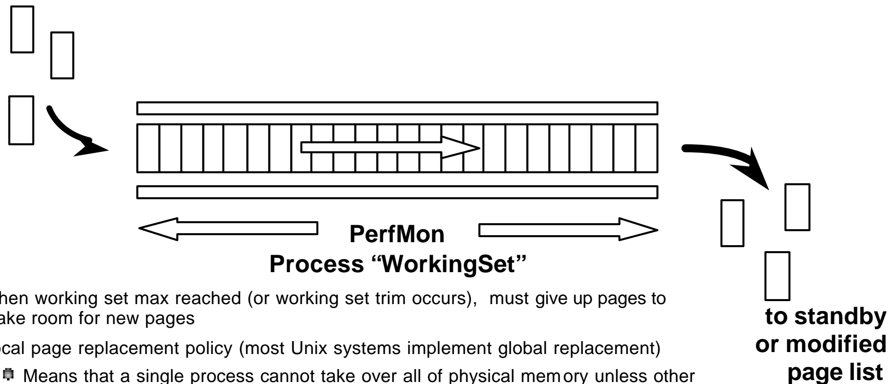
- First 10 seconds of file activity is traced and used to prefetch data the next time
  - Also done at boot time (described in Startup/Shutdown section)
- Prefetch "trace file" stored in \Windows\Prefetch
  - Name of .EXE-<hash of full path>.pf
- When application run again, system automatically
  - Reads in directories referenced
  - Reads in code and file data
    - Reads are asynchronous
    - But waits for all prefetch to complete



# Prefetch Mechanism

- In addition, every 3 days, system automatically defrags files involved in each application startup
- Bottom line: Reduces disk head seeks
  - This was seen to be the major factor in slow application/system startup
- 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

# Working Set Replacement




- When working set max reached (or working set trim occurs), must give up pages to make room for new pages
- Local page replacement policy (most Unix systems implement global replacement)
  - Means that a single process cannot take over all of physical memory unless other processes aren’t using it
- Page replacement algorithm is least recently accessed (pages are aged)
  - On UP systems only in Windows 2000 – done on all systems in Windows XP/Server 2003
- New VirtualAlloc flag in XP/Server 2003: MEM_WRITE_WATCH

# Soft vs. Hard Page Faults

- Types of “soft” page faults:
  - Pages can be faulted back into a process from the standby and modified page lists
  - A shared page that’s valid for one process can be faulted into other processes
- Some hard page faults unavoidable
  - Process startup (loading of EXE and DLLs)
  - Normal file I/O done via paging
    - Cached files are faulted into system working set
- To determine paging vs. normal file I/Os:
  - Monitor Memory->Page Reads/sec
    - Not Memory->Page Faults/sec, as that includes soft page faults
  - Subtract System->File Read Operations/sec from Page Reads/sec
  - Or, use Filemon to determine what file(s) are having paging I/O (asterisk next to I/O function)
  - Should not stay high for sustained period

# 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
```



# Working Set System Services

- Min/Max set on a per-process basis
  - Can view with !process in Kernel Debugger
- System call below can adjust min/max, but has minimal effect prior to Server 2003
  - Limits are “soft” (many processes larger than max)
  - Memory Manager decides when to grow/shink working sets
- New system call in Server 2003 (SetProcessWorkingSetSizeEx) allows setting hard min/max
- Can also self-initiate working set trimming
  - Pass -1, -1 as min/max working set size (minimizing a window does this for you)

**Windows API:**

```
SetProcessWorkingSetSize(
   HANDLE hProcess,
   DWORD dwMinimumWorkingSetSize,
   DWORD dwMaximumWorkingSetSize)
```

# Locking Pages

- Pages may be locked into the process working set
  - Pages are guaranteed in physical memory (“resident”) when any thread in process is executing

```
Windows API:
status = VirtualLock(baseAddress, size);
status = VirtualUnlock(baseAddress, size);
```

- Number of lockable pages is a fraction of the maximum working set size
  - Changed by SetProcessWorkingSetSize
- Pages can be locked into physical memory (by kernel mode code only)
  - Pages are then immune from “outswapping” as well as paging

