Operating Systems Programming

Discussion Section
February 24, 2010
Plan for Today

• Project Stuff
• Memory, virtual addressing, partitions, and paging (at least)!
• Nachos demo/walkthrough
Project 2 Timeline

• **Initial design doc:** due Thursday March 4\textsuperscript{th}

• **Design review** with me: Monday March 8\textsuperscript{th} and Tuesday March 9\textsuperscript{th}. I’ll add times on Wed March 10\textsuperscript{th} if necessary.

• **Code:** due Monday March 14\textsuperscript{th}

• **Final design doc:** due Tuesday March 15\textsuperscript{th}

• **Peer evaluations:** also Tuesday March 15\textsuperscript{th}
Initial Design Doc Redux

• Only 10 points, if you got a zero on the design doc but didn’t miss any other points you will still get 90/100 on the project!
Virtual Addressing

• Problem: want to run multiple processes using same physical memory
  – Need protection so one process doesn’t stomp on another one

• Solution: use a level of indirection, processes use virtual addresses, OS translates to physical addresses

• The set of virtual addresses a process can reference is its address space
  – Many different possible mechanisms for translating virtual addresses to physical addresses
Old technique #1: Fixed partitions

• Physical memory is broken up into fixed partitions
  – Partitions may have different sizes, but partitioning never changes
  – Hardware requirement: base register, limit register
    • physical address = virtual address + base register
    • Base register loaded by OS when it switches to a process
  – How do we provide protection?
    • if (physical address > base + limit) then... ?

• Advantages
  – Simple

• Problems
  – Internal fragmentation: the available partition is larger than what was requested
  – External fragmentation: two small partitions left, but one big process – what sizes should the partitions be??
Mechanics of fixed partitions

- Offset: virtual address
- Limit register: 2K
- Base register: P2’s base: 6K
- Physical memory:
  - Partition 0
  - Partition 1
  - Partition 2
  - Partition 3

Decision:
- Yes: Limit register < Offset + Base register
- No: Raise protection fault
Old technique #2: Variable partitions

- Obvious next step: physical memory is broken up into partitions dynamically – partitions are tailored to programs
  - Hardware requirements: base register, limit register
  - physical address = virtual address + base register
  - How do we provide protection?
    - if (physical address > base + limit) then... ?

- Advantages
  - No internal fragmentation
    - Simply allocate partition size to be just big enough for process (assuming we know what that is!)

- Problems
  - External fragmentation
    - As we load and unload jobs, holes are left scattered throughout physical memory
    - Slightly different than the external fragmentation for fixed partition systems
Mechanics of variable partitions

- Offset (virtual address)
- Limit register: P3's size
- Base register: P3's base
- Physical memory:
  - Partition 0
  - Partition 1
  - Partition 2
  - Partition 3
  - Partition 4

Flowchart:
- Offset < P3's size (yes/no)
- Raise protection fault if no
Dealing with fragmentation

- Swap a program out
- Re-load it, adjacent to another
- Adjust its base register
- “Lather, rinse, repeat”
- Ugh
Modern technique: Paging

- Solve the external fragmentation problem by using fixed sized units in both physical and virtual memory.
User’s perspective

• Processes view memory as a contiguous address space from bytes 0 through N
  – virtual address space (VAS)
• In reality, virtual pages are scattered across physical memory frames – not contiguous as earlier
  – virtual-to-physical mapping
  – this mapping is invisible to the program
• Protection is provided because a program cannot reference memory outside of its VAS
  – the virtual address 0xDEADBEEF maps to different physical addresses for different processes
• Note: For this project you will run programs where the entire virtual address space can be made resident in memory – no “page faults”
Address translation

• Translating virtual addresses
  – a virtual address has two parts: virtual page number & offset
  – virtual page number (VPN) is index into a page table
  – page table entry contains physical page number (PPN)
  – physical address is PPN::offset

• Page tables
  – managed by the OS
  – map virtual page number (VPN) to physical page number (PPN)
    • VPN is simply an index into the page table
  – one page table entry (PTE) per page in virtual address space
    • i.e., one PTE per VPN
Paging (K byte pages)

Page table

<table>
<thead>
<tr>
<th>page</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Process 0

Virtual address space

<table>
<thead>
<tr>
<th>page</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Physical memory

<table>
<thead>
<tr>
<th>page frame</th>
<th>0</th>
<th>K</th>
<th>2K</th>
<th>3K</th>
</tr>
</thead>
<tbody>
<tr>
<td>page frame</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page frame</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process 1

Virtual address space

<table>
<thead>
<tr>
<th>page</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Page fault – next lecture!
Mechanics of address translation

- Virtual address
  - Virtual page #
  - Offset

- Page table
  - Page frame #

- Physical address
  - Page frame #
  - Offset

- Physical memory
  - Page frame 0
  - Page frame 1
  - Page frame 2
  - Page frame 3
  - …
  - Page frame Y
Example of address translation

• Assume 32 bit addresses
  – assume page size is 4KB (4096 bytes, or $2^{12}$ bytes)
  – VPN is 20 bits long ($2^{20}$ VPNs), offset is 12 bits long

• Let’s translate virtual address 0x13325328
  – VPN is 0x13325, and offset is 0x328
  – assume page table entry 0x13325 contains value 0x03004
    • Physical page number is 0x03004
    • VPN 0x13325 maps to PPN 0x03004
  – physical address = PPN::offset = 0x03004328
Page Table Entries (PTEs)

1 1 1 2

<table>
<thead>
<tr>
<th>V</th>
<th>R</th>
<th>M</th>
<th>prot</th>
<th>page frame number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

- **PTE’s control mapping**
  - the **valid bit** says whether or not the PTE can be used
    - says whether or not a virtual address is valid
    - it is checked each time a virtual address is used
  - the **referenced bit** says whether the page has been accessed
    - it is set when a page has been read or written to
  - the **modified bit** says whether or not the page is dirty
    - it is set when a write to the page has occurred
  - the **protection bits** control which operations are allowed
    - read, write, execute
  - the **page frame number** determines the physical page
    - physical page start address = PFN
Paging advantages

• Easy to allocate physical memory
  – physical memory is allocated from free list of physical pages
    • to allocate a page, just remove it from the free list
  – external fragmentation is not a problem!

• Leads naturally to virtual memory
  – entire program need not be memory resident
  – take page faults using “valid” bit
  – all “chunks” are the same size (page size)
  – but paging was originally introduced to deal with external fragmentation, not to allow programs to be partially resident
Paging disadvantages

• Can still have internal fragmentation
  – process may not use memory in exact multiples of pages
• Memory reference overhead
  – 2 references per address lookup (page table, then memory)
  – solution: use a hardware cache to absorb page table lookups
    • translation lookaside buffer (TLB) – next class
• Memory required to hold page tables can be large
  – need one PTE per page in virtual address space
  – 32 bit AS with 4KB pages = $2^{20}$ PTEs = 1,048,576 PTEs
  – 4 bytes/PTE = 4MB per page table
    • OS’s have separate page tables per process
    • 25 processes = 100MB of page tables
  – solution: page the page tables (!!!)
    • (ow, my brain hurts...more later)