Unit OS5: Memory Management

5.3. Windows Memory Management Operation

Copyright Notice

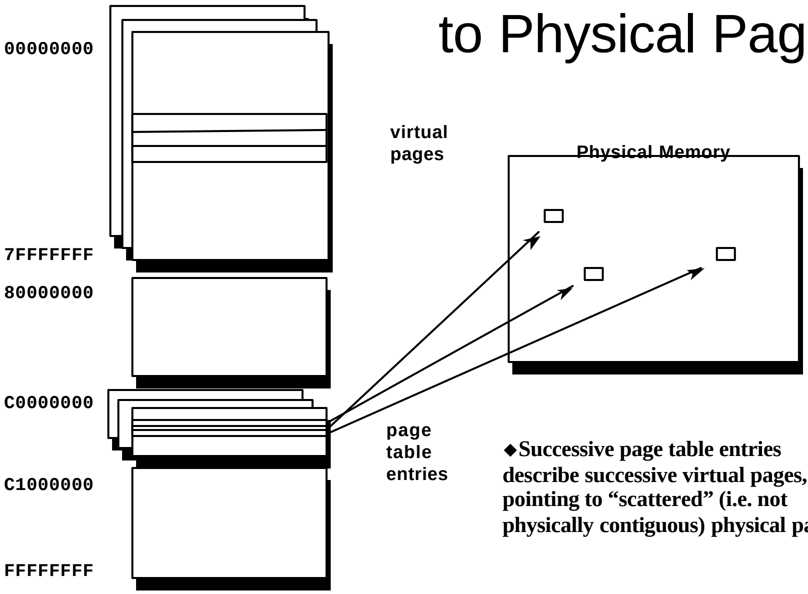
Roadmap for Section 5.3.

- From virtual to physical addresses
- Address space layout
- Address translation
- Page directories, page tables
- Page faults, invalid page table entries
- Page frame number database
- Recap: structuring of the memory manager

Virtual Memory - Concepts

- Application always references “virtual addresses”
- Hardware and software translates, or *maps,* virtual addresses to physical
- Not all of an application’s virtual address space is in physical memory at one time...
  - ...But hardware and software fool the application into thinking that it is
  - The rest is kept on disk, and is brought into physical memory automatically as needed

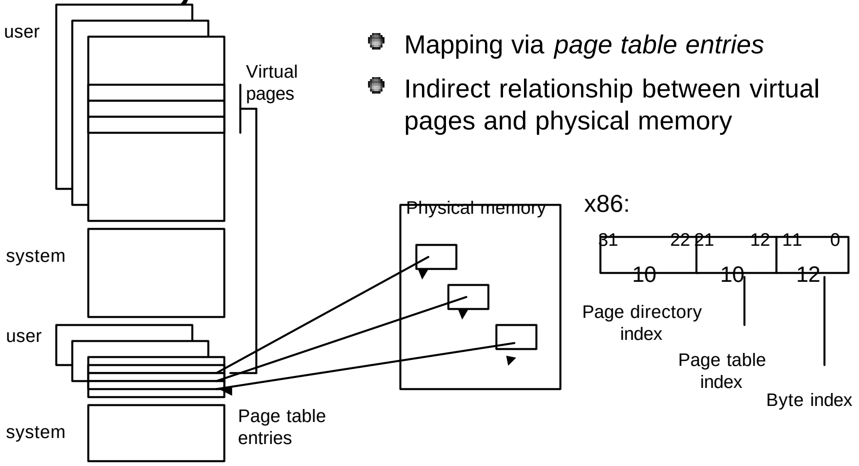
Mapping Virtual to Physical Pages



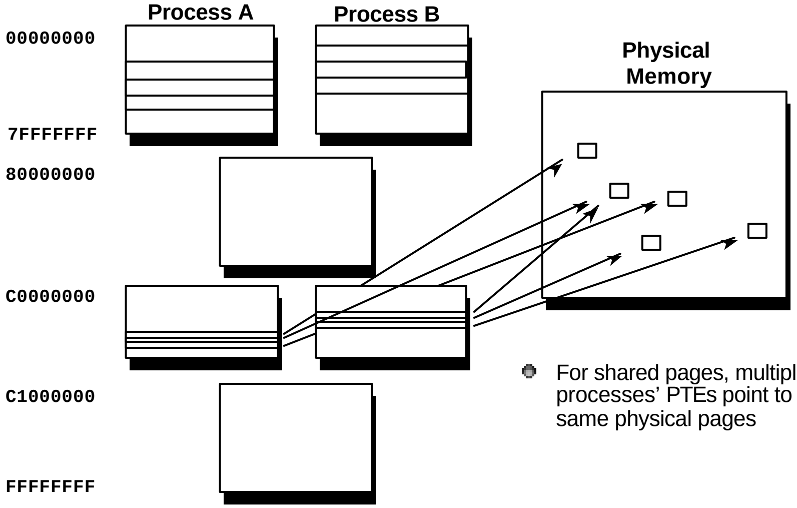
◆**Successive page table entries describe successive virtual pages, pointing to "scattered" (i.e. not physically contiguous) physical pages**