```
MmProbeAndLockPages
```

# 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)**

Windows NT Task Manager

File Options View Help

Applications | Processes | Performance

| Image Name | PID | CPU | CPU Ti... | Mem Usage (1) | VM Size (2) |
|---|---|---|---|---|---|
| System Idle Pr... | 0 | 97 | 8:24:18 | 16 K | 0 K |
| System | 2 | 00 | 0:00:35 | 200 K | 36 K |
| smss.exe | 20 | 00 | 0:00:00 | 0 K | 164 K |
| csrss.exe | 24 | 00 | 0:00:12 | 676 K | 1492 K |
| WINLOGON.E... | 34 | 00 | 0:00:02 | 0 K | 712 K |
| SERVICES.EXE | 40 | 00 | 0:00:04 | 1024 K | 1124 K |
| LSASS.EXE | 43 | 00 | 0:00:00 | 200 K | 948 K |
| SPOOLSS.EXE | 67 | 00 | 0:00:00 | 60 K | 2008 K |
| NETDDE.EXE | 74 | 00 | 0:00:00 | 0 K | 528 K |
| AMGRSRVC.E... | 84 | 00 | 0:00:00 | 0 K | 1056 K |
| clipsrv.exe | 90 | 00 | 0:00:00 | 0 K | 416 K |
| SDSRV.EXE | 95 | 00 | 0:00:00 | 20 K | 576 K |
| RPCSS.EXE | 109 | 00 | 0:00:00 | 320 K | 820 K |
| TCPSVCS.EXE | 112 | 00 | 0:00:00 | 172 K | 496 K |
| TAPISRV.EXE | 116 | 00 | 0:00:00 | 200 K | 664 K |
| wfxsvc.exe | 127 | 00 | 0:00:00 | 0 K | 324 K |
| EXPLORER.E... | 130 | 00 | 0:00:58 | 2604 K | 1768 K |
| PSTORES.EXE | 137 | 00 | 0:00:00 | 32 K | 1812 K |
| RASMAN.EXE | 140 | 00 | 0:00:00 | 44 K | 1080 K |
| wfxmod32.exe | 142 | 00 | 0:00:00 | 1604 K | 1496 K |

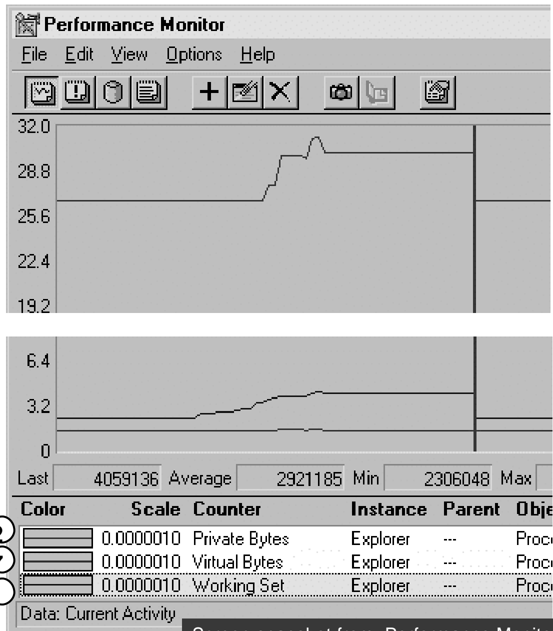
End Process (3)

Processes: 38 | CPU Usage: 3% | Mem Usage: 68312K / 274772K

Screen snapshot from:
Task Manager | Processes tab

# 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

# Balance Set Manager

- Nearest thing NT 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
  - CPU starvation avoidance - already covered

# Process Exploder
## Memory Information for a Process

Process Explode

Process Id 181 POWERPNT.EXE

Objects
