Multiprocessing: The Illusion

- Process provides each program with two key abstractions:
  - **Logical control flow**
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called *context switching*
  - **Private address space**
    - Each program seems to have exclusive use of main memory.
    - Provided by kernel mechanism called *virtual memory*
Multiprocessing: The Reality

- Computer runs many processes simultaneously
- Running program “top” on Mac
  - System has 123 processes, 5 of which are active
  - Identified by Process ID (PID)
Virtual Memory Goals

• **Isolation:** don’t want different process states collided in physical memory

• **Efficiency:** want fast reads/writes to memory

• **Sharing:** want option to overlap for communication

• **Utilization:** want best use of limited resource

• **Virtualization:** want to create illusion of more resources
Address Translation

Virtual Address → MMU → Exception

Data

Stack

Heap

Data

Code
Base-and-Bound

Physical Memory

Virtual Memory

Stack
Heap
Data
Code

Base
Bound

START

FINISH
Base-and-Bound

Stack

Heap

Data

Code

vaddr

MMU

Base

Bound

vaddr > Bound

Exception

paddr = vaddr + Base

Data
Exercise 1: Base-and-Bound

Assume that you are currently executing a process P with Base 0x1234 and Bound 0x100.

- What is the physical address that corresponds to the virtual address 0x47?
- What is the physical address that corresponds to the virtual address 0x123?
Evaluating Base-and-Bound

- **Isolation**: don’t want different process states collided in physical memory
  - ✔️

- **Efficiency**: want fast reads/writes to memory
  - ✔️

- **Sharing**: want option to overlap for communication
  - ✗

- **Utilization**: want best use of limited resource
  - ✗

- **Virtualization**: want to create illusion of more resources
  - ✗
Segmentation

- Stack
- Heap
- Data
- Code

Physical Memory
- DBound
- DBase
- HBound
- HBase
- SBound
- SBase
- CBound
- CBase
Segmentation

Stack

Heap

Data

Code

MMU

<table>
<thead>
<tr>
<th>Base</th>
<th>Bound</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R,W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R,W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R,W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R,X</td>
</tr>
</tbody>
</table>

paddr = Base[idx] + offset

offset > Bound[idx] or access not allowed

Exception

Data
Exercise 2: Segmentation

Assume that you are currently executing a process P with the following segment table:

<table>
<thead>
<tr>
<th>Base</th>
<th>Bound</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4747</td>
<td>0x80</td>
<td>R,W</td>
</tr>
<tr>
<td>0x2424</td>
<td>0x40</td>
<td>R,W</td>
</tr>
<tr>
<td>0x0023</td>
<td>0x80</td>
<td>R,W</td>
</tr>
<tr>
<td>0x1000</td>
<td>0x100</td>
<td>R,X</td>
</tr>
</tbody>
</table>

- What is the physical address that corresponds to the virtual address 0x000?
- What is the physical address that corresponds to the virtual address 0xC47?
Evaluating Segmentation

- **Isolation**: don’t want different process states collided in physical memory

- **Efficiency**: want fast reads/writes to memory

- **Sharing**: want option to overlap for communication

- **Utilization**: want best use of limited resource

- **Virtualization**: want to create illusion of more resources
Paging

Virtual Memory

- Stack
- Heap
- Data
- Code

Physical Memory

- Frame 0
- Frame 1
- Frame 2
- Frame 3
- Frame 4
- Frame 5
- Frame 6
- Frame 7
- Frame 8
- Frame 9
- Frame 10
- Frame 11
- Frame 12
- Frame 13
- Frame 14
- Frame 15
- Frame 16
- Frame 17
Paging

Stack

Heap

Data

Code

Exception

Page dimensions: 720.0x540.0

Data

<table>
<thead>
<tr>
<th>MMU</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>R,W</td>
</tr>
<tr>
<td>NULL</td>
<td>R,W</td>
</tr>
<tr>
<td>13</td>
<td>R,W</td>
</tr>
<tr>
<td>42</td>
<td>R,X</td>
</tr>
</tbody>
</table>

paddr = Frame[page#] offset

Page not allowed

Stacking

Heap

Data

Code

vaddr

page# offset

Frame

access not allowed
Memory as a Cache

- each page table entry has a valid bit
- for valid entries, frame indicates physical address of page in memory
- a page fault occurs when a program requests a page that is not currently in memory
  - takes time to handle, so context switch
  - evict another page in memory to make space (which one?)

<table>
<thead>
<tr>
<th>v</th>
<th>Frame</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47</td>
<td>R,W</td>
</tr>
<tr>
<td>0</td>
<td>NULL</td>
<td>R,W</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>R,W</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>R,X</td>
</tr>
</tbody>
</table>
Thrashing

- working set is the collection of a process's pages required in a given time interval
- if it doesn't fit in memory, the program will thrash
Exercise 3: Paging

Assume that you are currently executing a process P with the following page table on a system with 256 byte pages:

<table>
<thead>
<tr>
<th>v</th>
<th>Frame</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>1</td>
<td>0x47</td>
</tr>
<tr>
<td>249</td>
<td>1</td>
<td>0x24</td>
</tr>
<tr>
<td>248</td>
<td>0</td>
<td>NULL</td>
</tr>
<tr>
<td>247</td>
<td>0</td>
<td>0x23</td>
</tr>
</tbody>
</table>

- What is the physical address that corresponds to the virtual address 0xF947?
- What is the physical address that corresponds to the virtual address 0xF700?
Evaluating Paging

- **Isolation**: don’t want different process states collided in physical memory
- **Efficiency**: want fast reads/writes to memory
- **Sharing**: want option to overlap for communication
- **Utilization**: want best use of limited resource
- **Virtualization**: want to create illusion of more resources
Exercise 4: Feedback

1. Rate how well you think this recorded lecture worked
   1. Better than an in-person class
   2. About as well as an in-person class
   3. Less well than an in-person class, but you still learned something
   4. Total waste of time, you didn't learn anything

2. Do you have any comments or suggestions for future classes?