Precept 2: PMM / VMM

COS 318: Fall 2017

Project 2 Schedule

- Precept: Monday 10/02, 7:30pm
  - (You are here)
- Design Review: Monday 10/09
- Due: Sunday, 10/15, 11:55pm

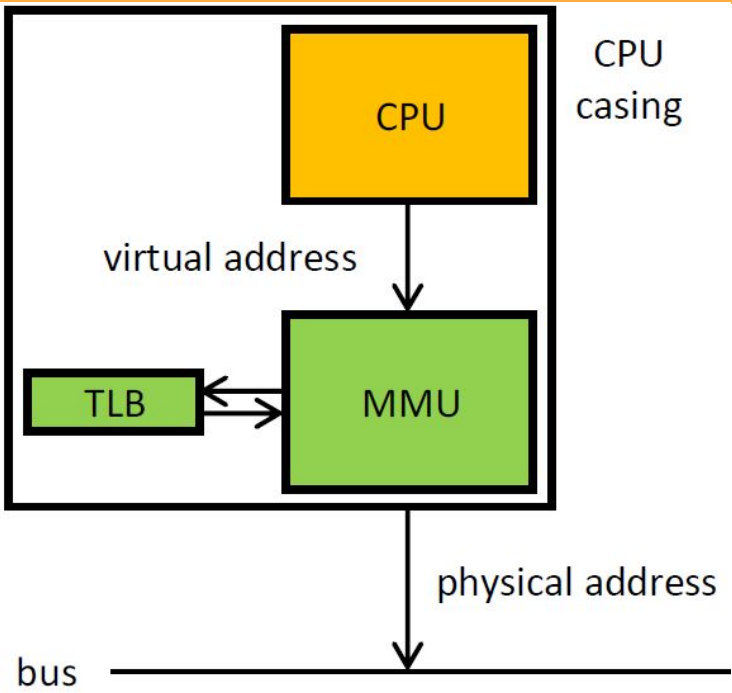
Precept Overview

- Virtual Memory Management
- Physical Memory Management
- Project Specific Topics

Virtual Memory Management

VA to PA Translation: Overview

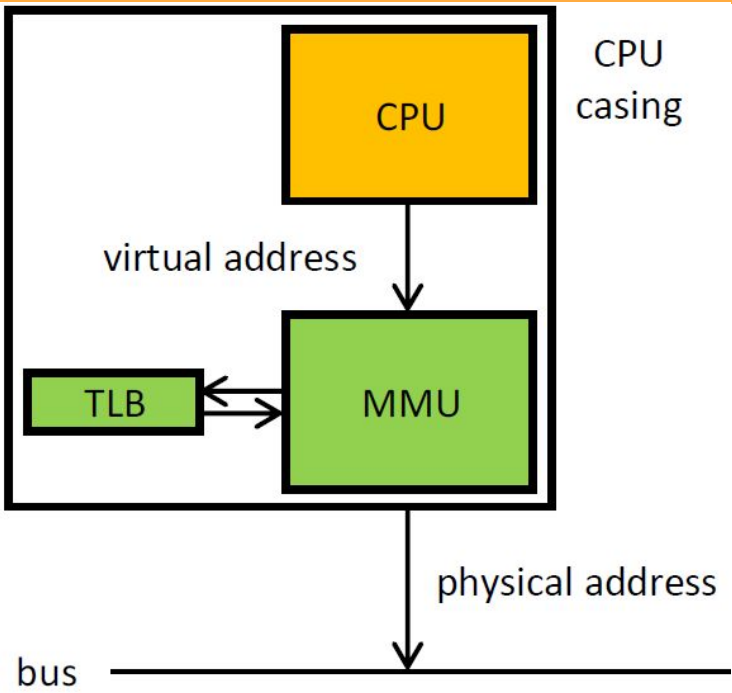
- All addresses are virtual
  => must go through MMU
- MMU checks TLB first
- On miss: performs translation using page tables
- Image Source



CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

VA to PA Translation: MMU

- Page tables defined in software
- Use CR3 register to find root page table in RAM
- Checks page permissions - faults if invalid
- Image Source



CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

Segmentation: Logical to Linear

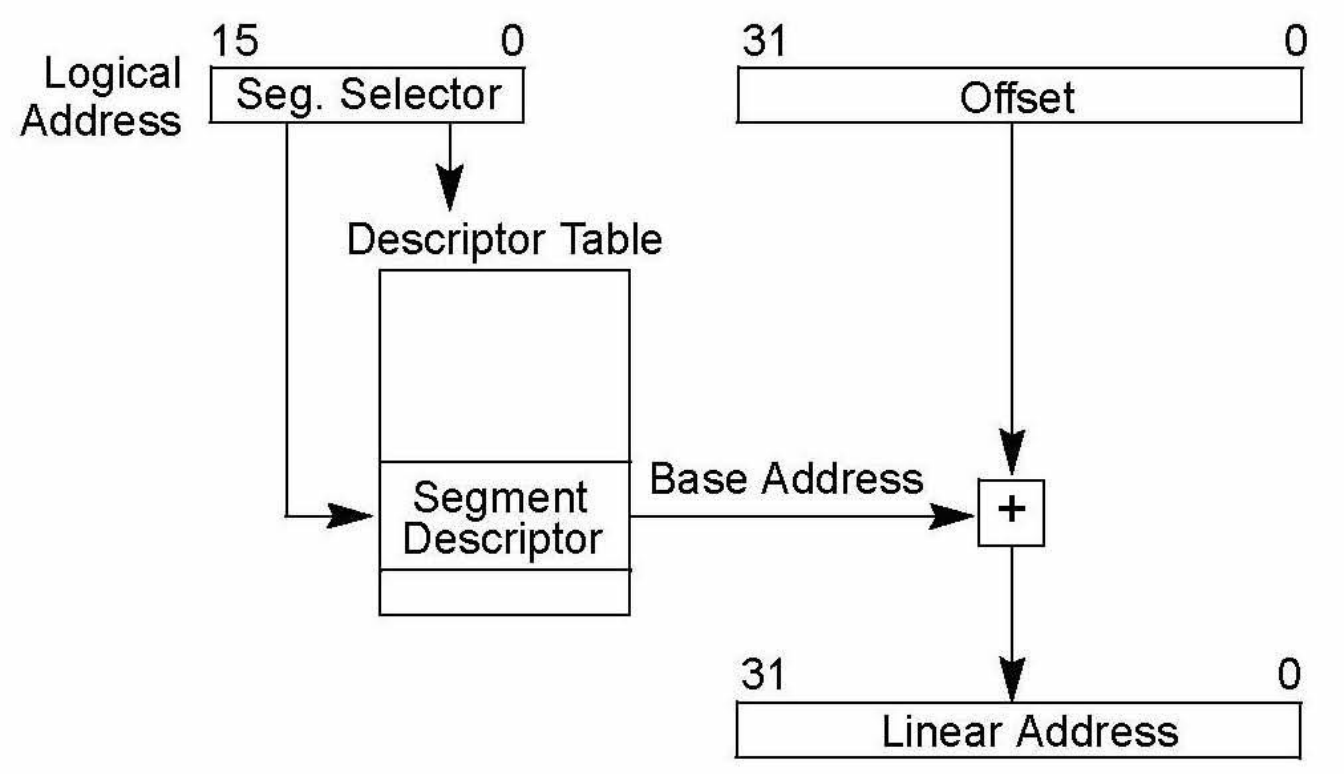


Image Source: http://iakovlev.org/images/intel/35.jpg

Segmentation: Logical to Linear

NO Segmentation on this assignment!

