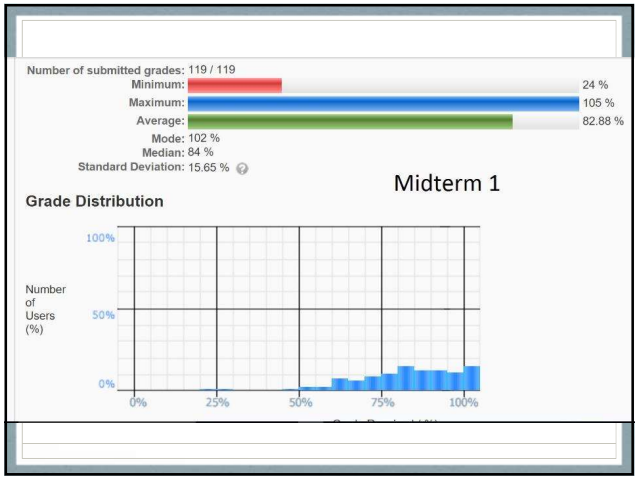
CS354: Machine Organization and Programming

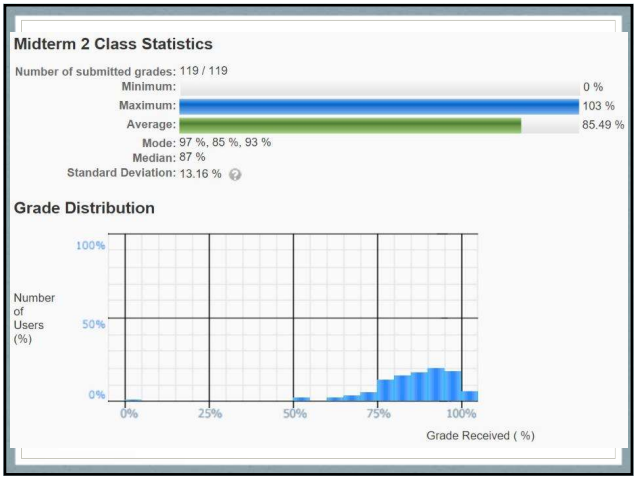
Lecture 32
Monday the November 16th 2015

Section 2
Instructor: Leo Arulraj

Class Announcements

1. Midterm 2 grades have been posted in learn@uw. Collect your graded exams from me this week during class or during office hours after that.
2. Please come and see me during office hours for any questions regarding grading or totaling errors for Midterm 2.





Lecture Overview

1. Multi level page tables
2. Example

Idea: Software Managed TLB

H/W has to know so much about the Page table structure. (Software managed TLB)

Upon a TLB miss,

- H/W raises the TLB miss exception
- Run TLB miss exception handler that updates the TLB using a special instruction
- Return from the exception to retry the instruction

Why bother?: It is advantageous to keep the H/W Simple and let the S/W have more flexibility.

Virtual Memory: Paging Problem #2's Solution

Page tables are too big in size.

Solutions:

1) Multi-level page tables (our focus)
2) Segmented Page tables (base+bounds earlier)
3) Inverted page tables
4) Swap page tables to disk (+break recursion)

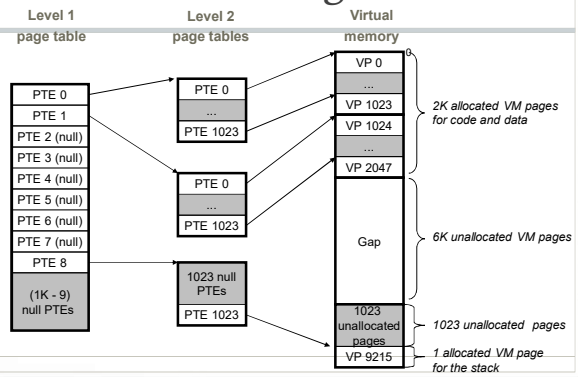
Two-level page table: Motivation

1. Consider 32bit virtual address and 4KB pages.
2. Needs 4MB for a flat page table per process.

3. Assume a process with memory layout as:
   a. First 2K pages : code and data
   b. Next 6K+1023 pages: unallocated
   c. Next page: stack

Then the two level page table for this process will look like as shown in next slide.

