Storage

Rachit Agarwal
Anne Bracy
<table>
<thead>
<tr>
<th></th>
<th>4410</th>
<th>4411</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Homeworks</strong></td>
<td>4 + 1</td>
<td>-</td>
</tr>
<tr>
<td><strong>Projects</strong></td>
<td>2 + 1</td>
<td>3 + 1</td>
</tr>
<tr>
<td><strong>Exams</strong></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Operating Systems — Where are we?

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW1</td>
<td>20</td>
<td>19.2</td>
<td>20</td>
<td>1.66</td>
</tr>
<tr>
<td>HW2</td>
<td>35</td>
<td>26.71</td>
<td>26</td>
<td>5.34</td>
</tr>
<tr>
<td>HW3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-P1</td>
<td>80</td>
<td>73</td>
<td>80</td>
<td>14.46</td>
</tr>
<tr>
<td>10-P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11-P1</td>
<td>100</td>
<td>83.2</td>
<td>84</td>
<td>8.38</td>
</tr>
<tr>
<td>11-P2</td>
<td>100</td>
<td>82.5</td>
<td>84</td>
<td>10.21</td>
</tr>
<tr>
<td>11-P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Operating Systems — Recap

• Processes and Threads
  • Abstraction of a computer (CPU, storage, network, …)

• Synchronization, Deadlock
  • Sharing resources “correctly”

• CPU Scheduling
  • Sharing CPU resources “efficiently”

• Networking
  • Sharing network resources “efficiently”
Operating Systems — Storage (Next 7 lectures)

Sharing Storage “efficiently” and ...

Diagram:
- Registers
- Caches
- Memory
- SSD
- Disk

- Faster (?)
- More expensive
- Lower capacity
Goal of Memory Management

• Sharing of memory across processes
  • Why share memory?
  • Why processes? Why not threads?

• Time-sharing
  • Load one program onto machine
  • Execute to completion
  • Problem: Long I/O leads to inefficiencies

• Space-sharing
  • Simultaneously running multiple processes
Memory — Sharing

Challenges of space-sharing

• Protection
  • Across processes

• Naming and addressing
  • Identify physical addresses?

• Efficiency
  • Utilization? Using faster memory?
Memory — Sharing

Option 1: Load all

- Load all processes into memory
- Switch between them under OS control
- Must relocate program when load it
- Big Problem: Protection
  - A bug in one processes can kill others
- Guess who used it?
  - MS-DOS, MS-Windows
Option 2: Copy on load

• Copy entire process memory to disk during I/O
• Copy back when it restarts
• No need to relocate

• Big Problem: Performance
  • Why?

• Guess who used it?
• Early versions of Unix
Memory — Sharing

Option 3: Access Check

• Give each program a piece of memory
• Upon each memory reference
  • check that it stays within its address space
• How to implement this?
  • Address translation
  • Base and bound registers
• Cray-1
Memory Sharing — Access Check

Address Translation (more later)

• Program generates virtual addresses

• “Virtual addresses” translated into physical addresses
Memory Sharing — Access Check

Base and Bound registers

- Base: Physical address corresponding to virtual address 0
- Bound: higher allowable virtual address
Memory Sharing — Model

Overall model

• Each process has a virtual address space

• Internally mapped to physical address space
  • Virtual to Physical allocation?
Memory Sharing — Model

Virtual to Physical allocation

• First-Fit
  • Allocate first “hole” that is big enough

• Best-Fit
  • Allocate smallest “hole” that is big enough

• Worst-Fit
  • Allocate largest “hole” that is big enough
Virtual to Physical allocation: Problem?

• External Fragmentation

• Available physical memory, but fragmented

• Various options
  • Wait for space (problem?)
  • Make space (how?)
Memory Sharing — Model

Virtual to Physical allocation: Solution

• Allocations at “finer granularity”
  • Pages
• Break physical address space into fixed size pages
• Map Virtual address space to multiple pages
  • Non-contiguous
• Dynamic address translation