Paging System: Linear to Physical

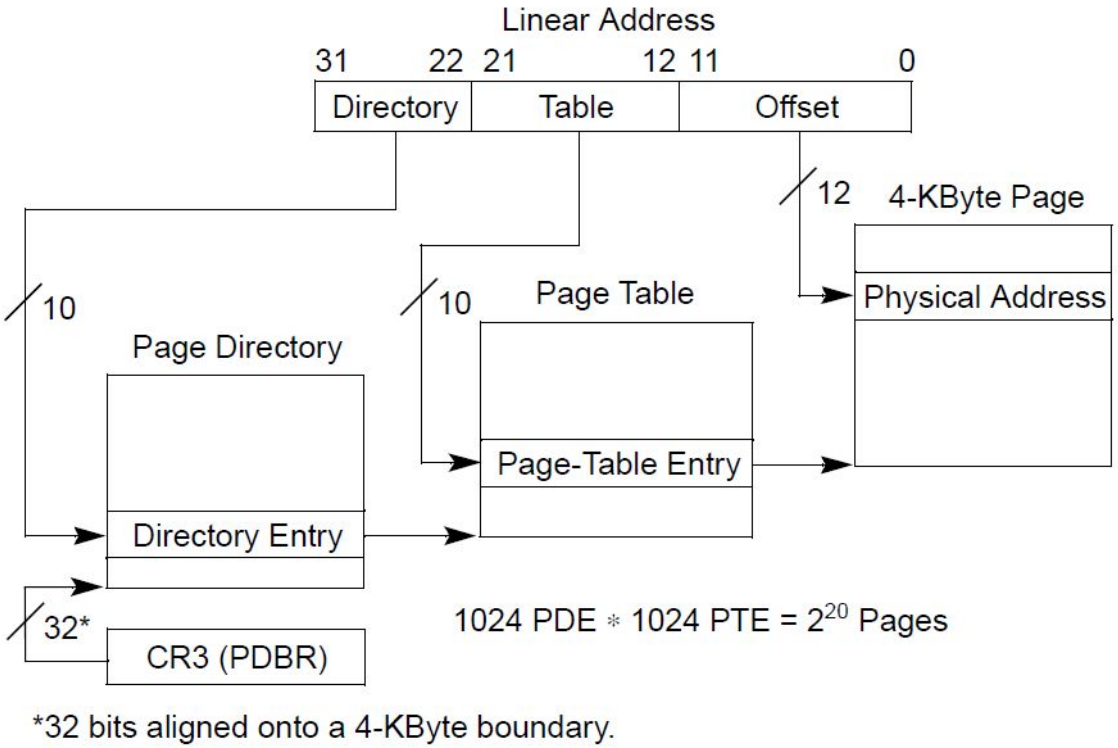


Image Source: https://www.coresecurity.com/sites/default/files/wp-content/uploads/2016/05/32bit-tables.png

Paging System: Dir. / Table Entries

- Hierarchical System:
  - Directory Entries hold page table start address
  - Table Entries hold page start address
    - (If page is not in memory, swap it in)
  - Page start address + offset = Physical address

Paging System: Dir. / Table Entries

- Dirs and Tables must fit onto a 4KB page!
  - Therefore, the lower 12 bits of the start address are always 0
- Higher 20 bits hold start address, lower 12 bits store permissions / status

