Memory Management

Chapter 7

B. Ramamurthy

Subdividing memory to accommodate multiple processes

Memory needs to be allocated efficiently to pack as many processes into memory as possible.

Memory Management Requirements

- **Relocation**
  - Programmer does not know where the program will be placed in memory when it is executed.
  - While the program is executing, it may be swapped to disk and returned to main memory at a different location.
  - Memory references must be translated in the code to actual physical memory address.

- **Protection**
  - Processes should not be able to reference memory locations in another process without permission.
  - Impossible to check addresses in programs since the program could be relocated.
  - Must be checked during execution.

- **Sharing**
  - Allow several processes to access the same portion of memory.
  - Better to allow each process (person) access to the same copy of the program rather than have their own separate copy.

- **Logical Organization**
  - Programs are written in modules.
  - Different degrees of protection given to modules (read-only, execute-only).
  - Share modules.
Memory Management Requirements

- Physical Organization
  - Memory available for a program plus its data may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory
  - Secondary memory cheaper, larger capacity, and permanent

Fixed Partitioning

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called internal fragmentation.

Fixed Partitioning

- Unequal-size partitions
  - Lessens the problem with equal-size partitions

Placement Algorithm with Partitions

- Equal-size partitions
  - Because all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - Can assign each process to the smallest partition within which it will fit
  - Queue for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition

One Process Queue per Partition

- When it is time to load a process into main memory, the smallest available partition that will hold the process is selected

One Common Process Queue
Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- Must use compaction to shift processes so they are contiguous and all free memory is in one block

Example Dynamic Partitioning

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>120 K</td>
<td>224 K</td>
<td>288 K</td>
<td></td>
</tr>
<tr>
<td>120 K</td>
<td>288 K</td>
<td>64 K</td>
<td></td>
</tr>
<tr>
<td>320 K</td>
<td>64 K</td>
<td>64 K</td>
<td></td>
</tr>
</tbody>
</table>

Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process
- Best-fit algorithm
  - chooses the block that is closest in size to the request
  - worst performer overall
  - since smallest block is found for process, the smallest amount of fragmentation is left; memory compaction must be done more often

Dynamic Partitioning Placement Algorithm

- First-fit algorithm
  - starts scanning memory from the beginning and chooses the first available block that is large enough.
- Next-fit
  - starts scanning memory from the location of the last placement and chooses the next available block that is large enough
  - Compaction is required to obtain a large block at the end of memory
Registers Used during Execution

- **Base register**
  - starting address for the process

- **Bounds register**
  - ending location of the process

- These values are set when the process is loaded and when the process is swapped in

The value of the base register is added to a relative address to produce an absolute address.

The resulting address is compared with the value in the bounds register.

If the address is not within bounds, an interrupt is generated to the operating system.

Paging

- Partition memory into small equal-size chunks and divide each process into the same size chunks.

- The chunks of a process are called pages and chunks of memory are called frames.

- Operating system maintains a page table for each process:
  - contains the frame location for each page in the process
  - memory address consist of a page number and offset within the page

<table>
<thead>
<tr>
<th>Frame Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process A</td>
<td></td>
<td></td>
<td></td>
<td></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>Process B</td>
<td></td>
<td></td>
<td></td>
<td></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>Process C</td>
<td></td>
<td></td>
<td></td>
<td></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>Process D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page Tables for Example