

Memory Management in Windows

Zoltán Micskei

<http://www.mit.bme.hu/~micskeiz>



Copyright Notice

- 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)
- <http://www.academicresourcecenter.net/curriculum/pfv.aspx?ID=6191>
- © 2000-2005 David A. Solomon and Mark Russinovich



Slides based on Windows Operating System Internals Curriculum Development Kit

Question

How much free memory do I
have right now?



Basics of memory management in Windows

- Virtual memory
 - Hiding physical memory, paging...
- Efficiency
 - demand driven paging
 - memory sharing, copy-on-write
 - file caching in memory (section object)
- Security
 - separate address space for every process
 - accessing through handles (access token)



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.)

Maximal physical memory (GB)

	x86	x64 64-bit	IA-64 64-bit
Vista Basic	4		
Vista Business	4	128	n/a
Server 2008 Standard	4	32	n/a
Server 2008 Enterprise	64	2048	n/a
Itanium	n/a	n/a	2048

max 4 GB can be handled on 32 bit

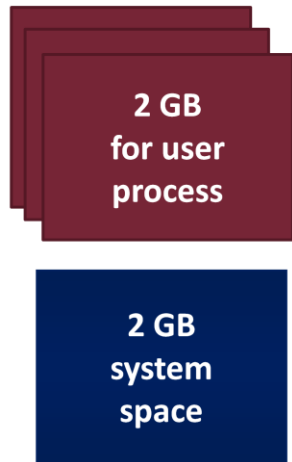
64 bit: much larger memory

Physical Address Extension (PAE)
36 address bit: CPU + OS support

Itanium: Intel server arch., not so popular

32-bit x86 address space

Default



/3GB switch



64-bit address space

- 64-bit = 17,179,869,184 GB
 - x64 currently supports 48 address bit = 262,144 GB

x64



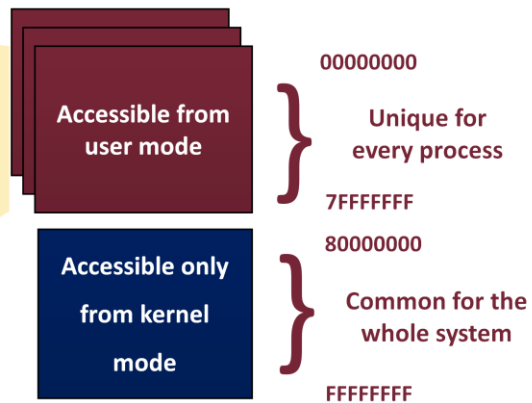
**6657 GB
system
space**



Virtual Address Space (V.A.S.)

User address space:

- The running application (.EXE and .DLLs)
- User space stack for every thread
- Data structures of the application