Address Translation - Mapping virtual addresses to physical memory

- Mapping via *page table entries*
- Indirect relationship between virtual pages and physical memory



Shared and Private Pages

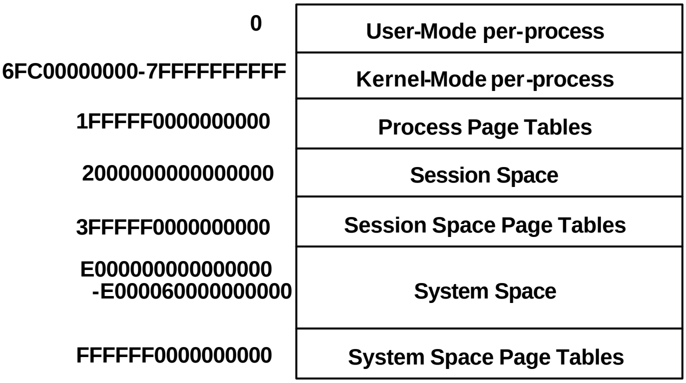


- For shared pages, multiple processes’ PTEs point to same physical pages

Windows 2000 x86 System Space Layout

80000000	System code (NTOSKRNL, HAL, boot drivers); initial nonpaged pool
A0000000	System Mapped Views (e.g. WIN32K.SYS) or session space (Terminal Server only)
A4000000	Additional System PTEs (& big cache)
C0000000	Process Page Tables and Page Directory
C0400000	Hyperspace and process working set list
C0800000	Unused No Access
C0C00000	System Working Set List
C1000000	System Cache
E1000000	Paged Pool
EB000000 (min)	System PTEs
	Non-Paged Pool expansion
FFBE0000	Crash dump information
FFC00000	HAL usage

64-Bit Address Space Layout (Itanium)

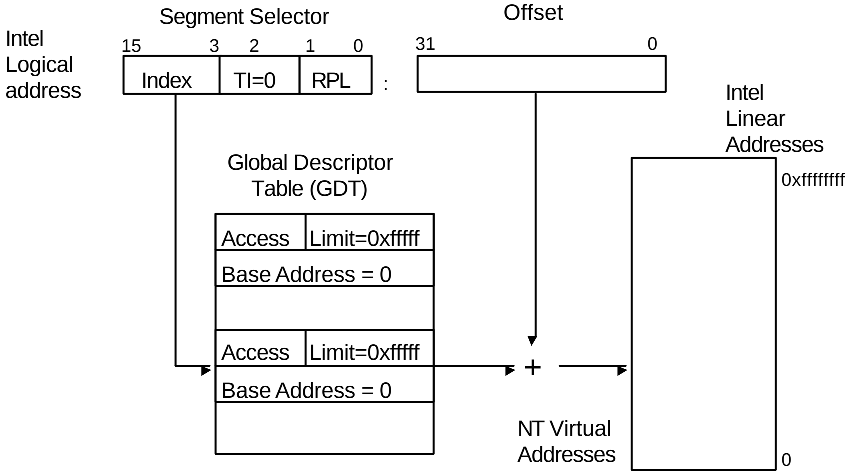


Address	Region
0	User-Mode per-process
6FC00000000-7FFFFFFFFFF	Kernel-Mode per-process
1FFFFF0000000000	Process Page Tables
2000000000000000	Session Space
3FFFFF0000000000	Session Space Page Tables
E000000000000000 -E000060000000000	System Space
FFFFFF0000000000	System Space Page Tables

Address Translation Hardware Support Intel x86

- Intel x86 provides two levels of address translation
  - Segmentation (mandatory, since 8086)
  - Paging (optional, since 80386)
- Segmentation: first level of address translation
  - Intel: logical address (selector:offset) to linear address (32 bits)
  - NT virtual address is Intel linear address (32 bits)
- Paging: second level of address translation
  - Intel: linear address (32 bits) to physical address
  - NT: virtual address (32 bits) to physical address
  - Physical address: 32 bits (4 GB) all NT versions, 36 bits (64 GB) PAE
  - Page size:
    - 4 kb since 80386 (all NT versions)
    - 4 MB since Pentium Pro (supported in NT 4, Windows 2000)

Intel x86 Segmentation



Segment Selector

Offset

Intel Logical address

15 3 2 1 0

Index | TI=0 | RPL

:

31 0

Intel Linear Addresses

Global Descriptor Table (GDT)