Paging System: Directory Entries

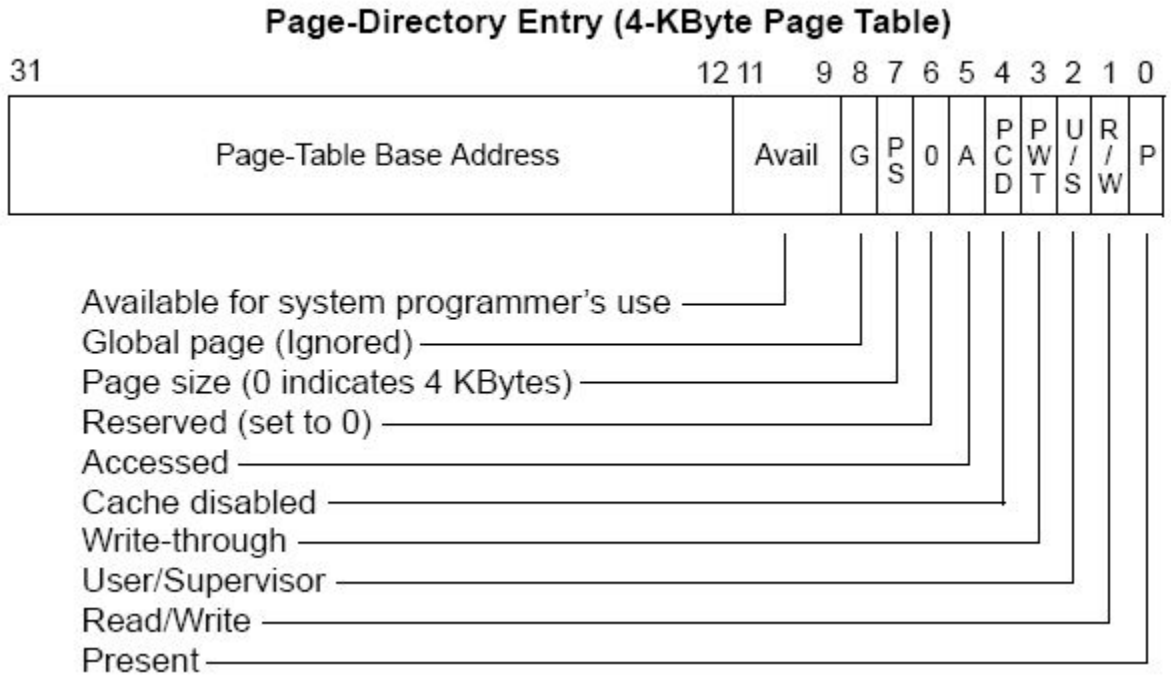


Image Source: http://valhalla.bofh.pl/~l4mer/WDM/secureread/PdePte.png

Paging System: Table Entries

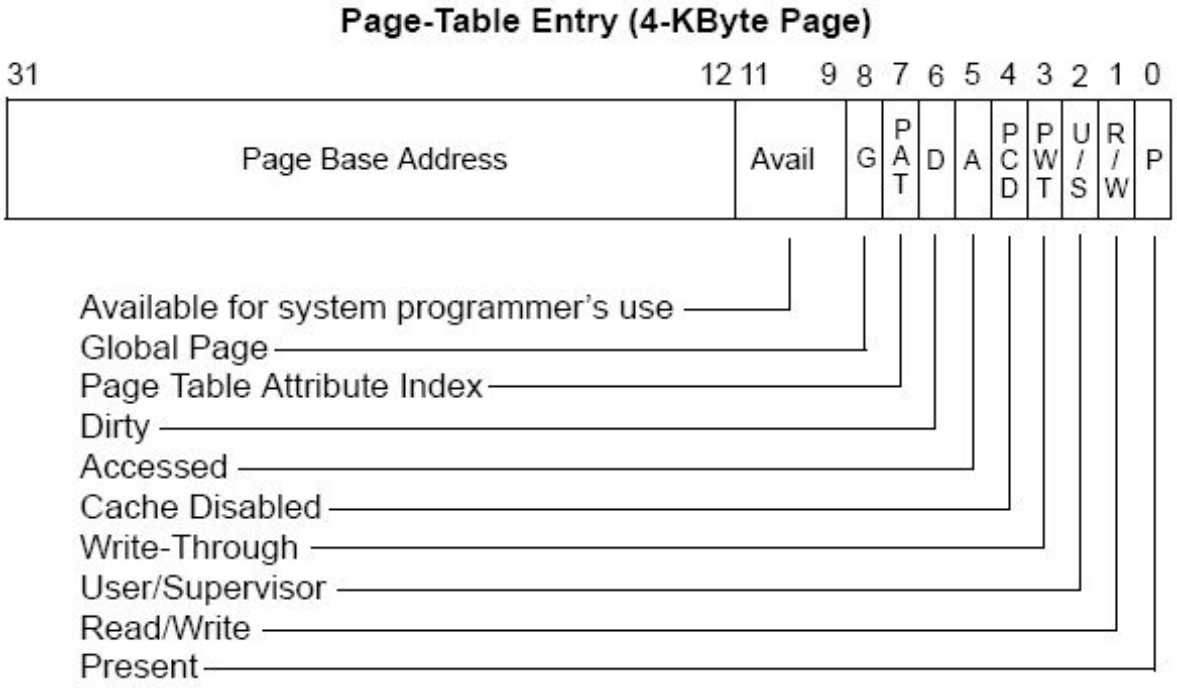


Image Source: http://valhalla.bofh.pl/~l4mer/WDM/secureread/PdePte.png

Paging System: VA Structure

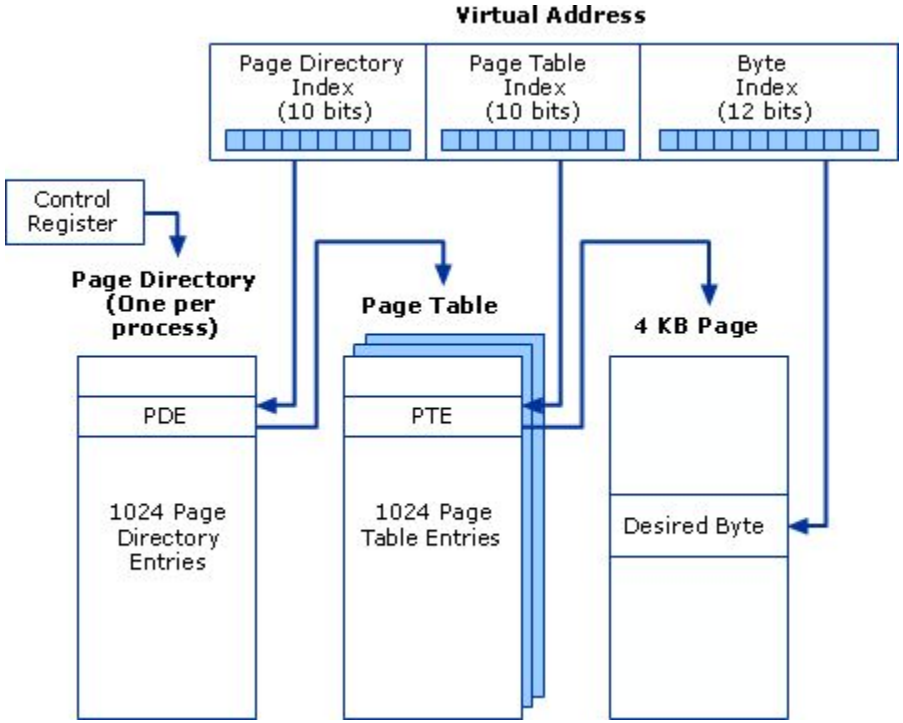


Image Source: https://i.stack.imgur.com/x10Lv.gif

Check: VA Space = Paging Space

- We use 32-bit (4-byte) VAs, 4KB pages, and a two level page table system
  - 4KB per page / 4 bytes per entry = 1K entries
- 2^10 (p.d.e) * 2^10 (p.t.e) * 2^12 (bytes per page) = 2^32 addressable bytes
- 32 bits can address 2^32 locations

Paging System: Process Allocation

- Each process gets its own page directory
- Kernel sometimes needs to access raw physical memory addresses
    - Solution: Reserve an identity page directory for the kernel

Physical Memory Management

Page Alloc / Free (Last Assignment)

- palloc function in MATOP layer;
  - Check your AT array to see whether the permission state and allocation state;
  - If both permission is normal and allocation is unallocated, then the page is free. (For the user space)
- pfree function to free the page.

Page Alloc / Free (This Assignment)

- Update the Container
  - Container is an array;
  - Each entry keeps track of metadata of one process.
    - eg. Quota -- total pages one process can used;
    - Used -- used pages of one process.
- Update the page table;
  - Might need to allocate a page for a new page table;
  - Update the page table entry;

MMU gets a physical address

- What happens when its already in memory
  - return the page

- When it misses -
  - page fault

Page Fault

1. A page fault happens because the virtual page is not resident on a physical page frame
    a. Allocate a new page
    b. The page was swapped to disk