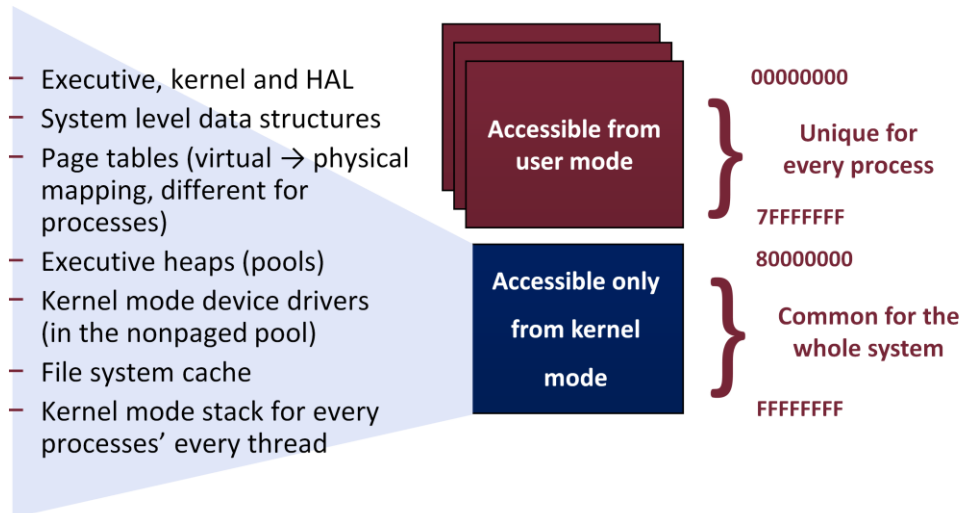


- No user process can touch another user process address space (without first opening a handle to the process, which means passing through NT security)
 - Separate process page tables prevent this
 - “Current” page table changed on context switch from a thread in 1 process to a thread in another process



Virtual Address Space (V.A.S.)

System space:



No user process can touch kernel memory

- Page protection in process page tables prevent this
- OS pages only accessible from “kernel mode”
- Threads change from user to kernel mode and back (via a secure interface) to execute kernel code, this does not affect scheduling (not a context switch)

Dynamic Kernel Address Space: Windows and the applications that run on it have bumped their heads on the address space limits of 32-bit processors. The Windows kernel is constrained by default to 2GB, or half the total 32-bit virtual address space, with the other half reserved for use by the process whose thread is currently running on the CPU. Inside its half, the kernel has to map itself, device drivers, the file system cache, kernel stacks, per-session code data structures, and both non-paged (locked-in physical memory) and paged buffers allocated by device drivers.

Prior to Windows Vista, the Memory Manager determined at boot time how much of the address space to assign to these different purposes, but this inflexibility sometimes led to situations where one of the regions became full while others still had plenty of available space. The exhaustion of an area can lead to application failures and prevent device drivers from completing I/O operations.

In 32-bit Windows Vista, the Memory Manager dynamically manages the kernel's address space, allocating and deallocating space to various uses as the demands of the workload require. Thus, the amount of virtual memory used to store paged buffers can grow when device drivers ask for more, and it can shrink when the drivers release it. Windows Vista will therefore be able to handle a wider variety of workloads and likewise the 32-bit version of the forthcoming Windows Server® code-named "Longhorn," will scale to handle more concurrent Terminal Server users.

Of course, on 64-bit Windows Vista systems, address space constraints are not currently a practical limitation and therefore require no special treatment as they are configured to their maximums.

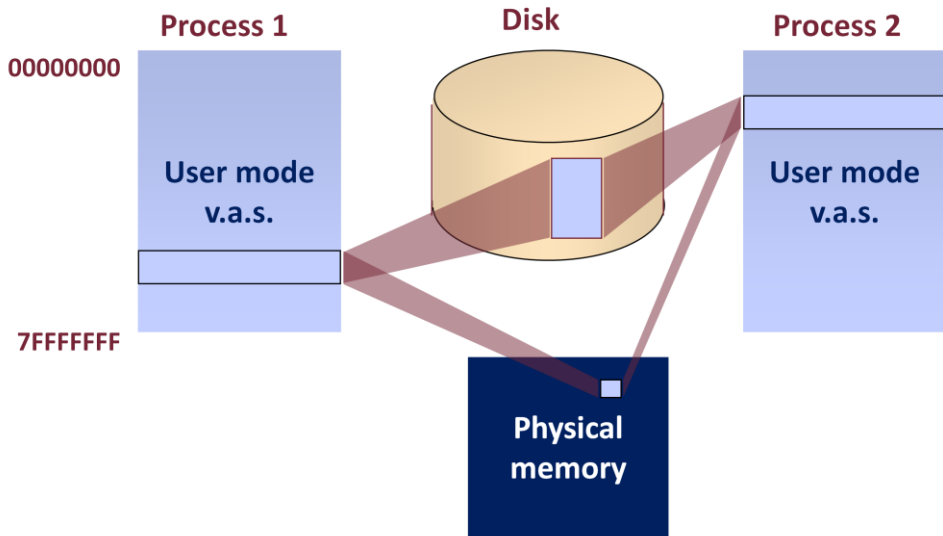
Memory allocation

- Two steps
- **Reserve:** reserving virtual address space
- **Commit:** allocating the virtual memory
- Only commits what is really needed