0xffffffff

Access | Limit=0xfffff

Base Address = 0

Access | Limit=0xfffff

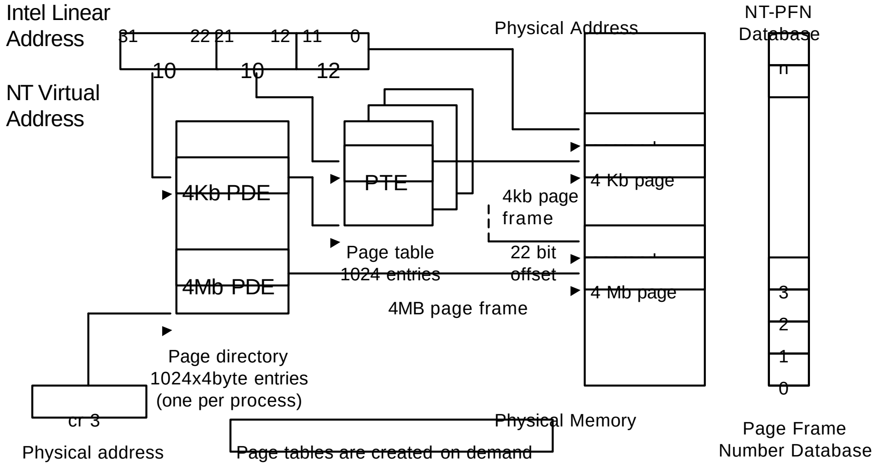
Base Address = 0

+

NT Virtual Addresses

0

Intel x86 Paging – Address Translation



Intel Linear Address

31 22 21 12 11 0

10 10 12

NT Virtual Address

Physical Address

NT-PFN Database

n

4Kb PDE

PTE

4 Kb page

4kb page frame

Page table 1024 entries

22 bit offset

4Mb PDE

4 Mb page

4MB page frame

3

2

1

0

Page directory 1024x4byte entries (one per process)

cr 3

Physical address

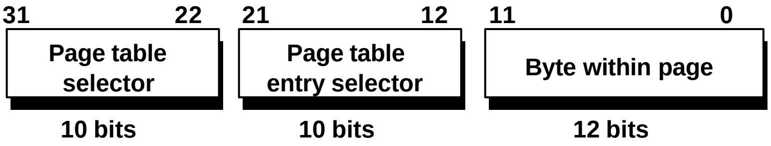
Physical Memory

Page tables are created on demand

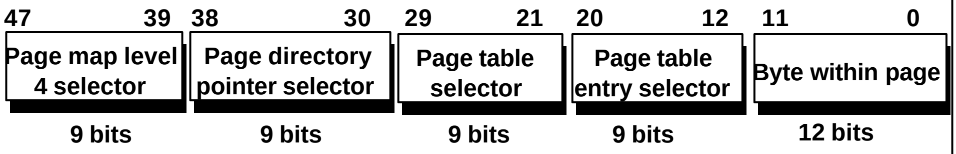
Page Frame Number Database

Interpreting a Virtual Address

x86 32-bit



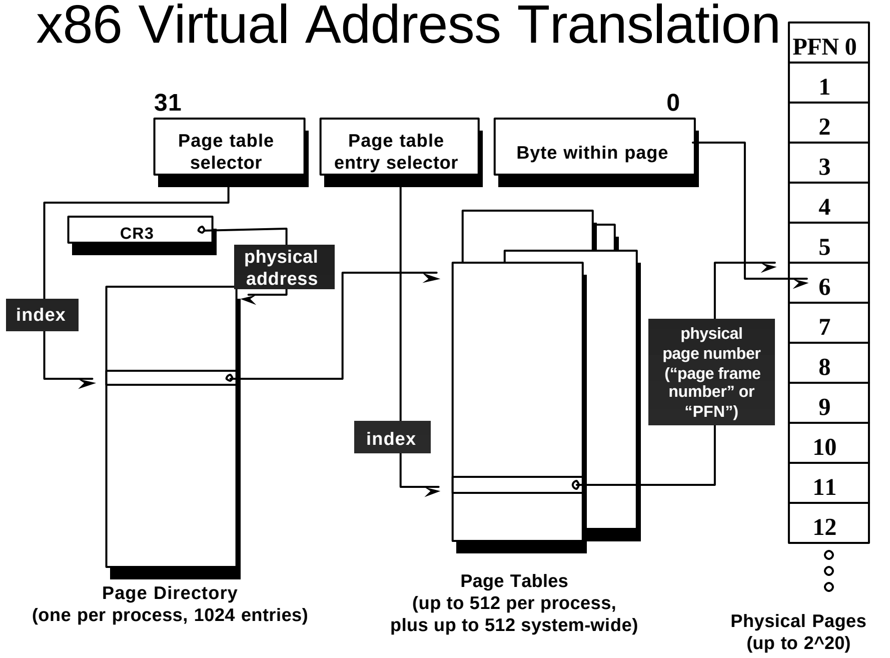
x64 64-bit (48-bit in today's processors)



