Memory Management

• Subdividing memory to accommodate multiple processes
• Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

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 (relocated)
  – Memory references must be translated in the code to actual physical memory address
Memory Management Requirements

- Protection
  - Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses at compile time
  - Must be checked at run time
  - Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
    - Operating system cannot anticipate all of the memory references a program will make
Memory Management Requirements

• Sharing
  – Allow several processes to access the same portion of memory
  – Better to allow each process access to the same copy of the program rather than have their own separate copy

Memory Management Requirements

• Logical Organization
  – Programs are written in modules
  – Modules can be written and compiled independently
  – Different degrees of protection given to modules (read-only, execute-only)
  – Share modules among processes
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
  – Programmer does not know how much space will be available

Fixed Partitioning

• Equal-size partitions
  – Any process whose size is less than or equal to the partition size can be loaded into an available partition
  – If all partitions are full, the operating system can swap a process out of a partition
  – A program may not fit in a partition. The programmer must design the program with overlays
Fixed Partitioning

• Fixed partitioning in main memory is inefficient.
  – Any program, no matter how small, occupies an entire partition.
  – What about the memory left over if the program does not fit perfectly.
  – This is called **internal fragmentation**.
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

Figure 7.3 Memory Assignment for Fixed Partitioning
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

Figure 7.4 The Effect of Dynamic Partitioning
Dynamic Partitioning Placement Algorithm

• Operating system must decide which free block to allocate to a process.
  – Let’s look at some algorithms.
• Best-fit algorithm
  – Chooses the block that is closest in size to the request
  – Despite its name: worst performer overall
  – Since smallest block is found for process, the smallest amount of fragmentation is left
    • leaves blocks too small to reallocate
  – Memory compaction must be done more often

• First-fit algorithm
  – Scans memory form the beginning and chooses the first available block that is large enough
  – Fastest
  – May have many process loaded in the front end of memory that must be searched over when trying to find a free block
Dynamic Partitioning Placement Algorithm

- **Next-fit**
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

![Diagram of memory configuration before and after allocation of 16 Mbyte block](image-url)
### Buddy System

- Entire space available is treated as a single block of $2^U$
- If a request of size $s$ such that $2^{U-1} < s \leq 2^U$, entire block is allocated
  - Otherwise block is split into two equal buddies
  - Process continues until smallest block greater than or equal to $s$ is generated

---

<table>
<thead>
<tr>
<th>1 Mbyte block</th>
<th>1 M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request 100 K</td>
<td>A = 128 K</td>
</tr>
<tr>
<td>Request 240 K</td>
<td>A = 128 K</td>
</tr>
<tr>
<td>Request 64 K</td>
<td>A = 128 K</td>
</tr>
<tr>
<td>Request 256 K</td>
<td>A = 128 K</td>
</tr>
<tr>
<td>Release B</td>
<td>A = 128 K</td>
</tr>
<tr>
<td>Release A</td>
<td>128 K</td>
</tr>
<tr>
<td>Request 75 K</td>
<td>E = 128 K</td>
</tr>
<tr>
<td>Release C</td>
<td>E = 128 K</td>
</tr>
<tr>
<td>Release E</td>
<td>512 K</td>
</tr>
<tr>
<td>Release D</td>
<td>1 M</td>
</tr>
</tbody>
</table>

*Figure 7.6 Example of Buddy System*
Relocation

- When program **loaded** into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions which means different absolute memory locations during execution (from swapping)
- Compaction will also cause a program to occupy a different partition which means different absolute memory locations
Addresses

- **Logical**
  - Reference to a memory location independent of the current assignment of data to memory
  - Translation must be made to the physical address
- **Relative**
  - Address expressed as a location relative to some known point
- **Physical**
  - The absolute address or actual location in main memory

---

**Figure 7.8 Hardware Support for Relocation**
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 or when the process is swapped in

Registers Used during Execution

• 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