Shared memory

- E.g. multiple processes use the same file



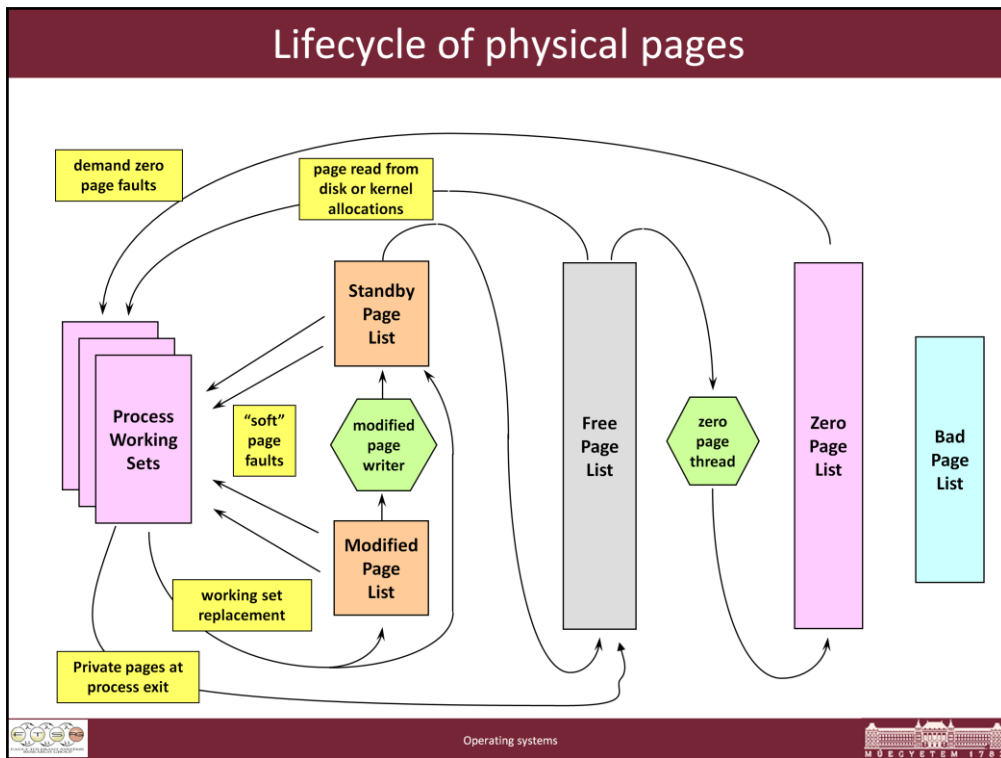
Working Set

- Working Set:
 - Physical memory pages belonging to a process
 - Can be accessed without a page fault

- Working set limit:
 - Maximal physical memory a process can have
 - If reached, should switch pages:
 - NT 4.0: modified FIFO algorithm
 - Windows 2000: Least Recently Used (UP systems)
 - If free memory falls under a limit: *trimming*



Working set: Implemented as array of working set list entries (WSLE)



Status Description

Active/valid: Page is part of working set (sys/proc), valid PTE points to it

Transition: Page not owned by a working set, not on any paging list, I/O is in progress on this page

Standby: Page belonged to a working set but was removed; not modified

Modified: Removed from working set, modified, not yet written to disk

Modified no write: Modified page, will not be touched by modified page write, used by NTFS for pages containing log entries (explicit flushing)

Free: Page is free but has dirty data in it – cannot be given to user process – C2 security requirement

Zeroed: Page is free and has been initialized by zero page thread

Bad: Page has generated parity or other hardware errors

Page file

- What?
 - Only modified data, not the code
- When?
 - Also if there is free memory
 - Processes cannot use as much physical memory as they like
 - Reservation for new processes
- One page file for each partition
 - Recommended: not on the system drive
 - But create a small one for the crash dump file
- Recommended size
 - 1,5 times physical memory (?), Fixed size (?)