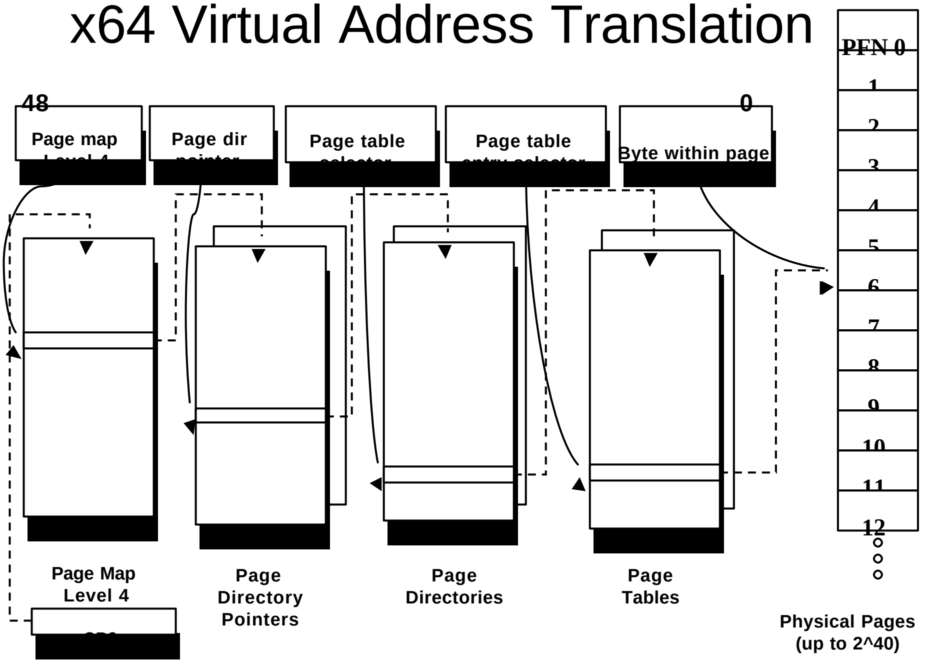
Windows Virtual Memory Use Performance Counters

Performance Counter	System Variable	Description
Memory: Committed Bytes	MmTotalCommitedPages	Amount of committed private address space that has a backing store
Memory: Commit Limit	MmTotalCommit-Limit	Amount of memory (in bytes) that can be committed without increasing size of paging file
Memory: %Commited Bytes in Use	MmTotalCommittedPages / MmTotalCommitLimit	Ratio of committed bytes to commit limit

x86 Virtual Address Translation

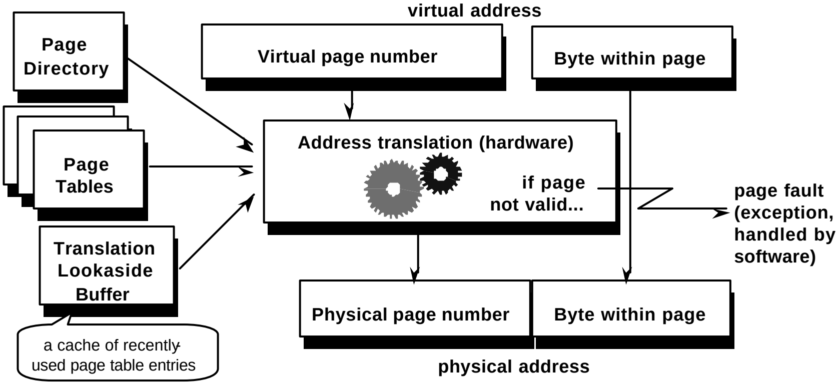


x64 Virtual Address Translation



Virtual Address Translation

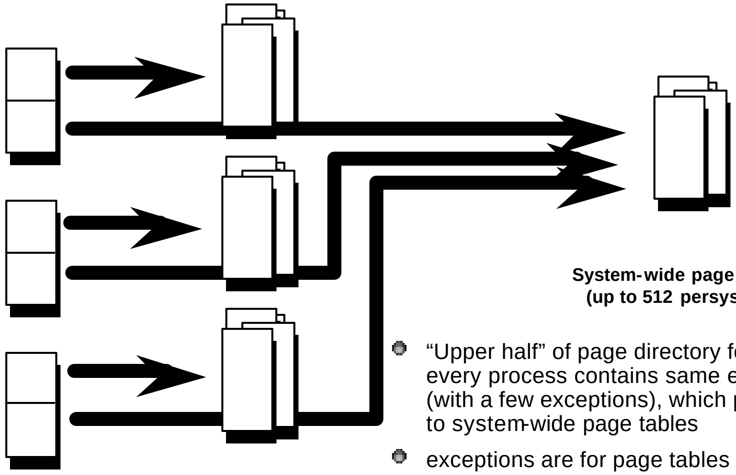
- The hardware converts each valid virtual address to a physical address



Itanium Address Translation

- 3 level page table (vs 2 on x86)
  - 43 bit virtual addressing
  - 44 bit physical addressing
- Two TLBs
  - Instruction TLB – translates instruction addresses
  - Data TLB – translates data addresses
- Each have OS-managed *translation registers* and hardware managed *translation cache*
  - OS can insert TLB entries
    - OS decides which slots when inserting into translation registers
    - Hardware decides when inserting into translation cache
  - Itanium: 96 instruction translation cache entries; 128 data translation cache entries