Two level Page table



Two level Page table

Reduces memory requirements in two ways:

1. If a PTE in level 1 is null, then corresponding level 2 page table does not even have to exist. Most programs have lots of unallocated virtual address space regions.
2. Only the level 1 page tables needs to be in memory at all times. Level 2 page tables can be paged in and out by the Virtual Memory system.

CS354: Machine Organization and Programming

Lecture 33
Wednesday the November 18th 2015

Section 2
Instructor: Leo Arulraj

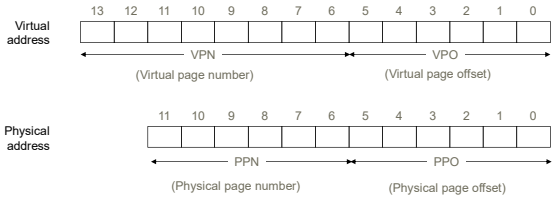
Lecture Overview

1. Continue with another example of end to end address translation
2. Intel core i7 case study
3. Linux specific virtual memory related details
   (Not important from Final exam perspective)

End-to-end Address Translation Example for CSAPP Textbook

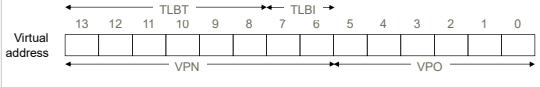
1. Memory is byte-addressable
2. Memory accesses are to 1-byte words (not 4-byte words)
3. Virtual Addresses are 14 bits wide (n=14)
4. Physical Addresses are 12 bits wide (m=12)
5. Page size is 64 bytes (P=64)
6. TLB is 4-way set associative with 16 total entries
7. L1 d-cache is physically addressed and direct mapped with a 4-byte line size and 16 total sets.

End-to-end Address Translation Format of virtual & physical addresses



TLB: Four sets, 16 entries, 4 way set associative

1. 2 low order bits of VPN used as set index.
2. 6 high order bits serve as the tag.



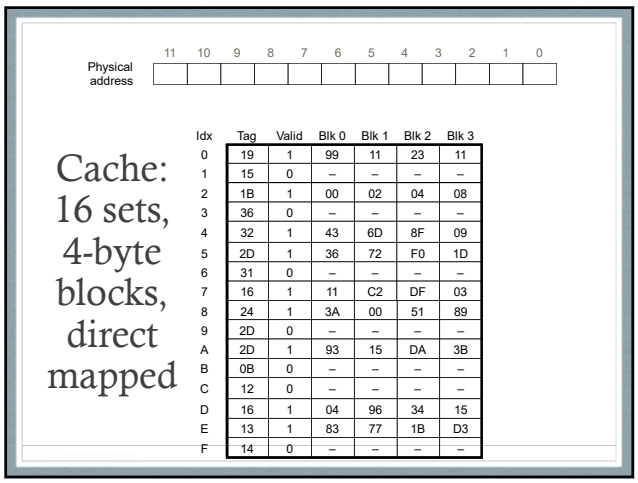
Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Page table

Only the first 16 PTEs are shown

VPN	PPN	Valid
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	1A	0
0D	2D	1
0E	11	1
0F	0D	1



Problems: Analyzing memory references

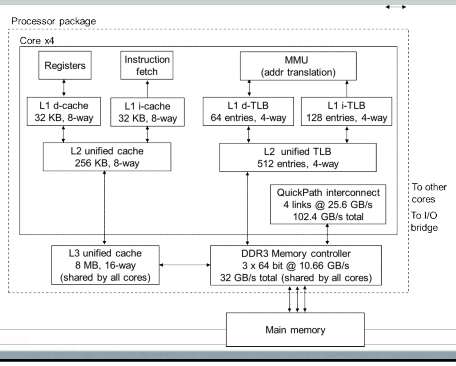
In Class:

