Windows Memory Management (I)

Memory Manager Concepts and Services

Roadmap for This Lecture

- Memory Manager Fundamentals
- Memory Management Services
- Virtual Address Space Layout
- Working Sets
- Kernel Mode Heaps
- Heap Manager
- Physical Memory Limits
- Memory management APIs

Introduction

- Main Tasks:
  - Translating (mapping) a process's virtual address space into physical memory, so running threads correctly reference physical address
  - Paging in and out contents between memory and disk when threads try to use more memory than currently available
- Additional Services:
  - Memory-mapped files (section object)
  - Copy-on-write memory
  - Large, sparse address space
  - Use of larger physical memory than max address space for a process (AWE)

Key Features

- Fully reentrant and provides synchronization to
  - Page frame number (PFN) database
  - Section objects and working sets
  - Page file creation
- Lazy evaluation
  - Sharing – usage of prototype PTEs (page table entries)
  - Extensive usage of copy-on-write
  - ...whenever possible
- Shares memory
  - Copy-on-write
  - Mapped files

Main Components

- System services for allocating, deallocating, and managing virtual memory
- An access fault trap handler for resolving hardware-detected memory management exceptions and making virtual pages resident on behalf of a process
- Six system threads
  - *Working set manager* (priority 16) – drives overall memory management policies, such as working set trimming, aging, and modified page writing
  - *Process/stack swapper* (priority 23) -- performs both process and kernel thread stack inswapping and outswapping
  - *Modified page writer* (priority 17) – writes dirty pages on the modified list back to the appropriate paging files
  - *Mapped page writer* (priority 17) – writes dirty pages from mapped files to disk
  - *Dereference segment thread* (priority 18) is responsible for cache and page file growth and shrinkage
  - *Zero page thread* (priority 0) – zeros out pages on the free list

Services

- Caller can manipulate own/remote memory
  - Parent process can allocate/deallocate, read/write memory of child process
  - Subsystems manage memory of their client processes this way
- Most services are exposed through Windows API
  - Page granularity virtual memory functions (Virtualxxx...)
  - Memory-mapped file functions (CreateFileMapping, MapViewofFile)
  - Heap functions (Heapxxx, Localxxx (old), Globalxxx (old))
- Services for device drivers/kernel code (Mm...)

Large Pages

Architecture	Small Page Size	Large Page Size
X86	4 KB	4 MB (2MB on PAE)
X64	4 KB	2 MB
IA64	8 KB	16 MB

- Virtual memory divided into pages – smallest unit of protection
- Two page sizes: small and large.
- Advantage of large page:
  - Faster address translation of references to other data in the same page
  - Single Translation Lookaside Buffer (TLB) entry used to map larger area
- Large pages are used to map NTOSKRNL, HAL, non-paged pool, and the PFN database of a “large memory system”

Large Pages

- Can specify other drivers to map with large pages:
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\LargePageDrivers (multi-string)
- Applications can use large pages for process memory
  - VirtualAlloc with MEM_LARGE_PAGE flag
  - Can query if system supports large pages with *GetLargePageMinimum*

Large Pages

- Disadvantage of large pages:
  - disables kernel write protection
  - With small pages, OS/driver code pages are mapped as read only; with large pages, entire area must be mapped read/write
  - Drivers can then modify/corrupt system & driver code without immediately crashing system
- Driver Verifier turns large pages off
- Can also override by changing HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\LargePageMinimum to FFFFFFFF

Protecting Memory

Attribute	Description
PAGE_NOACCESS	Read/write/execute causes access violation
PAGE_READONLY	Write/execute causes access violation; read permitted
PAGE_READWRITE	Read/write accesses permitted
PAGE_EXECUTE	Any read/write causes access violation; execution of code is permitted (relies on special processor support)
PAGE_EXECUTE_READ	Read/execute access permitted (relies on special processor support)
PAGE_EXECUTE_READWRITE	All accesses permitted (relies on special processor support)
PAGE_WRITECOPY	Write access causes the system to give process a private copy of this page; attempts to execute code cause access violation
PAGE_EXECUTE_WRITECOPY	Write access causes creation of private copy of page.
PAGE_GUARD	Any read/write attempt raises EXCEPTION_GUARD_PAGE and turns off guard page status

Reserving & Committing Memory

- Optional 2-phase approach to memory allocation:
  1. Reserve address space (in multiples of page size)
  2. Commit storage in that address space
  - Can be combined in one call (VirtualAlloc, VirtualAllocEx)
- Reserved memory:
  - Range of virtual addresses reserved for future use (contiguous buffer)
  - Accessing reserved memory results in access violation
  - Fast, inexpensive
- Committed memory:
  - Has backing store (pagefile.sys, memory-mapped file)
  - Either private or mapped into a view of a section
  - Decommit via VirtualFree, VirtualFreeEx

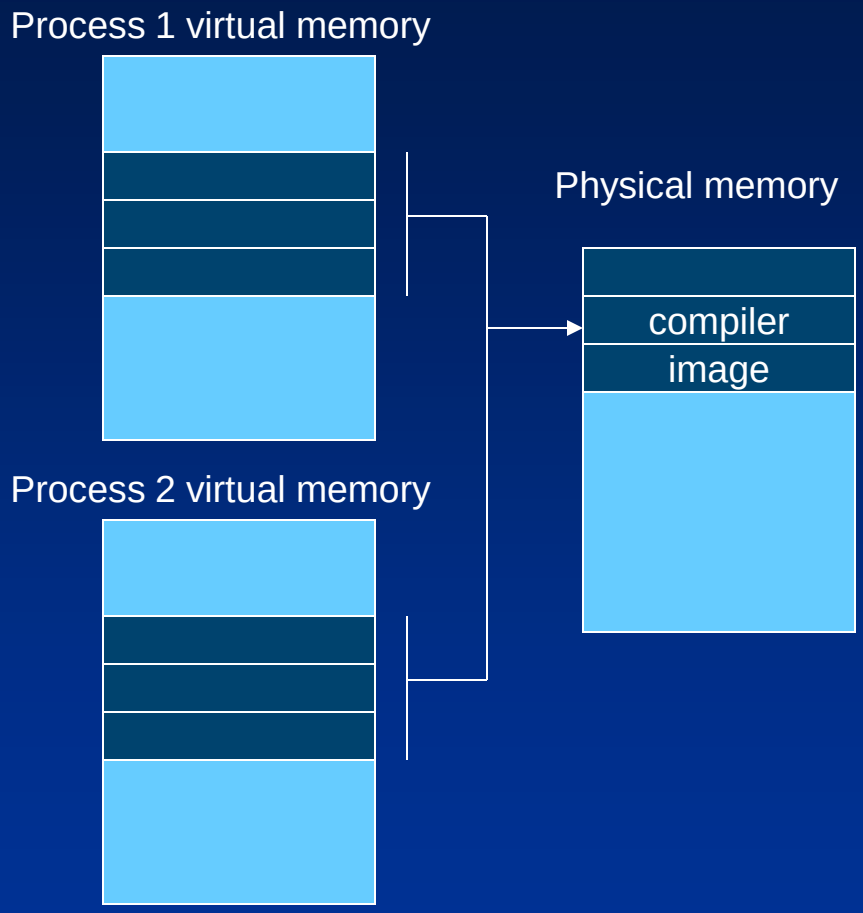
A thread‘s user-mode stack is constructed using this 2-phase approach: initial reserved size is 1MB, only 2 pages are committed: stack & guard page