Windows XP (& Embedded NT4) can run with no paging file

NT4/Win2K: zero pagefile size actually creates a 20MB temporary page file (\temp\temp.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

How to configure paging files for optimization and recovery in Windows XP
(<http://support.microsoft.com/kb/314482/en-us>)

DEMO

Page file

- Changing the size of the page file
 - GUI
 - regedit
- Perfmon: Page file utilization (%)
- Sysinternals: pagedfrg.exe



Page file: GUI: System Properties / Advanced / Performance, Settings / Advanced / Virtual Memory, Settings

Registry:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management

Memory usage

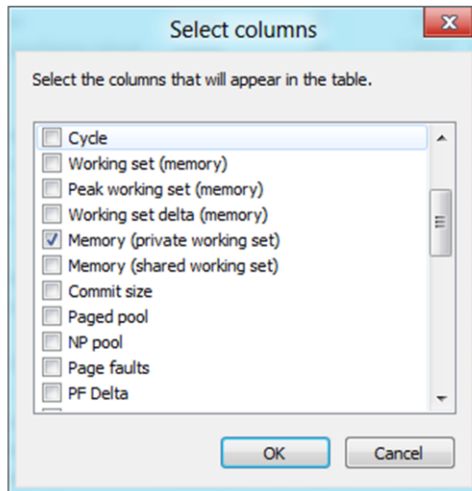
Simple question:

How much memory does a process consume and for what?

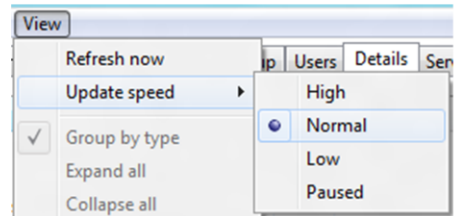


Process memory usage - 0

■ Task manager



■ Update speed



Process memory usage - 1

Name	Working set (memory)	Memory (private working set)	Commit size	PF Delta
conhost.exe	580 K	148 K	700 K	0
conhost.exe	752 K	156 K	712 K	0
ConsoleGuessGa...	4 740 K	1 496 K	13 832 K	0
csrss.exe	2 136 K	1 096 K	1 916 K	0
csrss.exe	16 184 K	956 K	1 952 K	0
dasHost.exe	7 276 K	2 260 K	2 800 K	0
devenv.exe	116 356 K	68 576 K	223 688 K	16

Memory - Working Set

- shared pages also

Memory – Private Working Set

- without shared pages

Memory – Commit size

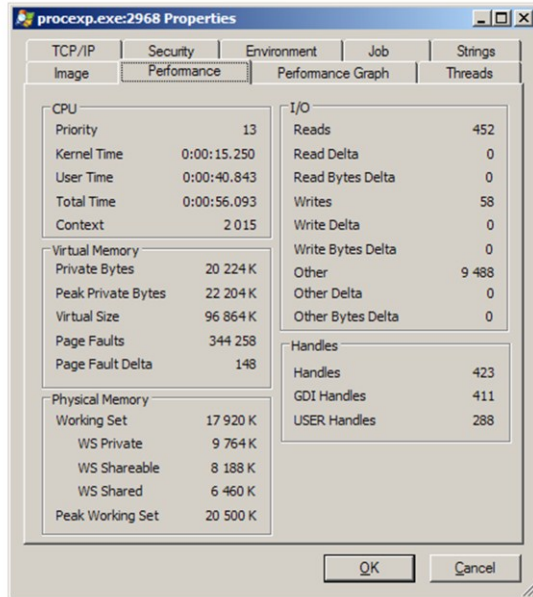
- private virtual memory
- this goes to the page file



<http://windowshelp.microsoft.com/Windows/en-US/help/e4598b92-b1c1-bc52-5e30-6871dcc59ca01033.msp>

Process memory usage - 2

- Process Explorer:
 - Details in the process property
- Virtual Memory
- Physical Memory



The screenshot shows the Performance tab of the Process Explorer Properties dialog for process procexp.exe:2968. The Performance Graph and Threads tabs are also visible. The CPU, I/O, Virtual Memory, and Physical Memory sections are expanded to show detailed statistics.