0x0354
0x0314

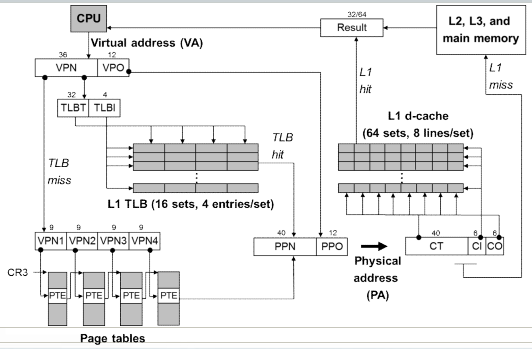
Try yourself (solved in text book):

1. 0x03d4?
2. 0x03d7?

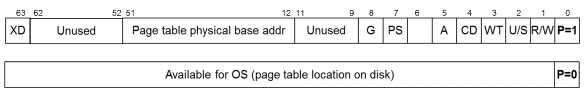
Case study: Intel core i7



Intel core i7: Address Translation



Intel core i7: Level 1,2,3 PTE Format



Some bits are: (more in Textbook)

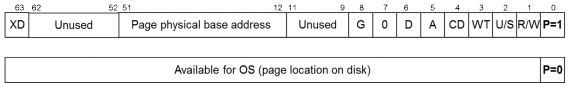
U/S – user or supervisor mode access

R/W – read only or read write access

XD – Disable or enable execute bit

CD – cache disabled or enabled

Intel core i7: Level 4 PTE Format



Some bits are: (more in Textbook)

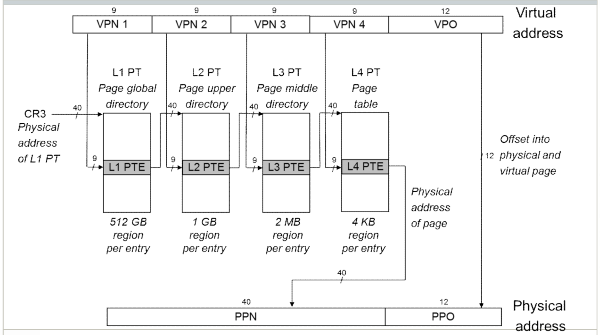
A – reference bit (set by MMU)

D – Dirty bit

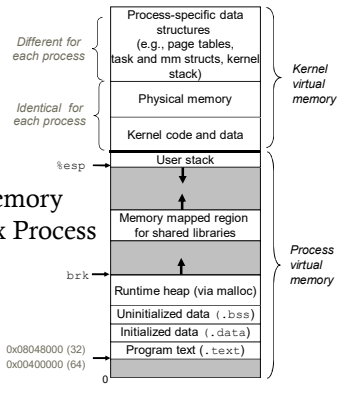
WT – Write through or write back cache policy

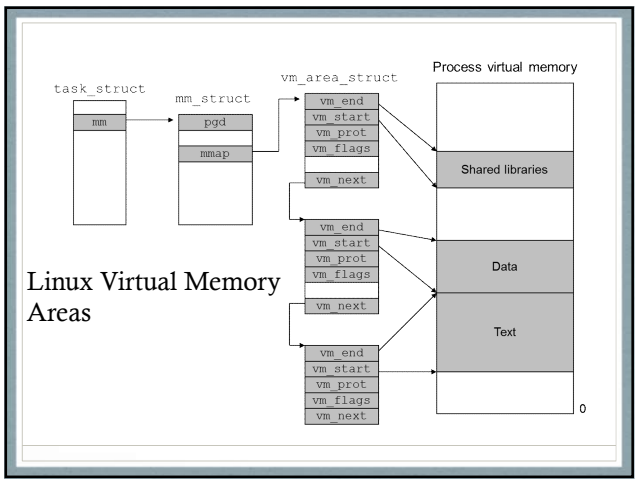
G – Global page (don't evict on task switch)