Allocation Granularity

- Each region of reserved process address space begin on multiple of *allocation granularity*
- *GetSystemInfo* or *GetNativeSystemInfo* function
- Value is 64 KB
- Size and base of each reserved memory region is multiple of the system page size (e.g. 4 KB on x86 systems)

Shared Memory

- Like most modern OS's, Windows provides a way for processes to share memory
  - High speed IPC (used by ALPC, which is used by RPC)
  - Threads share address space, but applications may be divided into multiple processes for stability reasons
- It does this automatically for shareable pages
  - E.g. code pages in an EXE or DLL
- Processes can also create shared memory sections
- Underlying primitives for sharing is "section objects", or "file mapping objects" in Windows API
- Section object connected to open file → *mapped file (or file backed section)*
- Section object connected to committed memory → *page file backed section*



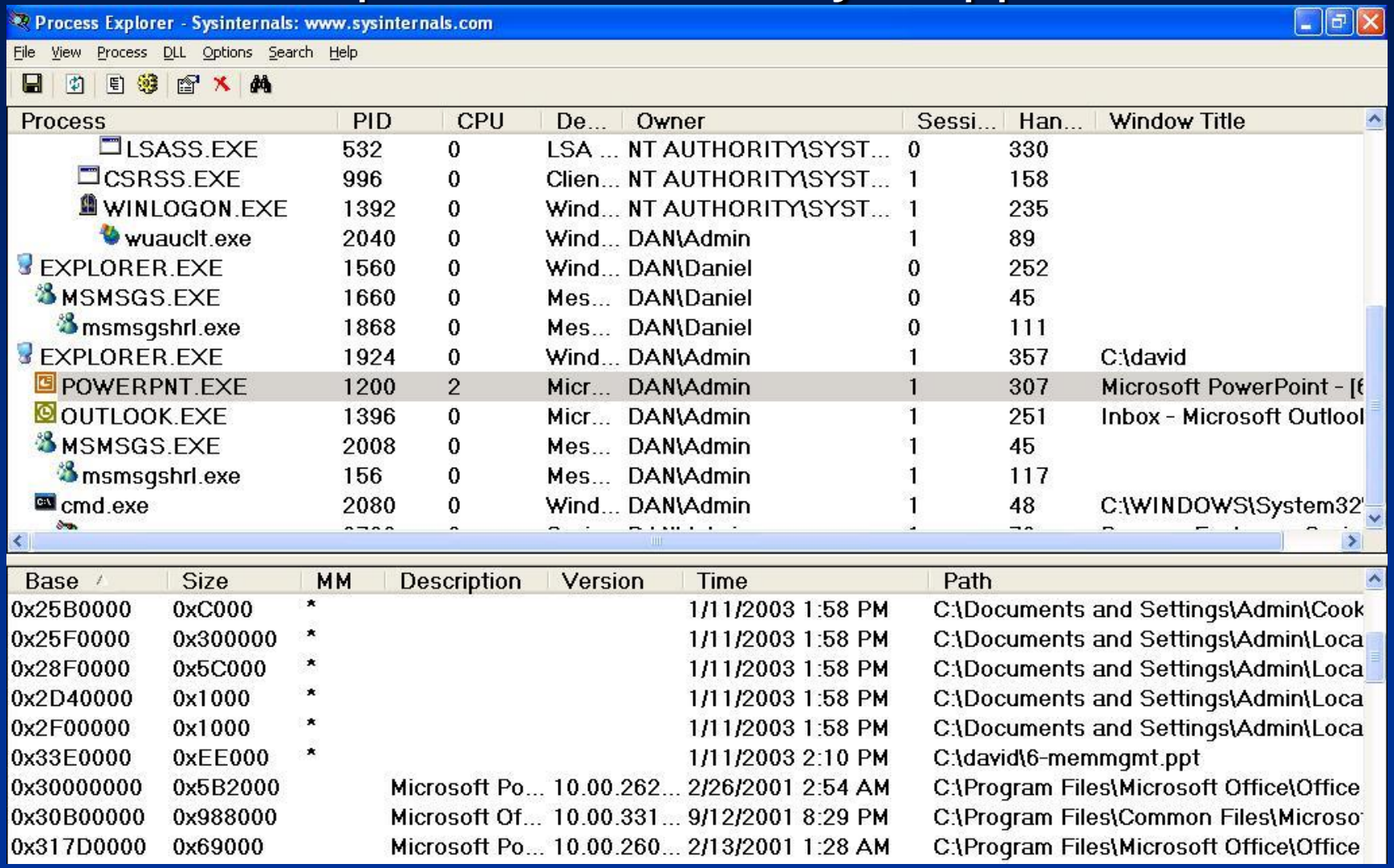
Mapped Files

- A way to take part of a file and map it to a range of virtual addresses

  (address space is 2 GB, but files can be much larger)
- Bytes in the file then correspond one-for-one with bytes in the region of virtual address space
  - Read from the “memory” fetches data from the file
  - Pages are kept in physical memory as needed
  - Changes to the memory are eventually written back to the file (can request explicit flush)
- Initial mapped files in a process include:
  - The executable image (EXE)
  - One or more Dynamically Linked Libraries (DLLs)
- Processes can map additional files as desired (data files or additional DLLs)

Viewing DLLs & Memory Mapped Files

- Process Explorer lists memory mapped files



Data Execution Prevention

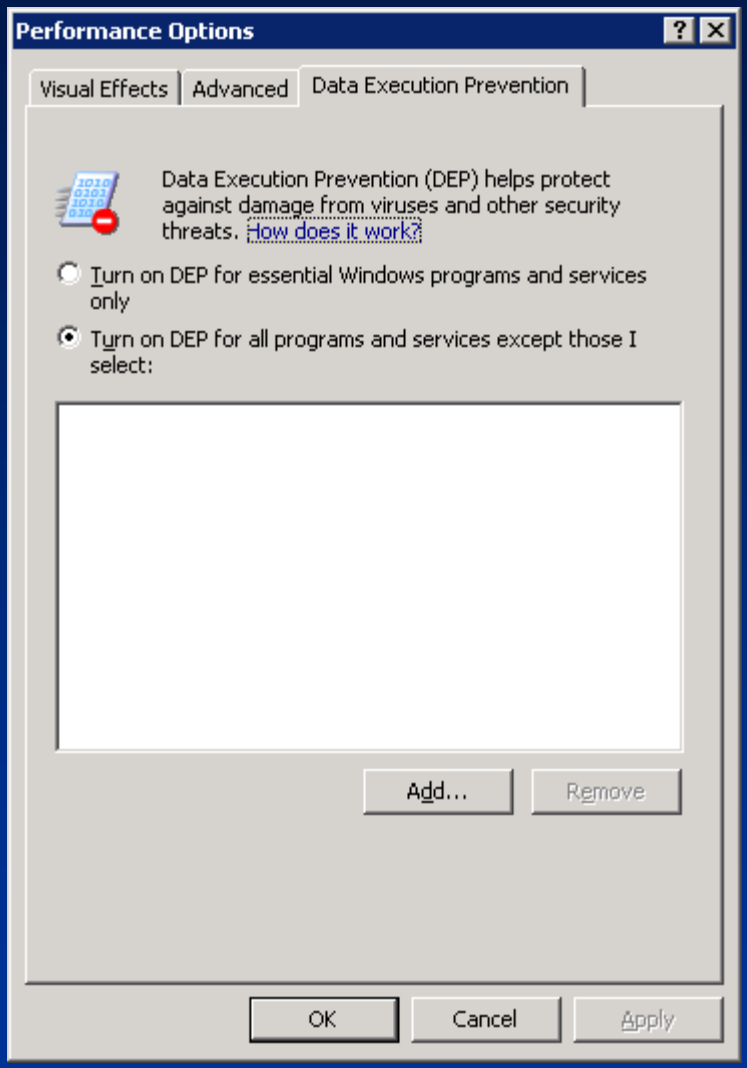
- Purposes
  - Prevents code from executing in a memory page not specifically marked as executable
  - Stops exploits that rely on getting code executed in data areas
- Relies on hardware ability to mark pages as non executable
  - Setting BCD nx
- Attempts to execute code in a page marked no execute result in:
  - User mode: access violation exception
  - Kernel mode: ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY bugcheck (blue screen)
- Memory that needs to be executable must be marked as such using page protection bits on VirtualAlloc and VirtualProtect APIs:
  - PAGE_EXECUTE, PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE, PAGE_EXECUTE_WRITECOPY

