Memory Management
Memory Manager

• Requirements
  – Minimize primary memory access time
  – Maximize primary memory size
  – Primary memory must be cost-effective

• Today’s memory manager:
  – Allocates primary memory to processes
  – Maps process address space to primary memory
  – Minimizes access time using cost-effective memory configuration
Address Space vs Primary Memory

Process Address Space

Primary Memory

Mapped to object other than memory
Building the Address Space

- Compose elements

- Source code
- Library code
- Other objects

Translation
Building the Address Space

- Compose elements
- Adjust addresses
  - Translation time
  - Load time

Source code
Library code
Other objects
Translation
Secondary memory

Process address space
Building the Address Space

- Compose elements
- Adjust addresses
  - Translation time
  - Load time
- Allocate executable memory space
Memory Hierarchies

Executable Memory

Secondary Memory

Rotating Magnetic Memory

Optical Memory

Sequentially Accessed Memory

Primary Memory

Cache Memory

CPU Registers
Managing the Hierarchy

• Move across executable-secondary memory boundary (or lower) requires I/O operation

• Upward moves are _copy_ operations
  – Require allocation in upper memory
  – Image exists in both memories

• Updates are first applied to upper memory

• Downward move is (usually) _destructive_
  – Deallocate upper memory
  – Updates image in secondary memory
Memory Allocation

- Unused
- In Use

Operating System

- Process 3
- Process 0
- Process 2
- Process 1
Fixed-Partition Memory

$p_i$ needs $n_i$ units

<table>
<thead>
<tr>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region 0</td>
</tr>
<tr>
<td>Region 1</td>
</tr>
<tr>
<td>Region 2</td>
</tr>
<tr>
<td>Region 3</td>
</tr>
</tbody>
</table>

$N_0$ $N_1$ $N_2$ $N_3$
Fixed-Partition Memory -- Best-Fit

- Loader must adjust every address in the absolute module when placed in memory.
Fixed-Partition Memory -- First-Fit

Operating System

\begin{align*}
  \text{Region 0} & : p_i, N_0 \\
  \text{Region 1} & : N_1 \\
  \text{Region 2} & : N_2 \\
  \text{Region 3} & : N_3
\end{align*}
Fixed-Partition Memory -- Next-Fit

Operating System

Region 0

Region 2

Region 3

$p_i$

$p_{i+1}$

$N_0$

$N_1$

$N_2$

$N_3$
Variable Partition Memory

Operating System
Variable Partition Memory

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process 0</td>
</tr>
<tr>
<td></td>
<td>Process 1</td>
</tr>
<tr>
<td></td>
<td>Process 2</td>
</tr>
<tr>
<td></td>
<td>Process 3</td>
</tr>
<tr>
<td></td>
<td>Process 4</td>
</tr>
</tbody>
</table>

• Loader must adjust every address in every absolute module when placed in memory
Variable Partition Memory

- External fragmentation
**Variable Partition Memory**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Operating System</th>
<th>Operating System</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 0</td>
<td>Process 0</td>
<td>Process 0</td>
<td>Process 0</td>
</tr>
<tr>
<td>Process 1</td>
<td>Process 6</td>
<td>Process 2</td>
<td>Process 2</td>
</tr>
<tr>
<td>Process 2</td>
<td>Process 5</td>
<td>Process 4</td>
<td>Process 4</td>
</tr>
<tr>
<td>Process 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Compaction** moves program in memory
Cost of Moving Programs

load R1, 0x02010

3F013010

Program loaded at 0x01000
Cost of Moving Programs

load R1, 0x02010

Program loaded at 0x01000

3F013010

Program loaded at 0x04000

3F016010

• Must run loader over program again!
Dynamic Memory Allocation

• Common to use *dynamically allocated* memory

• Process wants to change the size of its address space
  – Smaller? Creates an external fragment
  – Larger? Have to move/relocate the program

• Allocate “holes” in memory according to
  – Best- /Worst- / First- /Next-fit
Swapping

• Suppose there is high demand for executable memory
• Equitable policy might be to time-multiplex processes into the memory (also space-mux)
• Means that process can have its address space unloaded when it still needs memory
  – Usually only happens when it is blocked
• Have same problems as dynamic memory allocation
Dynamic Address Relocation

- Program loaded at 0x01000 ? Relocation Register = 0x01000
- Program loaded at 0x04000 ? Relocation Register = 0x04000
Runtime Bound Checking

CPU

Relative Address

Relocation Register

Limit Register

<

MARC

Interrupt
Strategies

• Fixed-Partition used only in batch systems

• Variable-Partition used everywhere (except in virtual memory)

• Swapping systems
  – Popularized in timesharing
  – Relies on dynamic address relocation
  – Now dated

• Virtual Memory
  – Paging -- mainstream in contemporary systems
  – Segmentation -- the future
NT Memory-mapped Files

- Open the file
- Create a *section object* (that maps file)
- Identify point in address space to place the file