Mapping Process vs. System-Space Addresses



System-wide page tables (up to 512 persystem)

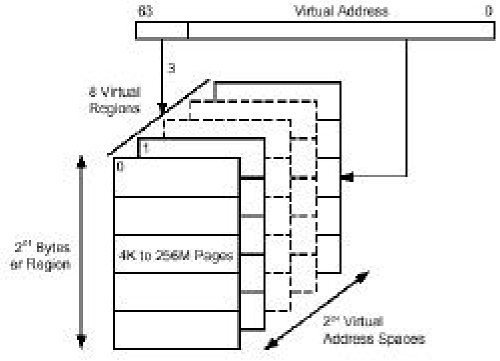
- “Upper half” of page directory for every process contains same entries (with a few exceptions), which point to system-wide page tables
- exceptions are for page tables that map the process page tables (not shown)

Page Directories (one per process)

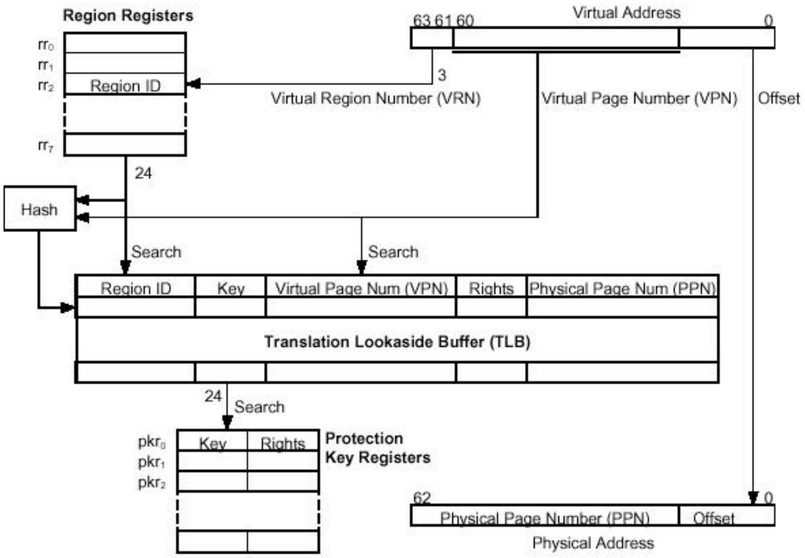
Sets of per-process page tables (up to 512 per process)

Itanium Address Translation

- Address space divided into 8 regions



Itanium Address Translation



Page Directory and Page Table Entries

1 virtual address of PD Entry or PT Entry

2 contents of PDE or PTE

3 interpreted contents

4 Page Frame Number (== physical page number) of Page Table

5 Page Frame Number (== physical page number) for valid page

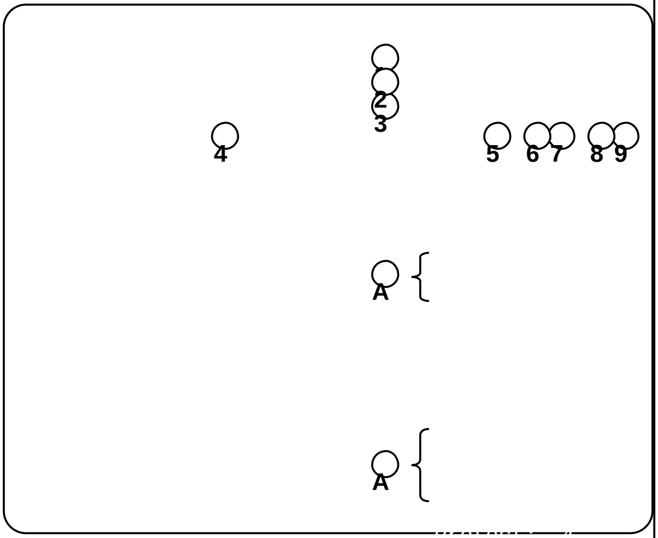
6 D = Dirty (modified since made valid)

7 A = Accessed (recently)

8 KW = Kernel mode writable

9 V = Valid bit

A Where pager can find contents of an invalid page



Screen snapshot from:
Kernel debugger !pte command on randomly-selected virtual addresses

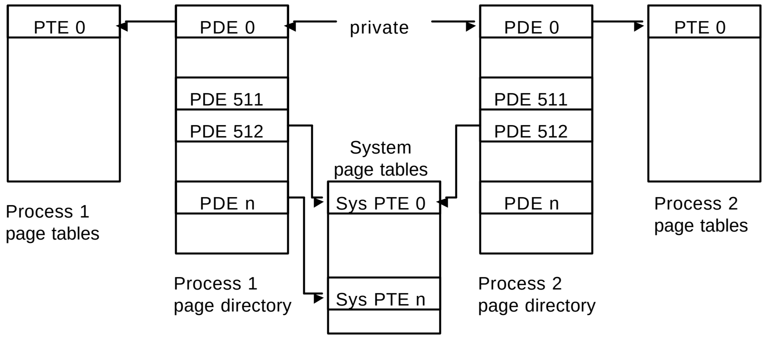
Translating a virtual address:

1. Memory management HW locates page directory for current process (cr3 register on Intel, PDR on Alpha)
2. Page directory index directs to requested page table
3. Page table index directs to requested virtual page
4. If page is valid, PTE contains physical page number (PFN – page frame number) of the virtual page
    - Memory manager fault handler locates invalid pages and tries to make them valid
    - Access violation/bug check if page cannot be brought in (prot. fault)
5. When PTE points to valid page, byte index is used to locate address of desired data

Page directories & Page tables

- Each process has a single page directory (phys. addr. in KPROCESS block, at 0xC0300000, in cr3 (x86))
    - cr3 is re-loaded on inter-process context switches
    - Page directory is composed of page directory entries (PDEs) which describe state/location of page tables for this process
        - Page tables are created on demand
    - x86: 1024 page tables describe 4GB
- Each process has a private set of page tables
- System has one set of page tables
    - System PTEs are a finite resource: computed at boot time
    - HKLM\System...\Control\SessionManager\SystemPages

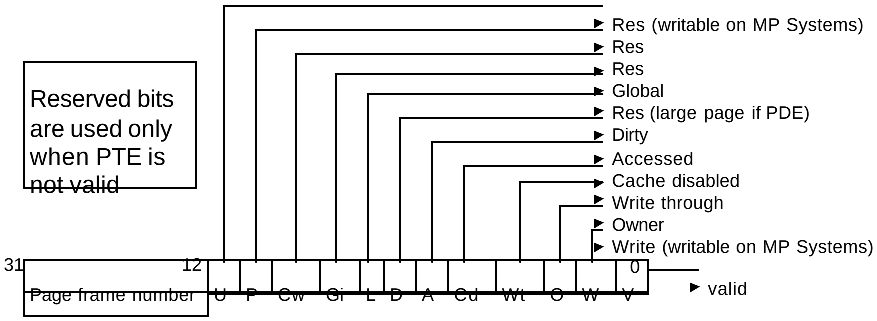
System and process-private page tables



- On process creation, system space page directory entries point to existing system page tables
- Not all processes have same view of system space (after allocation of new page tables)

Page Table Entries

- Page tables are array of Page Table Entries (PTEs)
- Valid PTEs have two fields:
  - Page Frame Number (PFN)
  - Flags describing state and protection of the page

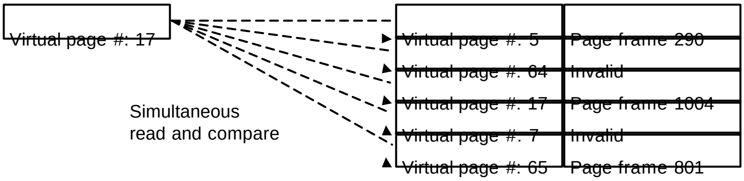


PTE Status and Protection Bits (Intel x86 only)

Name of Bit	Meaning on x86
Accessed	Page has been read
Cache disabled	Disables caching for that page
Dirty	Page has been written to
Global	Translation applies to all processes (a translation buffer flush won‘t affect this PTE)
Large page	Indicates that PDE maps a 4MB page (used to map kernel)
Owner	Indicates whether user-mode code can access the page of whether the page is limited to kernel mode access
Valid	Indicates whether translation maps to page in phys. Mem.
Write through	Disables caching of writes; immediate flush to disk
Write	Uniproc: Indicates whether page is read/write or read-only; Multiproc: ind. whether page is writeable/write bit in res. bit

Translation Look-Aside Buffer (TLB)

- Address translation requires two lookups:
  - Find right table in page directory
  - Find right entry in page table
- Most CPU cache address translations
  - Array of associative memory: translation look-aside buffer (TLB)
  - TLB: virtual-to-physical page mappings of most recently used pages



Virtual page #: 5	Page frame 290
Virtual page #: 64	Invalid
Virtual page #: 17	Page frame 1004
Virtual page #: 7	Invalid
Virtual page #: 65	Page frame 801

Page Fault Handling

- Reference to invalid page is called a page fault
- Kernel trap handler dispatches:
  - Memory manager fault handler (MmAccessFault) called
  - Runs in context of thread that incurred the fault
  - Attempts to resolve the fault or raises exception
- Page faults can be caused by variety of conditions
- Four basic kinds of invalid Page Table Entries (PTEs)

Reasons for access faults

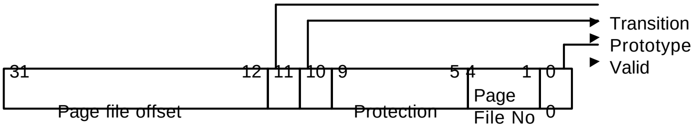
- Accessing a page that is not resident in memory but on disk in page file/mapped file
  - Allocate memory and read page from disk into working set
- Accessing page that is on standby or modified list
  - Transition the page to process or system working set
- Accessing page that has no committed storage
  - Access violation
- Accessing kernel page from user-mode
  - Access violation
- Writing to a read-only page
  - Access violation

Reasons for access faults (contd.)