CPU	
Priority	13
Kernel Time	0:00:15.250
User Time	0:00:40.843
Total Time	0:00:56.093
Context	2 015

I/O	
Reads	452
Read Delta	0
Read Bytes Delta	0
Writes	58
Write Delta	0
Write Bytes Delta	0
Other	9 488
Other Delta	0
Other Bytes Delta	0

Virtual Memory	
Private Bytes	20 224 K
Peak Private Bytes	22 204 K
Virtual Size	96 864 K
Page Faults	344 258
Page Fault Delta	148

Physical Memory	
Working Set	17 920 K
WS Private	9 764 K
WS Shareable	8 188 K
WS Shared	6 460 K
Peak Working Set	20 500 K


Handles	
Handles	423
GDI Handles	411
USER Handles	288




DEMO

Process memory usage

■ Sysinternals VMmap



Operating systems



System memory usage

1

- All committed virtual memory
- This should go to the page file, does not mean it is currently in the page file

2

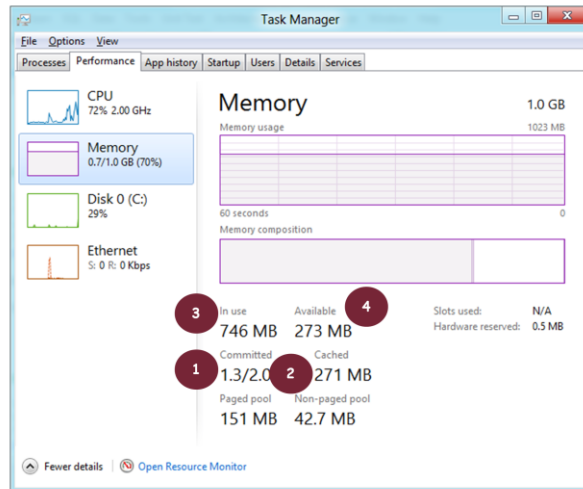
Commit limit: physical memory + actual size of the page files

3

~ Active memory pages

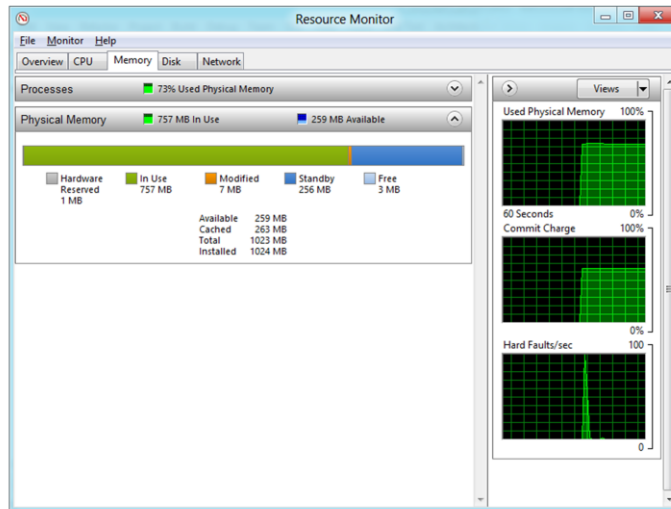
4

Standby list



Demo

Resource monitor



MemInfo: Peer Inside Memory Manager Behavior on Windows Vista and Server 2008

<http://www.alex-ionescu.com/?p=51>

Optimization: Prefetch (Windows XP)

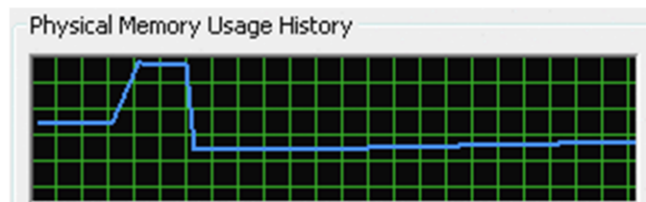
- Many page fault at the start of the application
- Always the same pages are needed
- Prefetch: watching the first 10 seconds
- Prefetch “trace file”: \Windows\Prefetch
 - Name: .EXE-<hash from full path>.pf
- On next startup
 - Needed pages are loaded *asynchronously*
- Watching the boot sequence also



