Lecture 20: Virtual Memory (cont'd)
Review: Address Translation

Stack
Heap
Data
Code

Virtual Address
MMU
invalid
Exception

Physical Address
Data

Code
Data
Stack
Heap
Review: Paging

Virtual Memory
- Page 7
- Stack
- Page 6
- Heap
- Page 5
- Data
- Page 4
- Code
- Page 3
- Frame 0
- Page 2
- Page 1
- Page 0

Physical Memory
- Frame 17
- Frame 16
- Frame 15
- Frame 14
- Frame 13
- Frame 12
- Frame 11
- Frame 10
- Frame 9
- Frame 8
- Frame 7
- Frame 6
- Frame 5
- Frame 4
- Frame 3
- Frame 2
- Frame 1
- Frame 0
Review: Virtual Pages

- NULL page or access not allowed
- SegFault
- Page Fault
- Invalid page

Data

<table>
<thead>
<tr>
<th>page#</th>
<th>Frame</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47</td>
<td>R,W</td>
</tr>
<tr>
<td>0</td>
<td>NULL</td>
<td>R,W</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>R,W</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>R,X</td>
</tr>
</tbody>
</table>

vaddr → MMU → paddr = Frame[page#] + offset
Review: Problems with Paging

- **Memory Consumption**: page table is really big
  - Example: consider 64-bit address space, 4KB ($2^{12}$) page size, assume each page table entry is 8 bytes.
  - Larger pages increase internal fragmentation

- **Performance**: every data/instruction access requires *two* memory accesses:
  - One for the page table
  - One for the data/instruction
Traditional Paging

- Page table is stored in physical memory
- Implemented as array of page table entries
- Page Table Base Register (PTBR) stores physical address of beginning of page table
- Page table entries are accessed by using the page number as the index into the page table
Two-level Page Tables

- Page table is stored in virtual memory pages
- Page directory is stored in physical memory (page table for the page table)
- Implemented as array of page directory entries
- Page Table Base Register (PTBR) stores physical address of beginning of page directory
Two-level Page Tables

- only store in-use page table entries in physical memory
- easier to allocate page table
- more memory accesses
Exercise 1: Two-level Page Tables

• Assume you are working on an architecture with a 32-bit virtual address space in which idx1 is 4 bits, idx2 is 12 bits, and offset is 16 bits.

  4 bit idx1  12 bit idx2  16 bit offset

• How big is a page in this architecture? \(2^{16}\) bytes = 64 KB

• How big is a page table entry in this architecture? 16 bytes
Exercise 2: Two-level Page Tables

Assume you are still working on that architecture.

Compute the physical address corresponding to each of the virtual address (or answer "invalid"):

a) 0x00000000 0x00470000
b) 0x20022002 invalid
c) 0x10015555 0xCAFE5555

<table>
<thead>
<tr>
<th>Frame 0</th>
<th>V</th>
<th>Frame</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>1</td>
<td>0x0047</td>
<td>R,W</td>
</tr>
<tr>
<td>0x1</td>
<td>0</td>
<td>NULL</td>
<td>R,W</td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td>0x0013</td>
<td>R,W</td>
</tr>
<tr>
<td>0x3</td>
<td>1</td>
<td>0x0042</td>
<td>R,X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>V</th>
<th>Frame</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>NULL</td>
<td>R,W</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0xCAFE</td>
<td>R,W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame 2</th>
<th>V</th>
<th>Frame</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>0x002A</td>
<td>R</td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td>0xCAFE</td>
<td>R,W</td>
</tr>
<tr>
<td>0x2</td>
<td>0</td>
<td>NULL</td>
<td>R,W</td>
</tr>
<tr>
<td>0x3</td>
<td>0</td>
<td>13</td>
<td>R,W</td>
</tr>
</tbody>
</table>
Problem: How big does the page directory get? 128 MB

- Assume you have a 48-bit address space
- Assume you have 4KiB pages
- Assume you have 8 byte page table entries/page directory entries