Controlling DEP

- BCD nx values:
  - OptIn: Turn on DEP for necessary Windows programs and services only
  - OptOut: Turn on DEP for all programs and services except the ones selected
  - AlwaysOn: Enables DEP for all components with no ability to exclude certain applications
  - AlwaysOff: Disables DEP (not recommended)

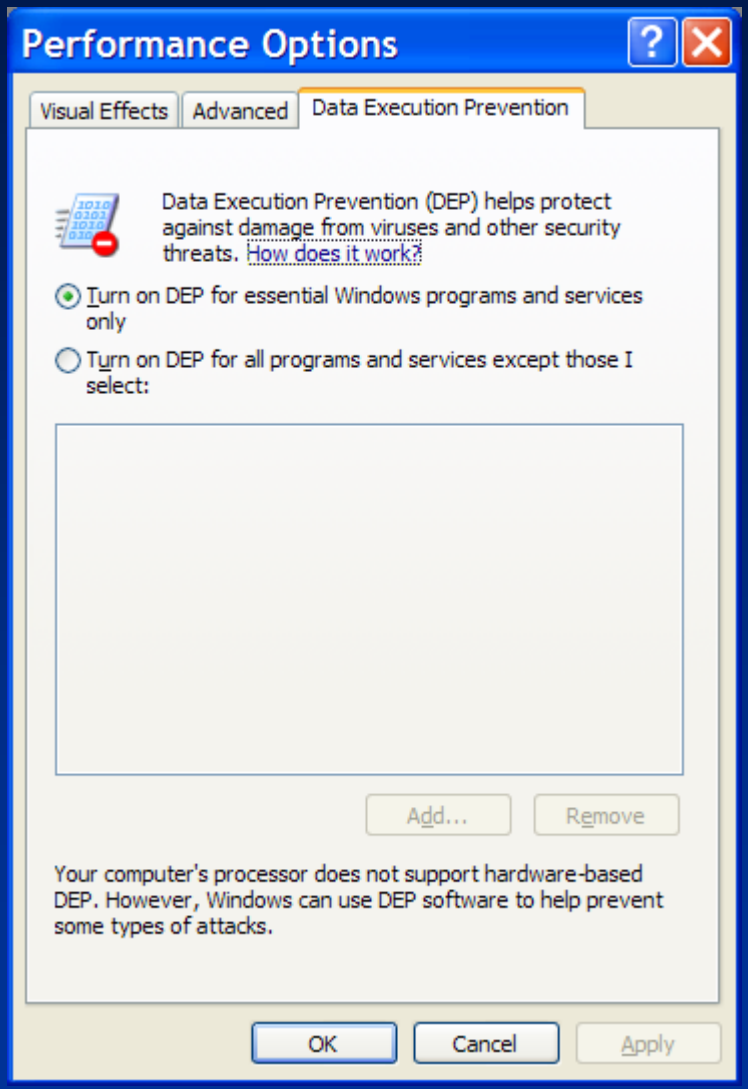
DEP on 64-bit Windows

- Always applied to all 64-bit processes and device drivers
  - Protects user and kernel stacks, paged pool, session pool
- 32-bit processes depend on configuration settings



DEP on 32-bit Windows

- Hardware DEP used when running 32-bit Windows on systems that support it
- When enabled, system boots PAE kernel (Ntkrnlpa.exe)
- Kernel mode: applied to kernel stacks, but not paged/session pool
- User mode: depends on system configuration
- Even on processors without hardware DEP, some limited protection implemented for exception handlers



Software DEP

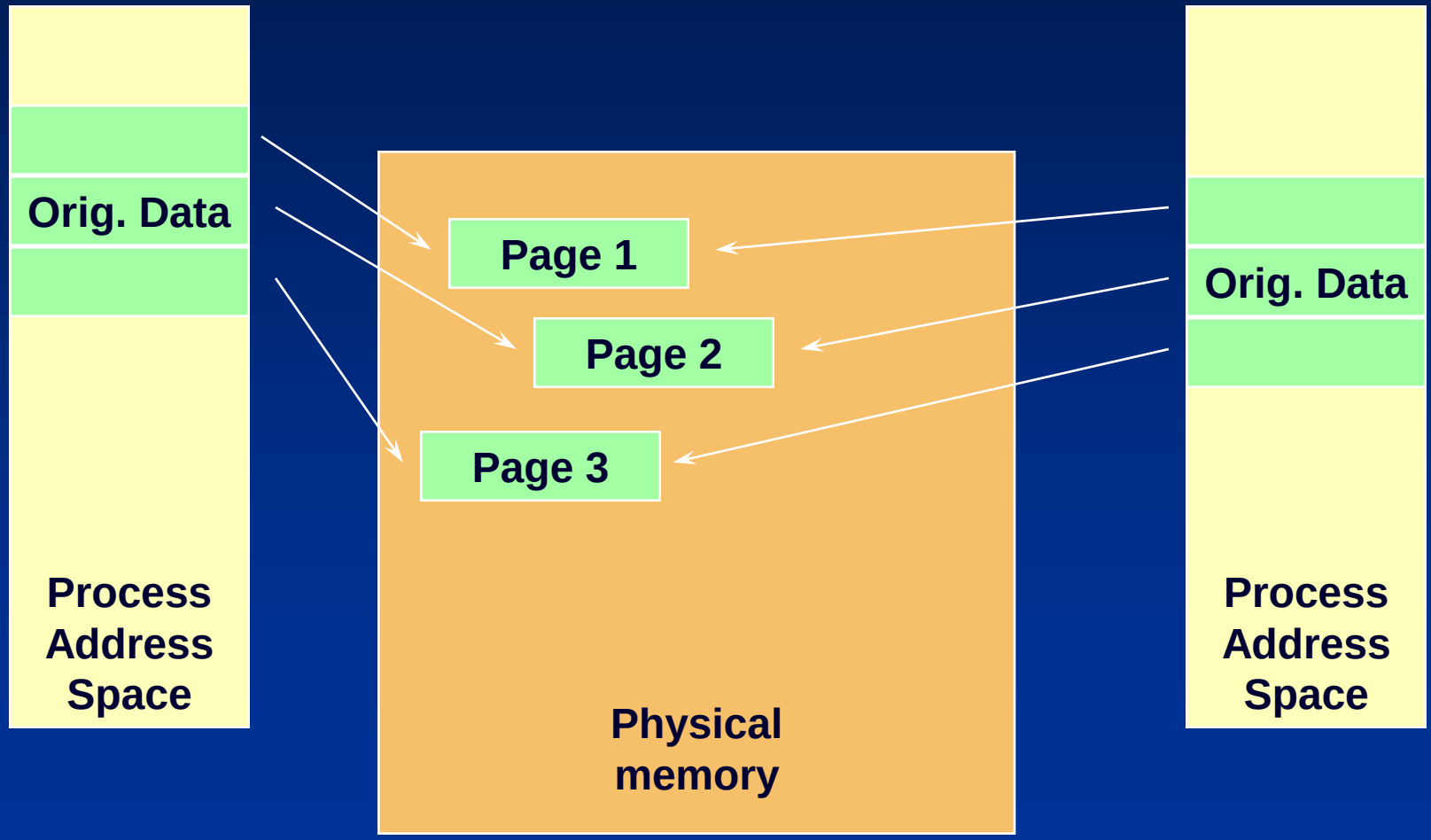
- Older processors which don't support hardware "no execution" protection
- Exception handling mechanism
  - image built with safe structure exception handling → verify that exception is registered in the function table
  - Not built with safe structure exception handling → exception handler is located in the memory region marked as executable
- Stack cookies
  - Special code at the beginning and end of function
  - Compare cookie values
- Pointer encoding
  - Encoding with cookie value
  - Corrupt decoded pointer crashes program

