Announcements

- **Reading:**
  - Today: Chapter 8.4-8.6 (8th Ed)

- **Midterm #1:**
  - Was returned

### Midterm #1 Results

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>2.00</td>
<td>0.00</td>
<td>7.00</td>
<td>6.00</td>
<td>0.00</td>
<td>0.00</td>
<td>29.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>20</td>
<td>20</td>
<td>16</td>
<td>12</td>
<td>12</td>
<td>20</td>
<td>93.00</td>
</tr>
<tr>
<td>Mean</td>
<td>13.38</td>
<td>15.82</td>
<td>14.16</td>
<td>11.87</td>
<td>6.73</td>
<td>12.38</td>
<td>74.33</td>
</tr>
<tr>
<td>Std Dev</td>
<td>3.60</td>
<td>5.60</td>
<td>2.39</td>
<td>0.89</td>
<td>3.21</td>
<td>4.15</td>
<td>12.03</td>
</tr>
</tbody>
</table>
Managing Memory

- Main memory is big, but what if we run out
  - use virtual memory
  - keep part of memory on disk
    - bigger than main memory
    - slower than main memory

- Want to have several program in memory at once
  - keeps processor busy while one process waits for I/O
  - need to protect processes from each other
  - have several tasks running at once
    - compiler, editor, debugger
    - word processing, spreadsheet, drawing program

- Use *virtual addresses*
  - look like normal addresses
  - hardware translates them to *physical addresses*
Advantages of Virtual Addressing

- Can assign non-contiguous regions of physical memory to programs
- A program can only gain access to its mapped pages
- Can have more virtual pages than the size of physical memory
  - pages that are not in memory can be stored on disk
- Every program can start at (virtual) address 0
Paging

- Divide physical memory into fixed sized chunks called *pages*
  - typical pages are 512 bytes to 64KB bytes
  - When a process is to be executed, load the pages that are *actually used* into memory

- Have a table to map virtual pages to physical pages

- Consider a 32 bit addresses
  - 4096 byte pages (12 bits for the page)
  - 20 bits for the page number
Problems with Page Tables

- One page table can get very big
  - $2^{20}$ entries (for most programs, most items are empty)
- solution 1: use a hierarchy of page tables

![Diagram of page table hierarchy]

Virtual Address → Page Directory (10 bits) → Page Table (10 bits) → Physical Page # → Main Memory
Faster Mapping from Virtual to Physical Addresses

- need hardware to map between physical and virtual addresses
  - can require multiple memory references
  - this can be slow
- answer: build a cache of these mappings
  - called a translation look-aside buffer (TLB)
  - associative table of virtual to physical mappings
  - typically 16-64 entries

<table>
<thead>
<tr>
<th>Virtual Page</th>
<th>Physical Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bits</td>
<td>20 bits</td>
</tr>
</tbody>
</table>

For Intel x86
Super Pages

- **TLB Entries**
  - Tend to be limited in number
  - Can only refer to one page

- **Idea**
  - Create bigger pages
  - 4MB instead of 4KB
  - One TLB entry covers more memory
Inverted Page Tables

- Solution to the page table size problem
- One entry per page frame of physical memory
  
  \(<\text{process-id, page-number}>\)
  
  - each entry lists process associated with the page and the page number
  - when a memory reference:
    
    \(<\text{process-id, page-number, offset}>\) occurs, the inverted page table is searched (usually with the help of a hashing mechanism)
    
    - if a match is found in entry \(i\) in the inverted page table, the physical address \(<i, offset>\) is generated
    
  - The inverted page table does not store information about pages that are not in memory
    
    - page tables are used to maintain this information
    
    - page table need only be consulted when a page is brought in from disk