Goal: Page Table Directory should fit in one frame

Multi-level Page Tables: add additional level(s) to tree
Review: Problems with Paging

- **Memory Consumption:** page table is really big
  - Example: consider 64-bit address space, 4KB \((2^{12})\) page size, assume each page table entry is 8 bytes.
  - Larger pages increase internal fragmentation

- **Performance:** every data/instruction access requires two memory accesses:
  - One for the page table
  - One for the data/instruction
Translation-Lookaside Buffer (TLB)

- General idea: if address translation is slow, cache some of the answers

- **Translation-lookaside buffer** is an address translation cache that is built into the MMU
Exercise 3: TLB

<table>
<thead>
<tr>
<th>idx</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>03</td>
<td>B</td>
<td>0</td>
<td>07</td>
<td>6</td>
<td>1</td>
<td>28</td>
<td>3</td>
<td>0</td>
<td>01</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>3E</td>
<td>4</td>
<td>1</td>
<td>0B</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2A</td>
<td>A</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1F</td>
<td>8</td>
<td>1</td>
<td>07</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>07</td>
<td>3</td>
<td>0</td>
<td>2A</td>
<td>A</td>
<td>0</td>
<td>1E</td>
<td>2</td>
<td>0</td>
<td>21</td>
<td>B</td>
</tr>
</tbody>
</table>

- Assume you are running on an architecture with a one-level page table with 4096 byte pages. For each of the following virtual addresses, determine whether the address translation is stored in the TLB. If so, give the corresponding physical address
  - 0x7E37C
  - 0x16A48
Exercise 3: TLB

<table>
<thead>
<tr>
<th>idx</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
<th>v</th>
<th>tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>03</td>
<td>B</td>
<td>0</td>
<td>07</td>
<td>6</td>
<td>1</td>
<td>28</td>
<td>3</td>
<td>0</td>
<td>01</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>3E</td>
<td>4</td>
<td>1</td>
<td>0B</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2A</td>
<td>A</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1F</td>
<td>8</td>
<td>1</td>
<td>07</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>07</td>
<td>3</td>
<td>0</td>
<td>2A</td>
<td>A</td>
<td>0</td>
<td>1E</td>
<td>2</td>
<td>0</td>
<td>21</td>
<td>B</td>
</tr>
</tbody>
</table>

- Assume you are running on an architecture with a one-level page table with 4096 byte pages. For each of the following virtual addresses, determine whether the address translation is stored in the TLB. If so, give the corresponding physical address
  - 0x7E37C
  - 0x837C
  - 0x16A48
  - TLB miss
Example: The Linux x86 Address Space

- Use "only" 48-bit addresses (top 16 bits not used)
- 4KiB pages by default
  - supports larger "superpages"
- Four-level page table
- Physical memory stores memory pages, memory-mapped files, cached file pages
- Updates are periodically written to disk by background processes
- Page eviction algorithm uses variant of LRU called 2Q
  - approximates LRU with clock
  - maintains two lists (active/inactive)
- Stack is marked non-executable
- Virtual address of stack/heap start are randomized each time process is initialized
Example: Core i7 Address Translation

Virtual address (VA)

CPU

VPN  VPO

36  12

TLBT TLBI

32  4

TLB

hit

miss

L1 TLB (16 sets, 4 entries/set)

VPN1 VPN2 VPN3 VPN4

9 9 9 9

CR3

PTE PTE PTE PTE

Page tables

Result

32/64

L1 hit

L1 d-cache

(64 sets, 8 lines/set)

L1 miss

L2, L3, and main memory

PPN  PPO

40 12

CT CI CO

Physical address (PA)
Exercise 4: Feedback

1. Rate how well you think this recorded lecture worked
   1. Better than an in-person class
   2. About as well as an in-person class
   3. Less well than an in-person class, but you still learned something
   4. Total waste of time, you didn't learn anything

2. How much time did you spend on this video (including exercises)?

3. Do you have any particular questions you’d like me to address in this week’s problem session?

4. Do you have any other comments or feedback?