Announcements

- **Reading:**
  - Today: Chapter 9.4-9.6
  - Thursday: Chapter 10

- **Office hours are only for those who attend class**

- **Midterm was returned**
  - All re-grade requests must:
    - Be in writing
    - Be submitted by 10:45 AM 3/18/03
  - Any re-grade request will result in the **entire** exam being re-graded higher or lower as appropriate.

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>Tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Max</td>
<td>20</td>
<td>25</td>
<td>20</td>
<td>15</td>
<td>20</td>
<td>96</td>
</tr>
<tr>
<td>Ave</td>
<td>14.8</td>
<td>16.1</td>
<td>9.4</td>
<td>7.8</td>
<td>12.5</td>
<td>60.6</td>
</tr>
<tr>
<td>StdDec</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18.6</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 64k 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

![Diagram of paging system]

Virtual Address: Location

Location: Present Rd/Write

20 bits: Page Table

12 bits: Main Memory
Problems with Page Tables

- One page table can get very big
  - $2^{20}$ entries (for most programs, most items are empty)
- solution1: use a hierarchy of page tables

```
Virtual Address
     | 10 bits
     | 12 bits
     | 10 bits
     | Page Directory
     | 10 bits
     | Pg Tbl Ptr
     | 12 bits
     | Physical Page #
     | +

   Page Table

Main 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
Inverted Page Table Example (PPC)

Virtual Address

<table>
<thead>
<tr>
<th>Seg</th>
<th>Page #</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
<td>12</td>
</tr>
</tbody>
</table>

16 Segment Registers (per process)

24

Virtual Segment ID

Hash Function

VS ID (40)

Physical page (20)

Status bits

Page Table Entry (PTE)

Page Table Group
8 page table entries

Page Table
(variable size)

one per system

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></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20 bits 20 bits

For Intel x86