Process Objects 29
Thread Objects 166
Event Objects 440
Semaphore Objects 67
Mutex Objects 78
Section Objects 282

181 POWERPNT.EXE

Thread Data
User PcValue 0x77e724e7
Start Address 0x77f052cc
#Cntxt Sw
# 6
190361

Thread Times
E 1:23:38.996
K 0:01:01.808
U 0:02:35.663

Thread Priority
Highest
Above Norma
Normal
Below Normal
Lowest

Dynamic 14

Security
Process
Thread
P.Token
T.Token

Token
Process
Thread

Base Priority
Normal
High
Idle

Times
E 1:23:38.996
K 0:01:11.092
U 0:02:53.619

User Address Space
TotalImageCon 17212 Kb
NoAccess 0 Kb
ReadOnly 3684 Kb
ReadWrite 416 Kb
WriteCopy 84 Kb
Execute 13028 Kb

Mapped Commit 7340 Kb
NoAccess 0 Kb
ReadOnly 6340 Kb
ReadWrite 552 Kb
WriteCopy 0 Kb
Execute 448 Kb

Private Commit 13048 Kb
NoAccess 0 Kb
ReadOnly 4 Kb
ReadWrite 13016 Kb
WriteCopy 0 Kb
Execute 28 Kb

Kill App Exit Hide

**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". 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**



# Process Exploder
## Memory Information for a Process (and Systemwide

Refresh Time 2:27:59.277

Vm Counts
Peak Vsize 69812 Kb
Vsize 66928 Kb
Fault Count 62758
Peak WS 14876 Kb
WS 3960 Kb
Peak PF 15816 Kb
PF 14200 Kb
Private Pg 14200 Kb
Peak Paged 41 Kb
Paged 39 Kb
Peak Non 19 Kb
NonPaged 19 Kb

Pooled Quotas
Peak Paged 1160 Kb
Cur Paged 825 Kb
Lim Paged 828 Kb
Peak Non 301 Kb
Cur Non 214 Kb
Lim Non 256 Kb
Peak PF 31168 Kb
Cur PF 29556 Kb
Lim PF Unlimited

Refresh

**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**

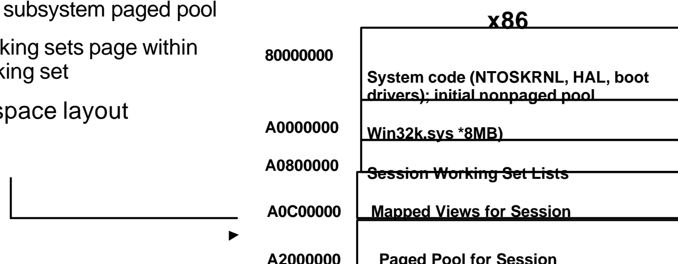
**Note, "limits" in the last three groups are per-process limits, ie how much each process can use of these**

# System Working Set

- Just as processes have working sets, NT's pageable system-space code and data lives in the "system working set"