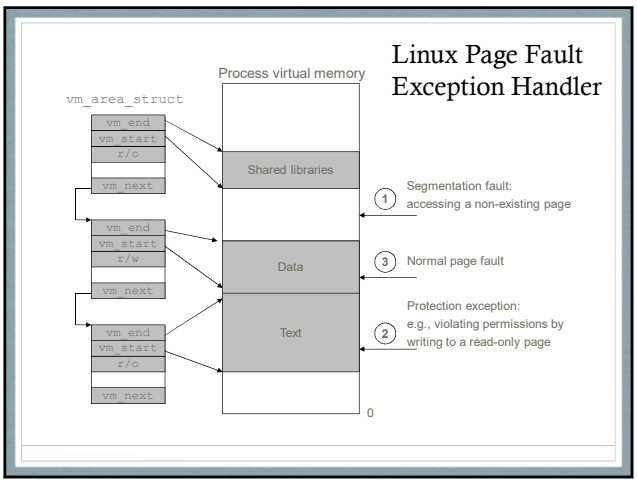
Intel core i7: Page Table Translation



Virtual Memory Of a Linux Process







Memory Mapping

Contents of Virtual Memory initialized by memory mapping in Linux:

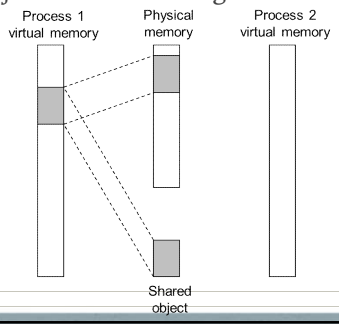
1. Regular file in the unix system
2. Anonymous file : demand zero pages

In both cases, initialized pages can be swapped in and out to on disk location called "swap space".

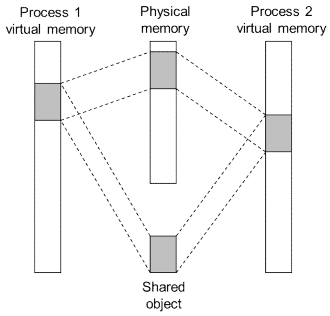
Total virtual memory that can be allocated by the currently running process is bound by the amount of swap space.

Shared Objects

Shared objects: Before sharing

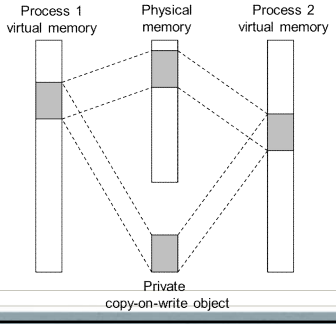


Shared Objects – After Sharing

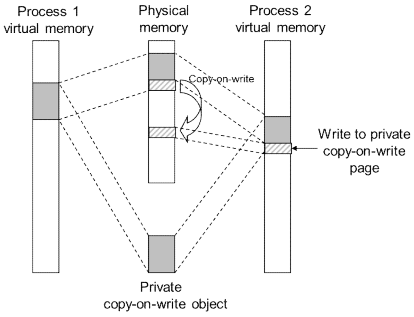


Private Copy on Write Objects

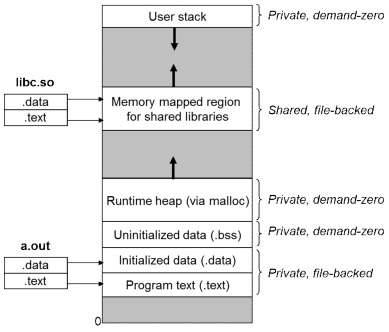
Before Writing to Copy on write object



Private Copy on Write Objects (After writing)



Memory mapping by loader for the user address space



mmap arguments interpretation

void *mmap(void* start, size_t length, int prot, int flags, int fd, off_t offset);
Returns: pointer to mapped area if OK