Copy-On-Write Pages

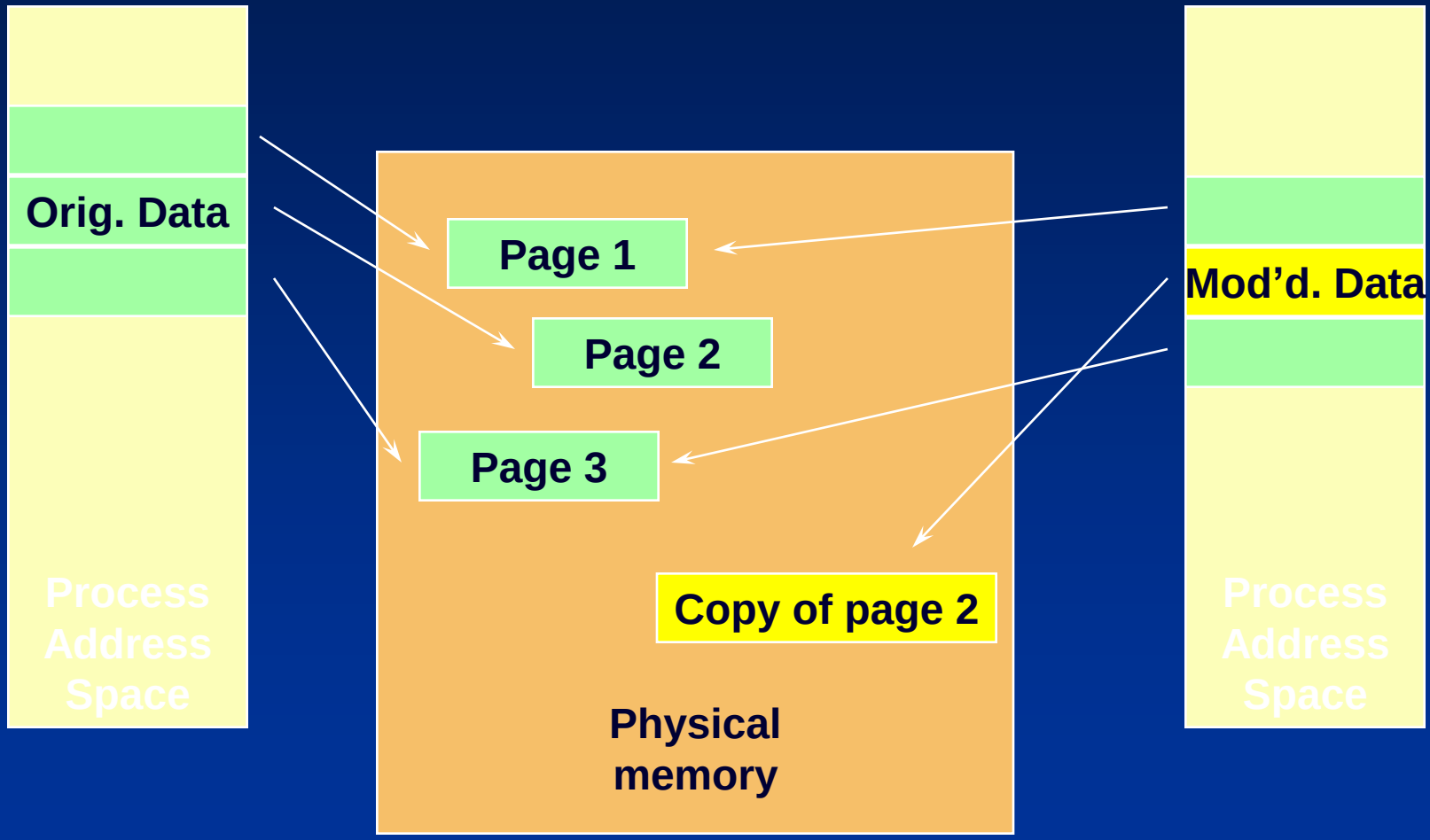
- Used for sharing between process address spaces
- Pages are originally set up as shared, read-only, faulted from the common file
  - Access violation on write attempt alerts pager
    - pager makes a copy of the page and allocates it privately to the process doing the write, backed to the paging file
  - So, only need unique copies for the pages in the shared region that are actually written (example of “lazy evaluation”)
  - Original values of data are still shared
    - e.g. writeable data initialized with C initializers