- Made up of 4 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
  - ⑤ Memory | Cache bytes counter is total of these four "resident" (physical) counters (not just the cache; in NT4, same as "File Cache" on Task Manager / Performance tab)

# Session Working Set

- New memory management object to support Terminal Services in Windows 2000/XP/Server 2003
- Session = an interactive user
- Session working set = the memory used by a session
  - Instance of WinLogon and Win32 subsystem process
  - WIN32K.SYS remapped for each unique session
    - Win32 subsystem objects
    - Win32 subsystem paged pool
  - Process working sets page within session working set
- Revised system space layout



# Managing Physical Memory

- System keeps unassigned 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

- New pages are allocated to working sets from the top of the free or zero page list
- Pages released from the working set due to working set replacement go to the bottom of:
  - The modified page list (if they were modified while in the working set)
  - The standby page list (if not modified)
    - Decision made based on “D” (dirty = modified) bit in page table entry
  - Association between the process and the physical page is still maintained while the page is on either of these lists

## Standby and Modified Page Lists

- Modified pages go to modified (dirty) list
  - Avoids writing pages back to disk too soon
- Unmodified pages go to standby (clean) list
- They form a system-wide cache of “pages likely to be needed again”
  - Pages can be faulted back into a process from the standby and modified page list
  - These are counted as page faults, but not page reads



## Modified Page Writer

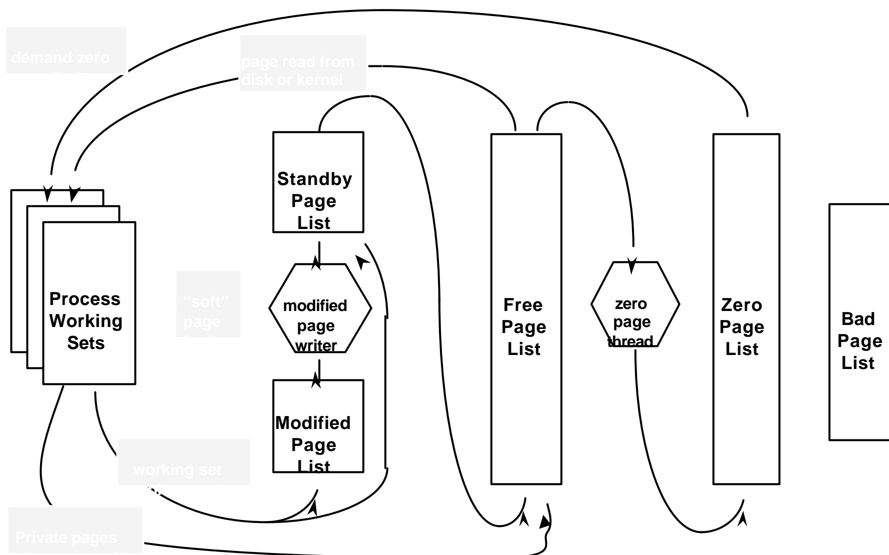
