Announcements

- Reading 8.7, 9.1-9.4
- Suggested problems
  - 8.10, 8.12, 8.17
- Midterm #1 was returned
Midterm Results

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>51.4</td>
</tr>
<tr>
<td>Min</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Max</td>
<td>15</td>
<td>12</td>
<td>20</td>
<td>22</td>
<td>15</td>
<td>15</td>
<td>88</td>
</tr>
<tr>
<td>S.D.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17.3</td>
</tr>
</tbody>
</table>

- Charles will go over the questions in section on Wed.
- **Notes:**
  - will see synchronization again
    - maybe next midterm, for sure on the final
    - understand: Critical Section, Deadlock, Starvation
  - don’t think your project grade will improve your overall grade
  - in addition to reading material, do the problems!
- **Appeal Process**
  - 24 hour cooling off period
  - reserve right to regrade your entire midterm
Inverted Page Tables

- Solution to the page table size problem
- One entry per page frame of physical memory

  - `<process-id, page-number>`
  - each entry lists process associated with the page and the page number
  - when a memory reference:
    - `<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
  - Seg
  - Page #
  - Byte

- 16 Segment Registers (per process)

- Page Table Group
  - 8 page table entries

- Page Table
  - (variable size)
  - one per system

- Virtual Segment ID
  - VS ID (40)
  - Physical page (20)

- Status bits

- Hash Function

- Page Table Entry (PTE)

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

For Intel x86
Sharing Memory

● Pages can be shared
  – several processes may share the same code or data
  – several pages can be associated with the same page frame
  – given read-only data, sharing is always safe

● when writes occur, decide if processes share data
  – operating systems often implement “copy on write” - pages are shared until a process carries out a write
    • when a shared page is written, a new page frame is allocated
    • writing process owns the modified page
    • all other sharing processes own the original page
  – page could be shared
    • processes use semaphores or other means to coordinate access
What Happens when a virtual address has no physical address?

- called a *page fault*
  - a trap into the operating system from the hardware

- caused by: the first use of a page
  - called *demand paging*
  - the operating system allocates a physical page and the process continues
  - read code from disk or init data page to zero

- caused by: a reference to an address that is not valid
  - program is terminated with a “segmentation violation”

- caused by: a page that is currently on disk
  - read page from disk and load it into a physical page, and continue the program

- caused by: a copy on write page
Page State (hardware view)

- Page frame number (location in memory or on disk)
- **Valid Bit**
  - indicates if a page is present in memory or stored on disk
- **A modify or dirty bit**
  - set by hardware on write to a page
  - indicates whether the contents of a page have been modified since the page was last loaded into main memory
  - if a page has not been modified, the page does not have to be written to disk before the page frame can be reused
- **Reference bit**
  - set by the hardware on read/write
  - cleared by OS
  - can be used to approximate LRU page replacement
- **Protection attributes**
  - read, write, execute
OS Protection attributes (Win32)

- NOACCESS: attempts to read, write or execute will cause an access violation
- READONLY: attempts to write or execute memory in this region cause an access violation
- READWRITE: attempts to execute memory in this region cause an access violation
- EXECUTE: Attempts to read or write memory in this region cause an access violation
- EXECUTE_READ: Attempts to write to memory in this region cause an access violation
- EXECUTE_READ_WRITE: Do anything to this page
- WRITE_COPY: Attempts to write will cause the system to give a process its own copy of the page. Attempts to execute cause access violation
- EXECUTE_WRITE_COPY: Attempts to write will cause the system to give a process its own copy of a page. Can’t cause an access violation