How Copy-On-Write Works

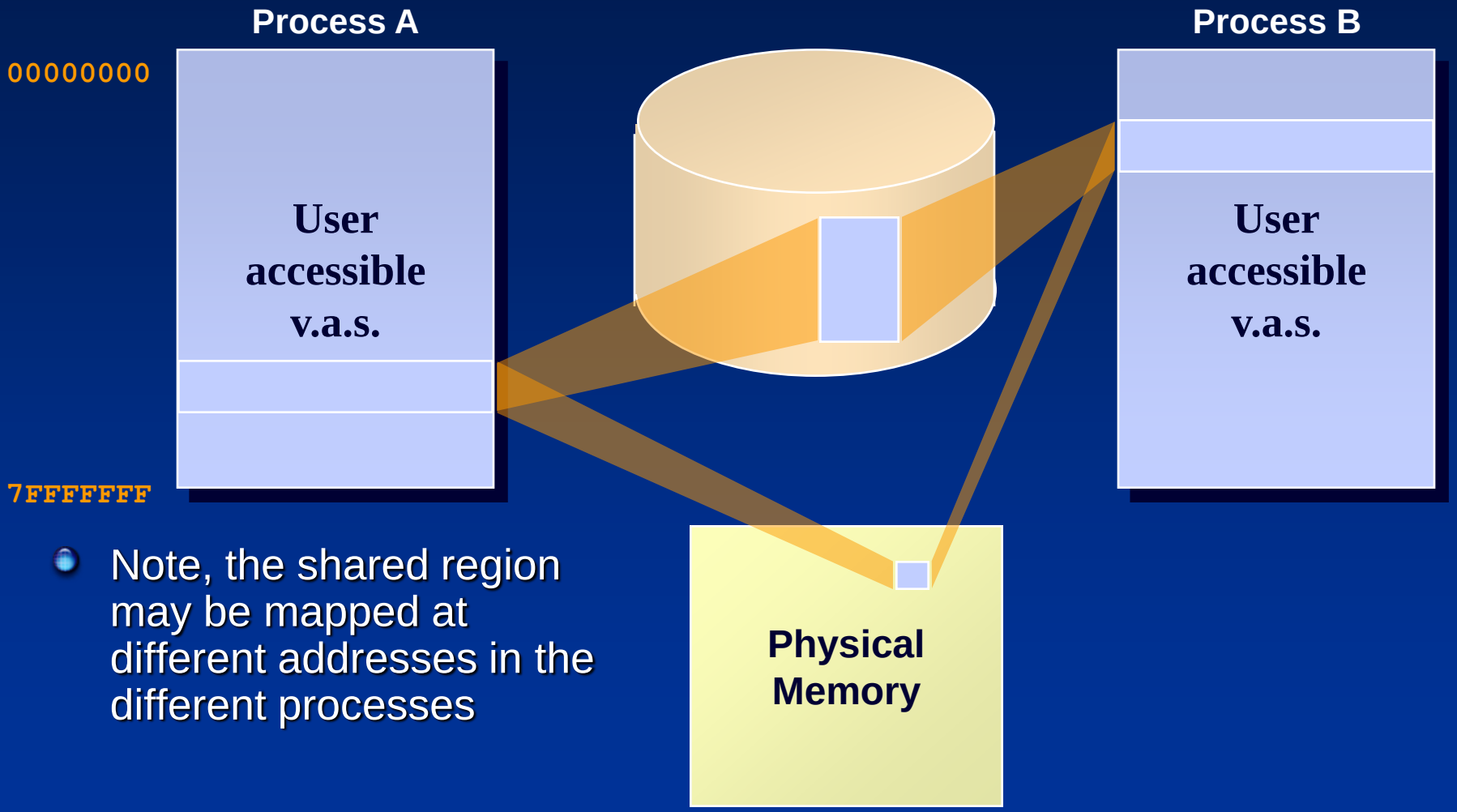
Before



How Copy-On-Write Works
After



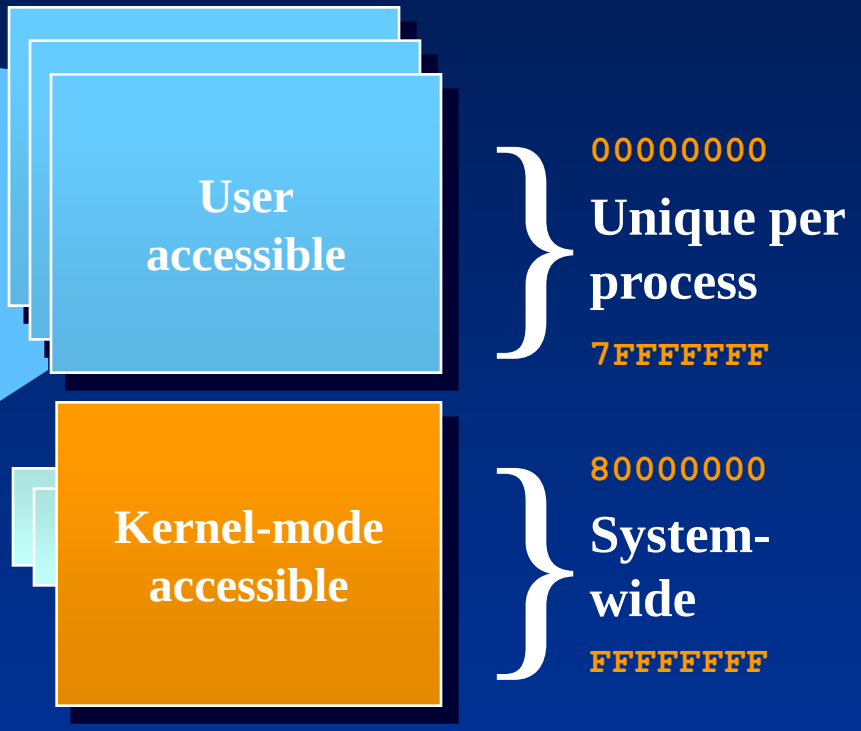
Shared Memory = File Mapped by Multiple Processes



- Note, the shared region may be mapped at different addresses in the different processes

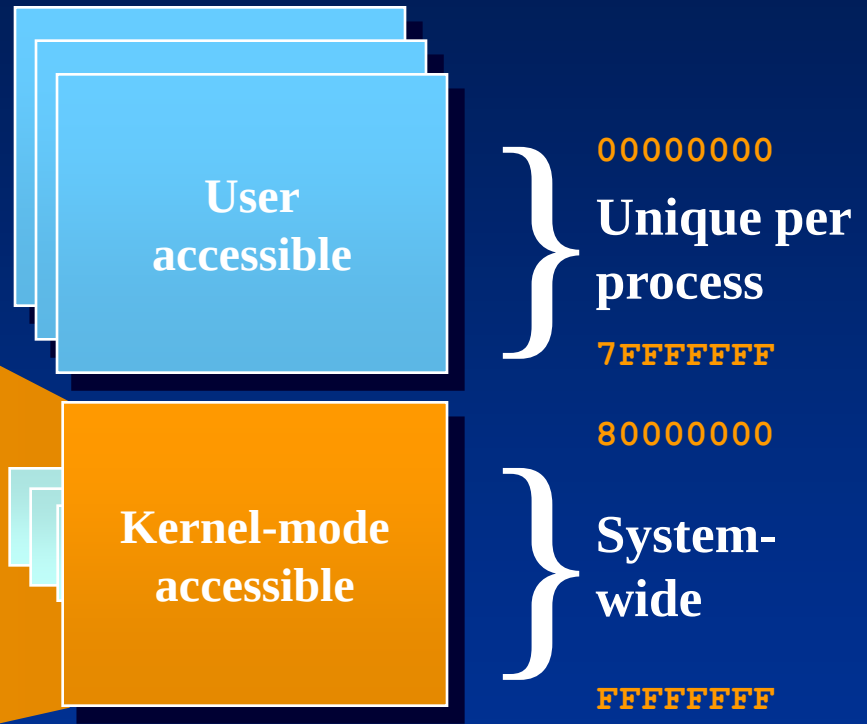
Virtual Address Space (V.A.S.)

- Process space contains:
  - The application you're running (.EXE and .DLLs)
  - A user-mode stack for each thread (automatic storage)
  - All static storage defined by the application



Virtual Address Space (V.A.S.)

