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.
Number of submitted grades: 119 / 119
Minimum: 24 %
Maximum: 105 %
Average: 82.88 %
Mode: 102 %
Median: 84 %
Standard Deviation: 15.65 %

Grade Distribution

Midterm 1
Midterm 2 Class Statistics

Number of submitted grades: 119 / 119

Minimum: 0 %
Maximum: 103 %
Average: 85.49 %
Mode: 97 %, 85 %, 93 %
Median: 87 %
Standard Deviation: 13.16 %

Grade Distribution
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

Level 1
Page table
- PTE 0
- PTE 1
- PTE 2 (null)
- PTE 3 (null)
- PTE 4 (null)
- PTE 5 (null)
- PTE 6 (null)
- PTE 7 (null)
- PTE 8
  - (1K - 9) null PTEs

Level 2
Page tables
- PTE 0
  - ...
  - PTE 1023
- PTE 0
  - ...
  - PTE 1023
- 1023 null PTEs
  - PTE 1023

Virtual memory
- VP 0
  - ...
  - VP 1023
  - VP 1024
  - VP 2047
  - Gap
  - 1023 unallocated pages
  - VP 9215

Allocations:
- 2K allocated VM pages for code and data
- 6K unallocated VM pages
- 1023 unallocated pages
- 1 allocated VM page for the stack
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 18\textsuperscript{th} 2015

Section 2
Instructor: Leo Arulraj

© 2015 Karen Smoler Miller
© Some examples, diagrams from the CSAPP text by Bryant and O’Hallaron
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

Virtual address

13 12 11 10 9 8 7 6 5 4 3 2 1 0

VPN
(Virtual page number)

VPO
(Virtual page offset)

Physical address

11 10 9 8 7 6 5 4 3 2 1 0

PPN
(Physical page number)

PPO
(Physical page offset)
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.

![Diagram of TLB with virtual address, tag, PPN, and valid bits]

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>–</td>
<td>0</td>
<td>09</td>
<td>0D</td>
<td>1</td>
<td>00</td>
<td>–</td>
<td>0</td>
<td>07</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>–</td>
<td>0</td>
<td>04</td>
<td>–</td>
<td>0</td>
<td>0A</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>–</td>
<td>0</td>
<td>08</td>
<td>–</td>
<td>0</td>
<td>06</td>
<td>–</td>
<td>0</td>
<td>03</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>–</td>
<td>0</td>
<td>03</td>
<td>0D</td>
<td>1</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
<td>–</td>
<td>0</td>
</tr>
</tbody>
</table>
Page table

Only the first 16 PTEs are shown

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>0B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0C</td>
<td>1A</td>
<td>0</td>
</tr>
<tr>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
### Cache: 16 sets, 4-byte blocks, direct mapped

<table>
<thead>
<tr>
<th>Blk 3</th>
<th>Blk 2</th>
<th>Blk 1</th>
<th>Blk 0</th>
<th>Valid</th>
<th>Tag</th>
<th>Idx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1B</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2D</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2D</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2D</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td>F</td>
</tr>
</tbody>
</table>

- **Physical address:**
  - Blk 0: 99, 11, 23, 11
  - Blk 1: 00, 02, 04, 08
  - Blk 2: 43, 6D, 8F, 09
  - Blk 3: 36, 72, F0, 1D

- **Cache:**
  - 16 sets, 4-byte blocks, direct mapped

---

**Notes:**
- **Valid:** Indicates whether the block is in the cache (1) or not (0).
- **Tag:** Used for identifying the block in the cache.
- **Idx:** Index of the block in the cache set.
Problems: Analyzing memory references

In Class:

0x0354  
0x0314  

Try yourself (solved in text book):
1. 0x03d4?  
2. 0x03d7?
Case study: Intel core i7

Processor package

Core x4

- Registers
- Instruction fetch
- L1 d-cache 32 KB, 8-way
- L1 i-cache 32 KB, 8-way
- L2 unified cache 256 KB, 8-way

MMU (addr translation)

- L1 d-TLB 64 entries, 4-way
- L1 i-TLB 128 entries, 4-way
- L2 unified TLB 512 entries, 4-way