2. How does the hardware know that a page fault happened?
    a. Check the page table entry

General Page Fault Handle Procedure

1. Allocate a page frame of physical memory
2. (If necessary) Load the contents of the page from the appropriate swap location on disk
3. Update the page table of the process

Swapping pages to / from disk

- Check present bit in PTE (page table entry);
  - 1 -> present; 0 -> not;
- Disk swap location is in PTE as well;
- However, No page swapping to/from disk in this assignment.

Project Specific Topics

Getting Started + Usable Code

- Please use the provided solutions
  - `cp -r /u/318/code/samples/* <lab2_path>`
- C std. libraries aren't usable
  - They depend on the kernel… which we're writing
- Provided materials / code you wrote from other courses are also acceptable

Potential Pain Points

- “Page Structure” is essentially “Page Directory”
- Need to cast between `char *` and `unsigned int`
- `PDirPool` = statically allocated pool of page dirs.
  - Page tables are allocated dynamically
- `IDPtbl` = identity page directory

Other Advice

- A lot of functions, not a lot of code
  - Understand what is going on BEFORE you start coding
- `MPTIntro` and `MPTOp` are the core of the project
  - Suggestion: Have a good handle on these before Week 2

Design Review

- Describe the process of converting a linear address to a physical address.
- What physical address does the virtual address `0x12345678` map to while executing kernel code? Why is this so?
- What page directory entry index, page table entry index, and page offset does the virtual address 0xBADDCAFE correspond to?
- If we used 64-bit addresses instead of 32-bit addresses, how many bytes of physical memory would a page directory be able to access? How many page table levels would we need to exceed 2^64 bytes of pageable memory?

Questions?