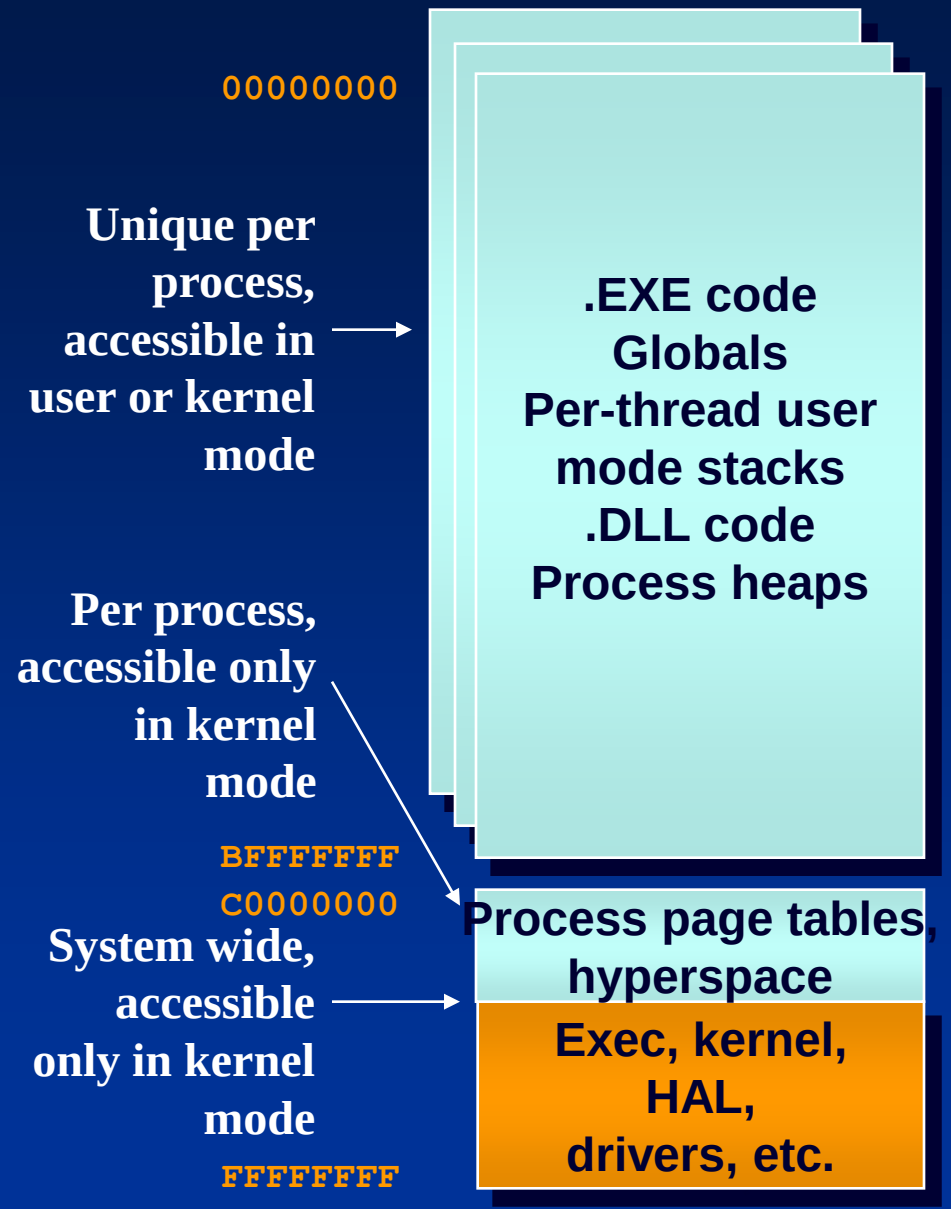
- System space contains:
  - Executive, kernel, and HAL
  - Statically-allocated system-wide data cells
  - Page tables (remapped for each process)
  - Executive heaps (pools)
  - Kernel-mode device drivers (in nonpaged pool)
  - File system cache
  - A kernel-mode stack for every thread in every process



Windows User Process Address Space Layout

Range	Size	Function
0x0 – 0xFFFF	64 KB	No-access region to catch incorrect pointer ref.
0x10000 - 07FFEFFF	2 GB minus at least 192kb	The private process address space
0x7FFDE000 - 0x7FFDEFFF	4 KB	Thread Environment Block (TEB) for first thread, more TEBs are created at the page prior to that page
0x7FFDF000 - 0x7FFDFFFF	4 KB	Process Environment Block (PEB)
0x7FFE0000 - 0x7FFE0FFF	4 KB	Shared user data page – read-only, mapped to system space, contains system time, clock tick count, version number (avoid kernel-mode transition)
0x7FFE1000 – 0x7FFEFFFF	60 KB	No-access region
0x7FFF0000 – 0x7FFFFFFF	64 KB	No-access region to prevent threads from passing buffers that straddle user/system space boundary

3GB Process Space Option



- Only available on:
  - Windows 2003 Server, Enterprise Edition & Windows 2000 Advanced Server, XP SP2
    - Limits phys memory to 16 GB
  - /3GB option in BOOT.INI
  - Windows Server 2003 and XP SP2 supports variations from 2GB to 3GB (/USERVA=)
- Provides 3 GB per-process address space
  - Commonly used by database servers (for file mapping)
  - .EXE must have “large address space aware” flag in image header, or they’re limited to 2 GB (specify at link time or with imagecfg.exe from ResKit)
  - Chief “loser” in system space is file system cache
  - Better solution: *address windowing extensions*
  - Even better: 64-bit Windows

Large Address Space Aware Images

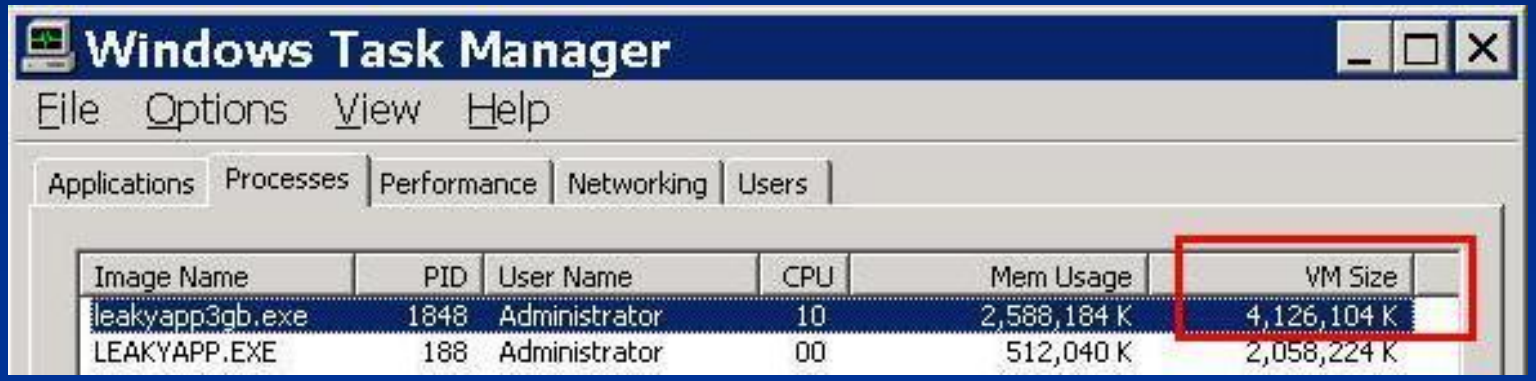
- Images marked as “large address space aware”:
  - Lsass.exe – Security Server
  - Inetinfo.exe—Internet Information Server
  - Chkdsk.exe – Check Disk utility
  - Dllhst3g.exe – special version of Dllhost.exe (for COM+ applications)
  - Esentutl.exe - jet database repair tool
- To see this, type:

```
Imagecfg \windows\system32\*.exe > large_images.txt
```