- When modified list reaches certain size, modified page writer system thread is awoken to write pages out
  - See MmModifiedPageMaximum
  - Also triggered when memory is overcomitted (too few free pages)
  - Does not flush entire modified page list
- Two system threads
  - One for mapped files, one for the paging file
- Pages move from the modified list to the standby list
  - E.g. they can still be soft faulted into a working set

# Free and Zero Page Lists

- Free Page List
  - Used for page reads
  - Private modified pages go here on process exit
  - Pages contain junk in them (e.g. not zeroed)
  - On most busy systems, this is empty
- Zero Page List
  - Used to satisfy demand zero page faults
    - References to private pages that have not been created yet
  - When free page list has 8 or more pages, a priority zero thread is awoken to zero them
  - On most busy systems, this is empty too



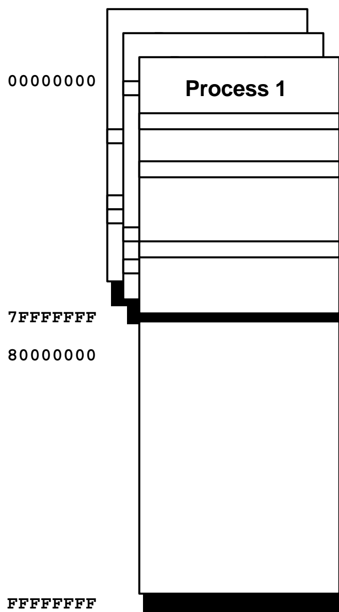
# Paging Dynamics

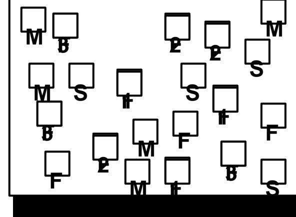


# Zero Page List

- 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"
- 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

# Working Sets in Memory



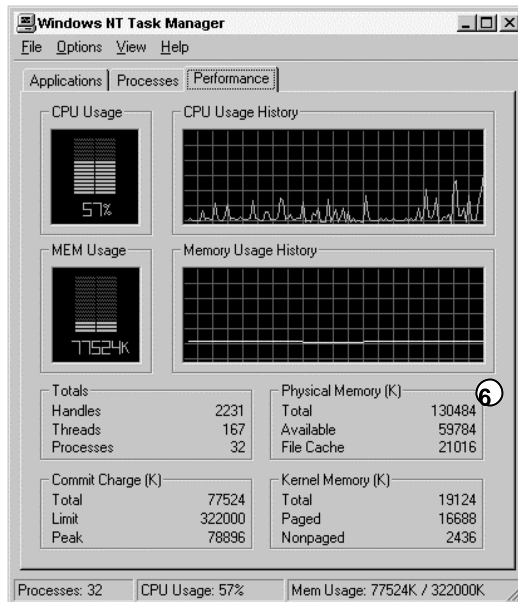




- As processes incur page faults, pages are removed from the free, modified, or standby lists and made part of the process working set
- A shared page may be resident in several processes' working sets at one time (this case not illustrated here)

# Memory Management Information

## 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"




Screen snapshot from:
Task Manager | Performance tab

# 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

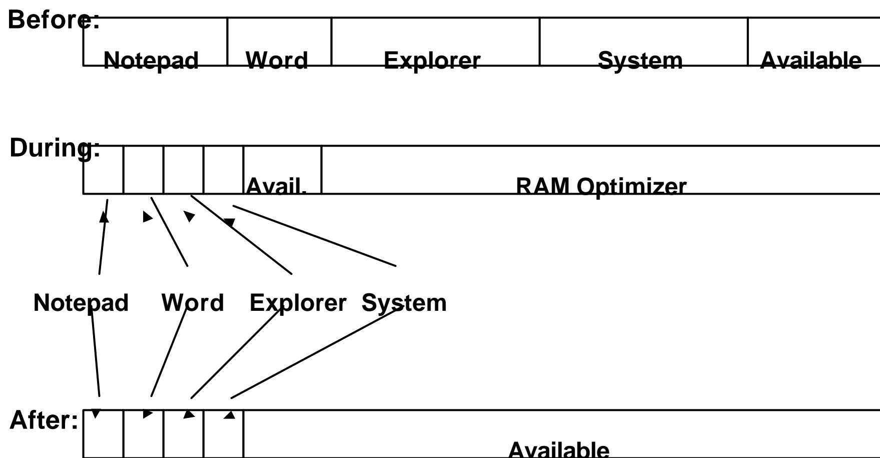
# PFN Database

- Only way to get actual size of physical memory lists is to use !memusage in Kernel Debugger