- Writing to a guard page
  - Guard page violation (if a reference to a user-mode stack, perform automatic stack expansion)
- Writing to a copy-on-write page
  - Make process-private copy of page and replace original in process or system working set
- Referencing a page in system space that is valid but not in the process page directory
  (if paged pool expanded after process directory was created)
  - Copy page directory entry from master system page directory structure and dismiss exception
- On a multiprocessor system: writing to valid page that has not yet been written to
  - Set dirty bit in PTE

Invalid PTEs and their structure

- **Page file**: desired page resides in paging file
  in-page operation is initiated

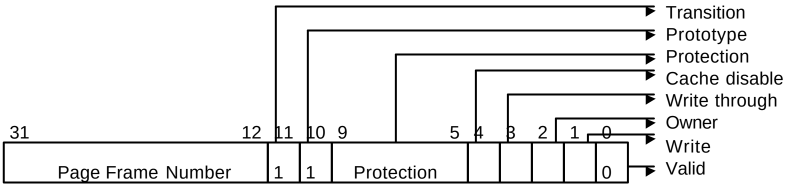


- **Demand Zero**: pager looks at zero page list;
  if list is empty, pager takes list from standby list and zeros it;

  PTE format as shown above, but page file number and offset are zeros

Invalid PTEs and their structure (contd.)

- **Transition**: the desired page is in memory on either the standby, modified, or modified-no-write list
  - Page is removed from the list and added to working set



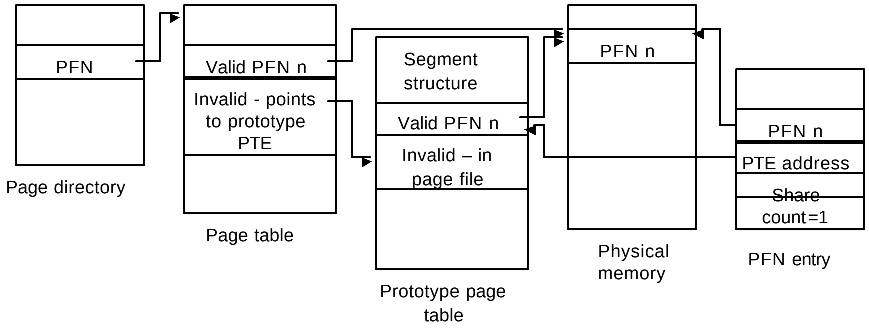
- **Unknown**: the PTE is zero, or the page table does not yet exist
  - examine virtual address space descriptors (VADs) to see whether this virtual address has been reserved
  - Build page tables to represent newly committed space

Prototype PTEs

- Software structure to manage potentially shared pages
  - Array of prototype PTEs is created as part of section object (part of segment structure)
  - First access of a page mapped to a view of a section object:
    memory manager uses prototype PTE to fill in real PTE used for address translation;
  - Reference count for shared pages in PFN database
- Shared page valid:
  - process & prototype PTE point to physical page
- Page invalidated:
  - process PTE points to prototype PTE
- Prototype PTE describes 5 states for shared page:
  - Active/valid, Transition, Demand zero, Page file, Mapped file
- Layer between page table and page frame database

Prototype PTEs for shared pages – the bigger picture

- Two virtual pages in a mapped view
- First page is valid; 2nd page is invalid and in page file
  - Prototype PTE contains exact location
  - Process PTE points to prototype PTE



PFN

Page directory

Valid PFN n

Invalid - points to prototype PTE

Page table

Segment structure

Valid PFN n

Invalid – in page file

Prototype page table

PFN n

Physical memory

PFN n

PTE address

Share count=1

PFN entry

In-Paging I/O

- Occurs when read operation must be issued to a file to satisfy page fault
  - Page tables are pageable -> additional page faults possible
- In-page I/O is synchronous
  - Thread waits until I/O completes
  - Not interruptible by asynchronous procedure calls
- During in-page I/O: faulting thread does not own critical memory management synchronization objects

  Other threads in process may issue VM functions, but:
  - Another thread could have faulted same page: collided page fault
  - Page could have been deleted (remapped) from virtual address space
  - Protection on page may have changed
  - Fault could have been for prototype PTE and page that maps prototype PTE could have been out of working set