  - Then search for “large” in large_images.txt

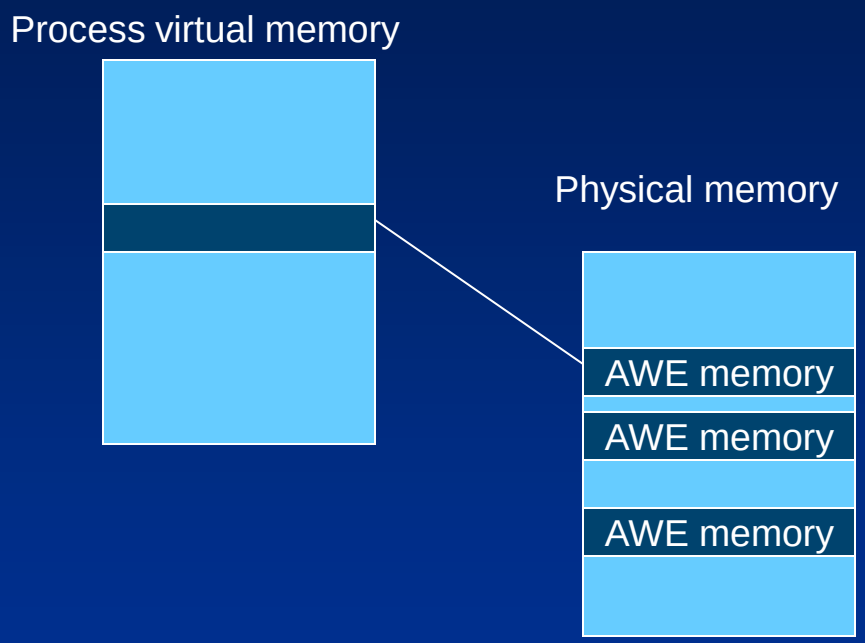
Large Address Space Aware on 64-bits

- Images marked large address space aware get a full 4 GB process virtual address space
  - OS isn't mapped there, so space is available for process



Address Windowing Extensions

- 32-bit user process by default has 2 GB virtual address space
- AWE functions allow Windows processes to allocate large amounts of physical memory and then map “windows” into that memory
- Applications: database servers can cache large databases
- Up to programmer to control
- AWE memory never paged out → a security feature



Steps:

1. Allocate physical mem to be used
2. Create a region of v.a.s. to act as a window to map views of physical mem
3. Map views of the physical mem into window

Working Sets

- Working Set:
  - The set of pages in memory at any time for a given process, or
  - All the pages the process can reference without incurring a page fault
  - Per process, private address space
  - WS limit: maximum amount of pages a process can own
  - Implemented as array of working set list entries (WSLE)
- Soft vs. Hard Page Faults:
  - Soft page faults resolved from memory (standby/modified page lists)
  - Hard page faults require disk access
- Working Set Dynamics:
  - Page replacement when physical memory limit reached and page fault happens
  - Replacement policies:
    - First-in-first (FIFO)
    - Least Recently Used (LRU)

Working Set Management

- Default working set size: min 50 pages, max 345 pages
- Memory manager examines working set limits and amount of free memory when page fault occurs
- Working set manager initiates automatic working set trimming when physical memory runs low
- Balance Set Manager thread
  - Created at system initialization
  - Wakes up every second
  - Executes MmWorkingSetManager
  - Trimming process WS when required: from current down to minimal WS for processes with lowest page fault rate
  - Aware of the system cache working set
  - Process can be out-swapped if all threads have pageable kernel stack

Kernel-Mode Heaps (System Memory Pools)

- Nonpaged pool
  - Reside in physical memory at all times
  - No page faults
  - Page faults can't be satisfied at DPC or above
- Paged pool
  - Can page in and out of the physical memory
  - Device drivers that don't need to access memory from DPC and above

Pool type	Max on 32-bit	Max on 64-bit
Non-paged	75% of physical mem or 2 GB (whichever smaller)	75% of physical memory or 128 GB (whichever smaller)
Paged	2 GB	128 GB

Look-Aside Lists

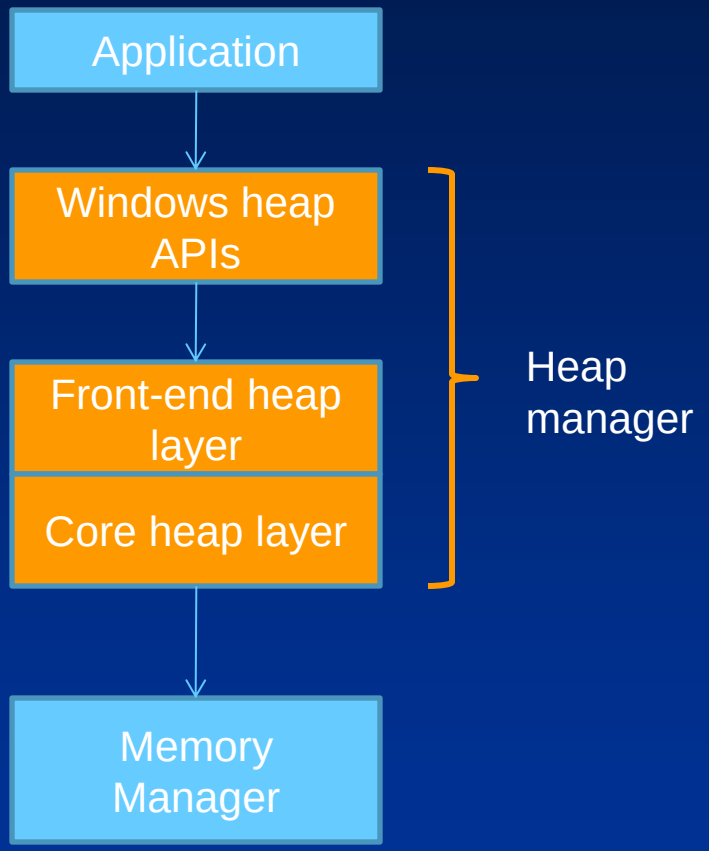
- Another fast memory allocation mechanism
- Contains only fixed-size blocks
- Faster → no spinlocks
- Executive components and device drivers use *ExInitializeNPagedLookasideList* to create look-aside lists that match frequently allocated data structures
- Size adjusted by balance set manager once a second

Heap Manager

- Most apps allocates smaller blocks than 64KB
- Heap manager manages allocations *inside* the larger memory areas reserved by *VirtualAlloc* function
- Allocation granularity:
  - 32-bit: 8 bytes
  - 64-bit: 16 bytes
- Designed to optimize memory usage and performance with these smaller allocations
- Exists in Ntdll.dll and Ntoskrnl.exe
- Each process has a default process heap:
  - 1 MB but expandable

Heap Manager Structure

- Two layers:
  - Core heap
  - Optional front-end layer
- Core heap
  - Common across user-mode and kernel mode
  - Manages blocks inside segments, segments, policies for extending heap, committing and decommitting memory and large blocks
- Front-end (for user mode only)
  - Low fragmentation heap (LFH)
  - Avoids fragmentation by allocating blocks according to bucket sizes
  - More popular allocation classes performed by LFH
  - Can be disabled by *HeapSetInformation* API