Details: <http://msdn.microsoft.com/msdnmag/issues/01/12/XPKernel/default.aspx>

Another: Superfetch (Vista)

- 8 priorities to the memory pages
 - 8 standby list for each priority
- Monitoring the usage of pages
- After memory deallocations pages are slowly moved back to the standby list



Source: <http://technet.microsoft.com/en-us/magazine/cc162480.aspx>

A significant change to the Memory Manager is in the way that it manages physical memory. The Standby List management used by previous versions of Windows has two limitations. First, the prioritization of pages relies only on the recent past behavior of processes and does not anticipate their future memory requirements. Second, the data used for prioritization is limited to the list of pages owned by a process at any given point in time. These shortcomings can result in scenarios like the "after lunch syndrome," where you leave your computer for a while and a memory-intensive system application runs (such as an antivirus scan or disk defragmentation). This application forces the code and data that your active applications had cached in memory to be overwritten by the memory-intensive activities. When you return, you experience sluggish performance as applications have to request their data and code from disk.

Windows XP introduced prefetching support that improved boot and application startup performance by performing large disk I/Os to preload memory with code and file system data that it expected, based on previous boots and application launches. Windows Vista goes a big step further with SuperFetch, a memory management scheme that enhances the least-recently accessed approach with historical information and proactive memory management.

SuperFetch is implemented in %SystemRoot%\System32\Sysmain.dll as a Windows service that runs inside a Service Host process (%SystemRoot%\System32\Svchost.exe). The scheme relies on support from the Memory Manager so that it can retrieve page usage histories as well as direct the Memory Manager to preload data and code from files on disk or from a paging file into the Standby List and assign priorities to pages. The SuperFetch service

essentially extends page-tracking to data and code that was once in memory, but that the Memory Manager has reused to make room for new data and code. It stores this information in scenario files with a .db extension in the %SystemRoot%\Prefetch directory alongside standard prefetch files used to optimize application launch. Using this deep knowledge of memory usage, SuperFetch can preload data and code when physical memory becomes available.

Whenever memory becomes free—for example, when an application exits or releases memory—SuperFetch asks the Memory Manager to fetch data and code that was recently evicted. This is done at a rate of a few pages per second with Very Low priority I/Os so that the preloading does not impact the user or other active applications. Therefore, if you leave your computer to go to lunch and a memory-intensive background task causes the code and data from your active applications to be evicted from memory while you're gone, SuperFetch can often bring all or most of it back into memory before you return.

SuperFetch also includes specific scenario support for hibernation, standby, Fast User Switching (FUS), and application launch. When the system hibernates, for example, SuperFetch stores data and code in the hibernation file that it expects (based on previous hibernations) will be accessed during the subsequent resume. In contrast, when you resume Windows XP, previously cached data must be reread from the disk when it is referenced.

See the sidebar "Watching SuperFetch" for a glimpse of how SuperFetch impacts available memory.



- Process Monitor: files used at application startup

Prefetch

- Prefetch files
 - C:\Windows\Prefetch
- Layout.ini

- Content of a prefetch file:
 - strings.exe

Summary

- Virtual memory, paging
- Multiple optimization
- Analysing memory usage
 - Task manager: quick summary
 - Process Explorer, Meminfo: details



To read

- [Windows XP Kernel Improvements](#): Prefetch
- [Inside the Windows Vista Kernel](#):
 - 1. part: Multimedia Class Scheduler
 - 2. part: Superfetch, Ready*
- [XP Myths](#): Prefetch and other misconceptions
- [MemInfo](#): Peer Inside Memory Manager Behavior on Windows Vista and Server 2008