```
lkd> !memusage
 loading PFN database
                Zeroed:      0 (      0 kb)
                  Free:      0 (      0 kb)
               Standby: 139069 (556276 kb)
              Modified:    161 (    644 kb)
      ModifiedNoWrite:      0 (      0 kb)
         Active/Valid: 122759 (491036 kb)
           Transition:      8 (     32 kb)
              Unknown:      0 (      0 kb)
                TOTAL: 261997 (1047988 kb)
```

Screen snapshot from:kernel debugger !memusage command

# Why "Memory Optimizers" are Fraudware




**See Mark's article on this topic at http://www.winnetmag.com/Windows/Article/ArticleID/41095/41095.html**

# Page Files

- What gets sent to the paging file?
  - Not code – only modified data (code can be re-read from image file anytime)
- When do pages get paged out?
  - Only when necessary
  - Page file space is only reserved at the time pages are written out
  - Once a page is written to the paging file, the space is occupied until the memory is deleted (e.g. at process exit), even if the page is read back from disk
- Windows XP (& Embedded NT4) can run with no paging file
  - NT4/Win2K: zero pagefile size actually creates a 20MB temporary page file (\temppf .sys)
  - WinPE never has a pagefile
- Page file maximums:
  - 16 page files per system
  - 32-bit x86: 4095MB
  - 32-bit PAE mode, 64-bit systems: 16 TB



# Why Page File Usage on Systems with Ample Free Memory?

- Because memory manager doesn't let process working sets grow arbitrarily
  - Processes are not allowed to expand to fill available memory (previously described)
    - Bias is to keep free pages for new or expanding processes
  - This will cause page file usage early in the system life even with ample memory free
- We talked about the standby list, but there is another list of modified pages recently removed from working sets
  - Modified private pages are held in memory in case the process asks for it back
  - When the list of modified pages reaches a certain threshold, the memory manager writes them to the paging file (or mapped file)
  - Pages are moved to the standby list, since they are still "valid" and could be requested again

# Sizing the Page File

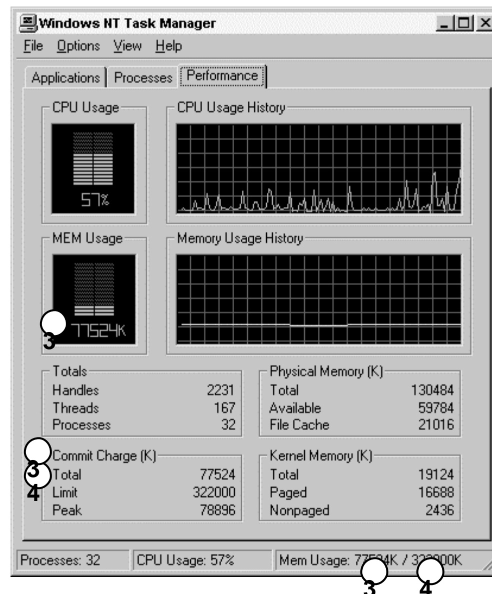
- Given understanding of page file usage, how big should the total paging file space be?
  - (Windows 2000/XP supports multiple paging files)
- Size should depend on total private virtual memory used by applications and drivers
  - Therefore, not related to RAM size (except for taking a full memory dump)
- Worst case: system has to page all private data out to make room for code pages
  - To handle, minimum size should be the maximum of VM usage ("Commit Charge Peak")
    - Hard disk space is cheap, so why not double this
  - Normally, make maximum size same as minimum
  - But, max size could be much larger if there will be infrequent demands for large amounts of page file space
    - Performance problem: page file extension will likely be very fragmented
    - Extension is deleted on reboot, thus returning to a contiguous page file



# 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"
- (4) "Commit charge limit" = sum of physical memory available for processes + current total size of paging file(s)
- does not reflect true maximum page file sizes (expansion)
- when "total" reaches "limit", further VirtualAlloc attempts by any process will fail



# Contiguous Page Files

- A few fragments won't hurt, but hundreds of fragments will
- Will be contiguous when created if contiguous space available at that time
- Can defrag with Pagedefrag tool (freeware - www.sysinternals.com)
  - Or buy a paid defrag product…



# When Page Files are Full

- 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 Task Manager, Processes tab, VM Size column
    - or Perfmon "private bytes", same counter
  - 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" in Task Manager

# 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



# System Nonpaged Memory

- Nonpageable components:
  - Nonpageable parts of NtosKrnl.Exe, drivers
  - Nonpaged pool (see PerfMon, Memory object: Pool nonpaged bytes)
  - non-paged code
  - (8) non-paged data
  - (9) pageable code+data (virtual size)
  - (A) output of “drivers.exe” is similar
  - Win32K.Sys is paged, even though it shows up as nonpaged
  - Other drivers might do this too, so total nonpaged size is not really visible

Command Prompt