Physical Memory

- Amount of physical memory affects performance
- Limit other resources e.g. nonpaged pools, OS buffers backed by physical memory
- System virtual memory limit → sum of roughly the size of physical memory and configured size of paged files
- 32-bit limit is 64GB:
  - Data structure that keeps track of physical memory (e.g. PFN database) grows with the physical memory
  - Large system would consume too much of virtual address
  - Mapping pieces of PFN database into the system address as needed → too much complexity and overhead with mapping, unmapping and locking
- 2 TB limit on 64-bit windows not hardware/implementation limit:
  - largest test Itanium system was 2TB

Physical Memory Limits

	32-Bit	64-Bit	Limiting Factor
Windows Server 2008 Data center and enterprise	64 GB	2 TB on IA64 1 TB on x64	PFN database size on 32-bit, hardware platform on 64-bit
Windows Server 2008 Standard and Web server	4 GB	32 GB	Licensing
Windows Server 2008 HPC Edition	N/A	128 GB	Licensing
Windows Vista Ultimate, Enterprise, Business	4 GB	128 GB	Licensing on 64-bit, hardware support and driver compatibility on 32-bit
Windows Vista Home Premium	4 GB	16 GB	Licensing on 64-bit, hardware support and driver compatibility on 32-bit
Windows Vista Home Basic	4 GB	8 GB	Licensing on 64-bit, hardware support and driver compatibility on 32-bit

I/O Support

- I/O Support operations:
  - Locking/Unlocking pages in memory
  - Mapping/Unmapping Locked Pages into current address space
  - Mapping/Unmapping I/O space
  - Get physical address of a locked page
  - Probe page for access
- Memory Descriptor List
  - Starting VAD
  - Size in Bytes
  - Array of elements to be filled with physical page numbers
- Physically contiguous vs. Virtually contiguous

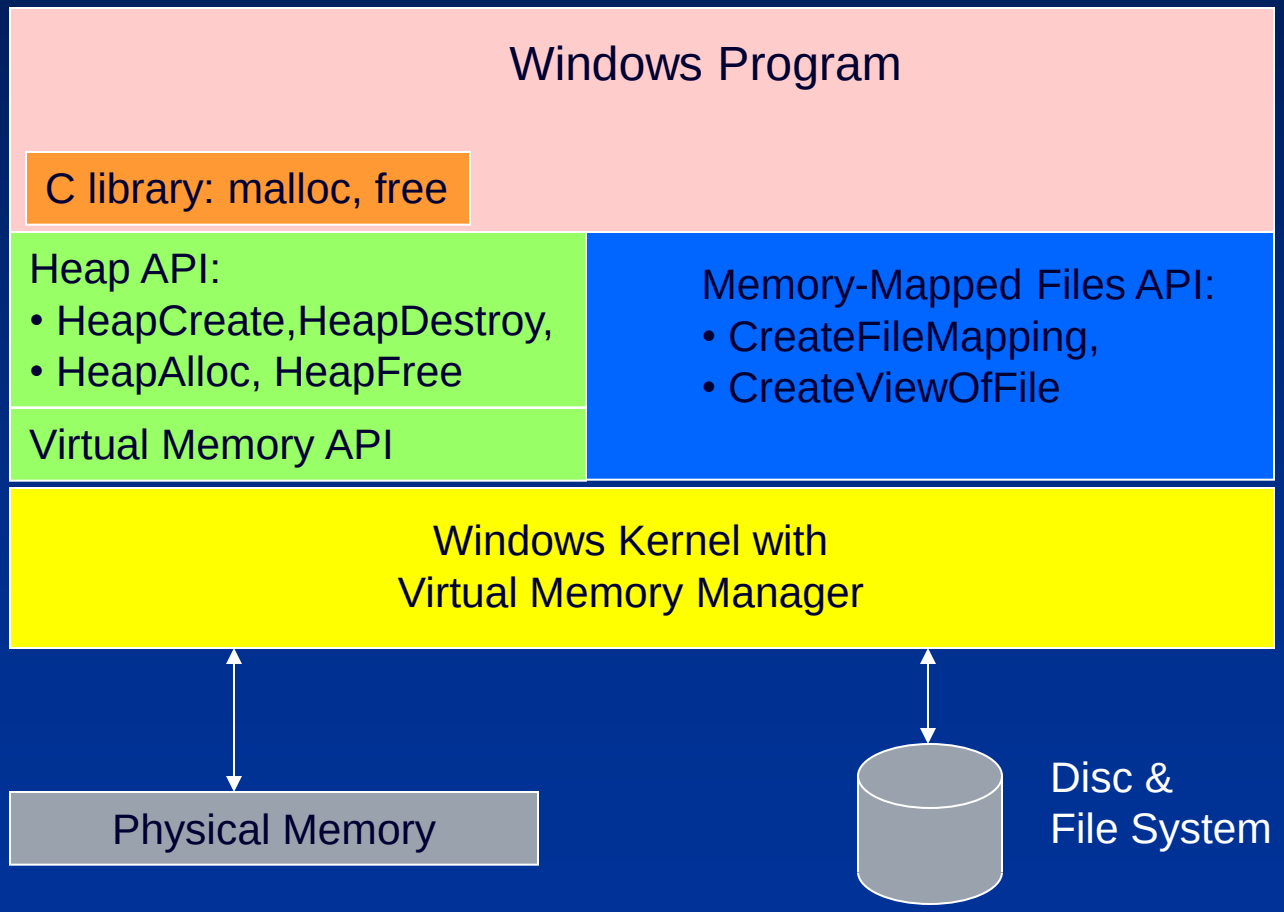
Cache Support

- System wide cache memory
  - Region of system paged area reserved at initialization time
  - Initial default: 512 MB (min. 64MB if /3GB, max 960 MB)
  - Managed as system wide working set
    - A valid cache page is valid in all address spaces
    - Lock the page in the cache to prevent WS removal
  - WS Manager trimming thread is aware of this special WS
  - Not accessible from user mode
  - Only views of mapped files may reside in the cache
- File Systems and Server interaction support
  - Map/Unmap view of section in system cache
  - Lock/Unlock pages in system cache
  - Read section file in system cache
  - Purge section

Windows Memory Allocation APIs

- HeapCreate, Alloc, etc. (process heap APIs)
  - Windows equivalent of malloc(), free(), etc.
- VirtualAlloc(MEM_RESERVE)
- VirtualAlloc(MEM_COMMIT)
- VirtualFree
- VirtualQuery

Windows API Memory Management Architecture



Further Reading

- Mark E. Russinovich *et al.* Windows Internals, 5th Edition, Microsoft Press, 2009.
  - Chapter 9 - Memory Management
  - Introduction to memory manager (from pp.699)
  - Working sets (from pp. 822)
  - Services the Memory Manager Provides (from pp. 704)
  - Kernel-mode Heaps (from pp. 721)
  - Heap manager (from pp. 729)
  - Physical memory limits (from pp. 818)

Source Code References

- Windows Research Kernel sources
  - \base\ntos\mm – Memory manager
  - \base\ntos\inc\mm.h – additional structure definitions
  - \base\ntos\cache – Cache manager

Lab Demo: View & Create & Use DLL

2013-10-12

Content

- View DLLs using “*listdll.exe*” from *sysinternals*
- Create your own DLL
- Using your own DLL