Relocation

- Correct starting address when a program should start in the memory
- Different jobs will run at different addresses
  - When a program is linked, the linker must know at what address the program will begin in memory.
- Logical addresses, Virtual addresses
  - Logical address space, range (0 to max)
- Physical addresses, Physical address space
  - range (R+0 to R+max) for base value R.
- User program never sees the real physical addresses
- Who translates virtual to physical addresses?
  - Program rewriting at loading time
  - Help from relocation registers at execution time

Relocation Register

- Diagram showing the relationship between CPU Instruction Address, Base Register, Logical Address, Physical Address, and Memory.
Protection

Problem:
- How to prevent a malicious process to write or jump into other user's or OS partitions

Solutions:
- Memory protection code
- Base bounds registers
Memory Management (More...)

- Batch System
  - Multiprogramming with fixed partitions
  - In the memory until job finishes
  - Keep CPU busy

- Timesharing Systems
  - No enough memory to hold all active processes
  - Swapping (whole process)
  - Virtual memory (partial process)

Swapping

- Move the whole process to/from disk
- Allows several processes to share a fixed partition
- Processes that grow can be swapped out and swapped back in a bigger partition
Variable-Sized Partitions and Fragmentation

<table>
<thead>
<tr>
<th></th>
<th>Monitor</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Job 4</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Monitor</td>
<td>Job 1</td>
<td></td>
<td></td>
<td></td>
<td>Free</td>
</tr>
<tr>
<td>3</td>
<td>Monitor</td>
<td>Job 1</td>
<td>Job 5</td>
<td>Job 3</td>
<td>Job 4</td>
<td>Free</td>
</tr>
<tr>
<td>4</td>
<td>Monitor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Monitor</td>
<td>Job 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fixed Partitions ↔ Variable-sized Partitions

External Fragmentation

- How to solve it?

Compaction

- What is it? What does it looks like?
- Assumes all programs are relocatable
- Processes must be suspended during compaction
- Need be done only when fragmentation gets very bad

<table>
<thead>
<tr>
<th></th>
<th>Monitor</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Job 4</th>
<th>Job 5</th>
<th>Job 6</th>
<th>Job 7</th>
<th>Job 8</th>
<th>Job 9</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Monitor</td>
<td>Job 1</td>
<td></td>
<td></td>
<td></td>
<td>Job 5</td>
<td>Job 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Monitor</td>
<td>Job 1</td>
<td>Job 5</td>
<td>Job 3</td>
<td>Job 4</td>
<td>Job 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Monitor</td>
<td>Job 1</td>
<td>Job 5</td>
<td>Job 3</td>
<td>Job 4</td>
<td>Job 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Monitor</td>
<td>Job 1</td>
<td>Job 5</td>
<td>Job 3</td>
<td>Job 4</td>
<td>Job 6</td>
<td>Free</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Memory Management with Bitmaps

- Use bitmaps for memory status (free or allocated)
- Memory is divided into allocation units.
  - One allocation unit corresponds to 1 bit in the bitmap
  - 0: free, 1: allocated
- Size of allocation unit
  - The smaller the allocation unit, the larger the bitmap.
- Problem: allocation
  - When a new process arrives, the manager must find consecutive 0 bits in the map.
  - Searching a bitmap for a run of a given length is a slow operation.

![Bitmap Example]

Memory Management with Linked Lists

- Use a linked list of allocated and free memory segments (called hole)
  - sorted by the address or by the size

### Table

<table>
<thead>
<tr>
<th>Before X terminates</th>
<th>After X terminates</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) A X B</td>
<td>becomes A B</td>
</tr>
<tr>
<td>(b) A X</td>
<td>becomes A</td>
</tr>
<tr>
<td>(c) X B</td>
<td>becomes B</td>
</tr>
<tr>
<td>(d) X</td>
<td>becomes</td>
</tr>
</tbody>
</table>

Four neighbor combinations for the terminating process X
Memory Allocation Strategies

- **Best Fit**
  - Uses the hole whose size is equal to the need, or if none is equal, the hole that is larger but closest in size.
  - Problem: Creates small holes that can’t be used.

- **First Fit**
  - Uses the first available hole whose size is sufficient to meet the need.
  - Problem: Creates average size holes.

- **Next Fit**
  - Minor variation of first fit: search from the last stopped place.
  - Problem: slightly worse performance than first fit.

- **Worst Fit**
  - Uses the largest available hole.
  - Problem: Gets rid of large holes making it difficult to run large programs.

- **Quick Fit**
  - Maintains separate lists of holes for some of the more common sizes requested. When a request comes for placement it finds the closest fit.
  - A very fast scheme, but merge is expensive. If merge is not done memory will quickly fragment in a large number of holes into which no processes fit.

---

How Bad Is Fragmentation?

- Statistical arguments - Random sizes
- First-fit
- Given N allocated blocks
- 0.5*N blocks will be lost because of fragmentation
- Known as 50% RULE
Memory Management Problems

- Fixed partitions
  - suffer from internal fragmentation
- Variable-sized partitions
  - suffer from external fragmentation
- Compaction
  - suffer from overhead
- Overlays
  - painful for programmers
- Swapping
  - requires writing to disk sectors

Summary

- Swapping
- Variable Partitions
- Memory Management Problems

- Next lecture: Virtual Memory (I)