```
D:\A>pstat
Pstat version 0.3:  memory: 81332 kb  uptime:  0  0:
        .
        .
        .
  ModuleName Load Addr    Code(8)   Data(9)  Paged(A)
----------------------------------------------------------
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>
```

# Optimizing Applications
## Minimizing Page Faults

- **Some 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
- **First concern is to minimize number of “hard” page faults**
  - i.e. page faults to disk
  - see Performance Monitor, “Memory” object, “page faults” vs. “page reads” (this is system-wide, not per process)
  - for an individual app, see Page Fault Monitor:

```
c:\> pfmon /?
c:\> pfmon program-to-be-monitored
```

  - note that these results are highly dependent on system load (physical memory usage by other apps)

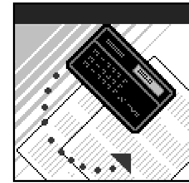
# Page Fault Monitor (pfmon)

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

# Accounting for Physical Memory Usage



LAB: to quickly see the performance counter values below:

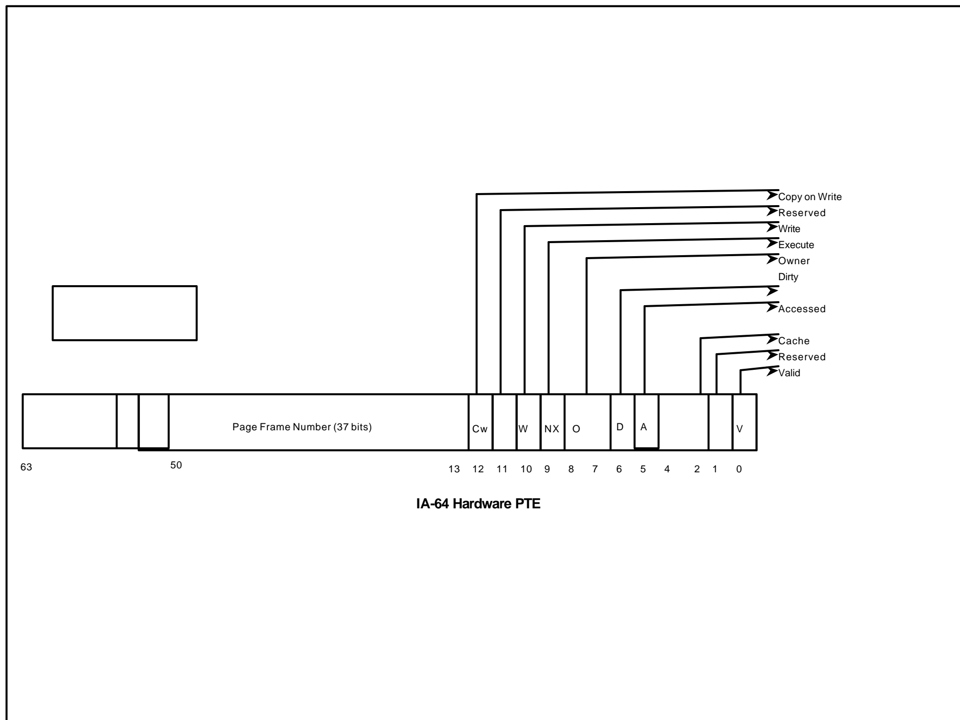
1. Run \sysint\solomon\memory-experiment.pmr
2. Click on "camera" icon twice to query values

- Process working sets
  - Perfmon: Process / Working set
  - Note, shared resident pages are counted in the process working set of every process that's faulted them in
  - Hence, the total of all of these may be greater than physical memory
- Resident system code (NTOSKRNL + drivers, including win32k.sys & graphics drivers)
  - see total displayed by !drivers 1 command in kernel debugger
- Nonpageable pool
  - Perfmon: Memory / Pool nonpaged bytes
- Free, zero, and standby page lists
  - Perfmon: Memory / Available bytes
- Pageable, but currently-resident, system-space memory
  - Perfmon: Memory / Pool paged resident bytes
  - Perfmon: Memory / System cache resident bytes

  Memory | Cache bytes counter is really total of these four "resident" (physical) counters
- Modified, Bad page lists
  - can only see size of these with !memusage command in Kernel Debugger

# 64-Bit Address Space Layout (AMD64)

| Address | Region |
|---|---|
| 0 | User-Mode per-process |
| 7FFFFFFFFFF | System Space |
| 1FFFFF0000000000 | Process Page Tables |
| 2000000000000000 | Session Space |
| 3FFFFF0000000000 | Session Space Page Tables |
| E000000000000000 -E000060000000000 | System Space |
| FFFFFF0000000000 | System Space Page Tables |




IA-64 Hardware PTE

# Further Reading

- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals, 4th Edition, Microsoft Press, 2004.
- Chapter 7 - Memory Management
  - Page Fault Handling (pp. 439 ff.)
  - Working Sets (pp. 457 ff.)
  - Address Space Layouts (pp. 413 ff.)
  - Memory Pools (pp. 399 ff.)