QuickPath interconnect
4 links @ 25.6 GB/s
102.4 GB/s total

L3 unified cache 8 MB, 16-way (shared by all cores)

DDR3 Memory controller
3 x 64 bit @ 10.66 GB/s
32 GB/s total (shared by all cores)

Main memory

To other cores
To I/O bridge
Intel core i7: Address Translation

Virtual address (VA)

CPU

VPN VPO

36 12

TLBT TLBI

32 4

TLB
miss

L1 TLB (16 sets, 4 entries/set)

VPN1 VPN2 VPN3 VPN4

9 9 9 9

CR3

PTE PTE PTE PTE

Page tables

32/64 Result

L2, L3, and main memory

L1 hit

L1 miss

L1 d-cache
(64 sets, 8 lines/set)

PPN PPO

40 12

Physical address (PA)

CT CI CO

40 6 6
Intel core i7: Level 1,2,3 PTE Format

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>52</th>
<th>51</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD</td>
<td>Unused</td>
<td>Page table physical base addr</td>
<td>Unused</td>
<td>G</td>
<td>PS</td>
<td>A</td>
<td>CD</td>
<td>WT</td>
<td>U/S</td>
<td>R/W</td>
<td>P=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available for OS (page table location on disk) P=0

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>XD</td>
</tr>
<tr>
<td>62</td>
<td>Unused</td>
</tr>
<tr>
<td>52</td>
<td>Page physical base address</td>
</tr>
<tr>
<td>51</td>
<td>Unused</td>
</tr>
<tr>
<td>12</td>
<td>Unused</td>
</tr>
<tr>
<td>11</td>
<td>Unused</td>
</tr>
<tr>
<td>9</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>D</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>CD</td>
</tr>
<tr>
<td>4</td>
<td>WT</td>
</tr>
<tr>
<td>3</td>
<td>U/S</td>
</tr>
<tr>
<td>2</td>
<td>R/W</td>
</tr>
<tr>
<td>1</td>
<td>P=1</td>
</tr>
<tr>
<td>0</td>
<td>P=0</td>
</tr>
</tbody>
</table>

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 address\u2014**

- VPN 1
- VPN 2
- VPN 3
- VPN 4
- VPO

**Physical address**

- CR3 (Physical address of L1 PT)

- L1 PT Page global directory
- L2 PT Page upper directory
- L3 PT Page middle directory
- L4 PT Page table

- L1 PTE
- L2 PTE
- L3 PTE
- L4 PTE

- 512 GB region per entry
- 1 GB region per entry
- 2 MB region per entry
- 4 KB region per entry

- PPN
- PPO

**Offset into physical and virtual page**

- 40
- 9
- 12

**Physical address of page**
Virtual Memory Of a Linux Process

- Program text (.text)
- Initialized data (.data)
- Uninitialized data (.bss)
- Runtime heap (via malloc)
- Memory mapped region for shared libraries
- Physical memory
- Process-specific data structures (e.g., page tables, task and mm structs, kernel stack)
- Kernel code and data

Virtual Memory Of a Linux Process

- Process virtual memory
- Kernel virtual memory

- Physical memory
- Kernel code and data
- Process-specific data structures (e.g., page tables, task and mm structs, kernel stack)

- User stack
- Memory mapped region for shared libraries
- Runtime heap (via malloc)
- Uninitialized data (.bss)
- Initialized data (.data)
- Program text (.text)

Different for each process

Identical for each process

%esp

brk

0x08048000 (32)
0x00400000 (64)
Linux Virtual Memory Areas
Process virtual memory

0

Segmentation fault: accessing a non-existing page

Protection exception: e.g., violating permissions by writing to a read-only page

Normal page fault
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

Process 1
virtual memory

Physical memory

Process 2
virtual memory

Shared object
Shared Objects – After Sharing
Private Copy on Write Objects

Before Writing to Copy on write object

Process 1 virtual memory

Physical memory

Process 2 virtual memory

Private copy-on-write object
Private Copy on Write Objects
(After writing)
Memory mapping by loader for the user address space
mmap arguments interpretation

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