Page files

- Windows supports up to 16 paging files
- Once open, page file can‘t be deleted
  while system is running
  - System process maintains open handle to each page file
- NtCreatePageFile system service in NTDLL.DLL (internal only)
- Page files are always created as uncompressed files
- Memory management tracks page file usage:
  - Global: commitment
  - On a per-process basis: Page file quota
  - VM allocation will fail when commit limit has reached

Virtual address descriptors (VADs)

- Memory manager uses demand paging algorithm
- Lazy evaluation is also used to construct page tables
  - Reserved vs. commited memory
  - Even for commited memory, page table are constructed on demand
- Memory manager maintains VAD structures to keep track of reserved virtual addresses
  - Self-balancing binary tree
- VAD store:
  - range of addresses being reserved;
  - whether range will be shared or private;
  - Whether child process can inherit contents of the range
  - Page protection applied to pages within the address range

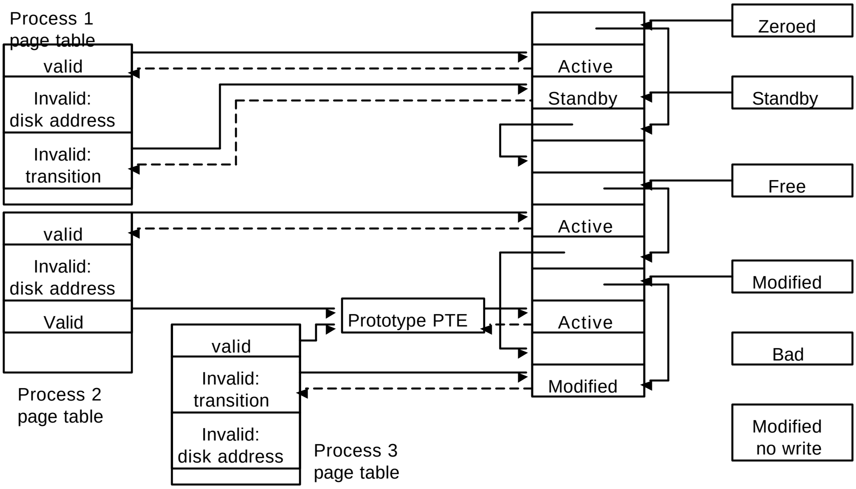
Page Frame Number-Database

- One entry (24 bytes) for each physical page
  - Describes state of each page in physical memory
- Entries for active/valid and transition pages contain:
  - Original PTE value (to restore when paged out)
  - Original PTE virtual address and container PFN
  - Working set index hint (for the first process...)
- Entries for other pages are linked in:
  - Free, standby, modified, zeroed, bad lists (parity error will kill kernel)
- Share count (active/valid pages):
  - Number of PTEs which refer to that page; 1->0: candidate for free list
- Reference count:
  - Locking for I/O: INC when share count 1->0; DEC when unlocked
  - Share count = 0 & reference count = 1 is possible
  - Reference count 1->0: page is inserted in free, standby or modified lists

Page Frame Database – states of pages in physical memory

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 tables and page frame database



MM: Process Support

- MmCreateProcessAddressSpace – 3 pages
  - The page directory
    - Points to itself
    - Map the page table of the hyperspace
    - Map system paged and nonpaged areas
    - Map system cache page table pages
  - The page table page for working set
  - The page for the working set list
- MmInitializeProcessAddressSpace
  - Initialize PFN for PD and hyperspace PDEs
  - MiInitializeWorkingSetList
  - Optional: MmMapViewOfSection for image file
- MmCleanProcessAddressSpace,
- MmDeleteProcess AddressSpace

MM: Process Swap Support

- MmOutSwapProcess / MmInSwapProcess
- MmCreateKernelStack
  - MiReserveSystemPtes for stack and no-access page
- MmDeleteKernelStack
  - MiReleaseSystemPtes
- MmGrowKernelStack
- MmOutPageKernelStack
  - Signature (thread_id) written on top of stack before write
  - The page goes to transition list
- MmInPageKernelStack
  - Check signature after stack page is read / bugcheck

MM: 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 WS limit is reached
  - NT 4.0: page replacement based on modified FIFO
  - Windows 2000: Least Recently Used algorithm (uniproc.)

MM: Working Set Management

- Modified Page Writer thread
  - Created at system initialization
  - Writing modified pages to backing file
  - Optimization: min. I/Os, contigous pages on disk
  - Generally MPW is invoked before trimming
- 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

MM: 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

MM: 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

MM: POSIX fork() support

- MiCloneProcessAddressSpace
  - Copy parent‘s address space to the child address space
  - Examines each VAD‘s inheritance attribute
  - If needed, copies each PTE into the new address space
  - For private pages: use prototype PTEs, copy-on-write between the two processes

Further Reading

- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals, 4th Edition, Microsoft Press, 2004.
- Chapter 7 - Memory Management
  - Address Translation (pp. 425 ff.)
  - Shared Memory and Mapped Files (pp. 386 ff.)
  - Page Frame Number Database (pp. 469 ff.)