Memory Management

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

Processors typically do not know in advance which other programs will be resident in main memory at the time of execution of their program

Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization

Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting

May need to relocate the process to a different area of memory
Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
- Mechanisms that support relocation also support protection

Sharing

- Advantageous to allow each process access to the same copy of the program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation support sharing capabilities

Logical Organization

- Memory is organized as linear

Programs are written in modules

- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Sharing on a module level corresponds to the user’s way of viewing the problem

- Segmentation is the tool that most readily satisfies requirements

Physical Organization

- The organization of the flow of information between main and secondary memory is a major system concern.
  - Can’t be left for the individual programmer to decide.
  - Why?
    - The main memory available for a program plus its data may be insufficient.
      - The programmer must engage in a practice known as overlaying.
    - In a multiprogramming environment, the programmer does not know at the time of coding how much space will be available or where that space will be.
  - It is clear, then, that the task of moving information between the two levels of memory should be a system responsibility.
Memory Partitioning

- Memory management brings processes into main memory for execution by the processor
  - Involves virtual memory
  - Based on segmentation and paging

- Partitioning
  - Used in several variations in some now-obsolete operating systems
  - Does not involve virtual memory

(1) Fixed partitioning

- The simplest scheme for managing this available memory is to partition it into regions with fixed boundaries.

- What will the placement algorithm be?

Disadvantages

- A program may be too big to fit in a partition
  - Program needs to be designed with the use of overlays

- Main memory utilization is inefficient
  - Any program, regardless of size, occupies an entire partition
  - Internal fragmentation
    - Wasted space due to the block of data loaded being smaller than the partition
Variable sized partitions

- Can you think of the placement algorithm?

![Diagram showing memory assignment for fixed partitioning](image)

Disadvantages

- The number of partitions specified at system generation time limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently

(2) Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as it requires
- This technique was used by IBM’s mainframe operating system, OS/MVT
Dynamic Partitioning

External Fragmentation

- Memory becomes more and more fragmented
- Memory utilization declines

Compaction

- Technique for overcoming external fragmentation
- OS shifts processes so that they are contiguous
- Free memory is together in one block
- Time consuming and wastes CPU time
**Placement Algorithms**

- **Best-fit**
  - Chooses the block that is closest in size to the request

- **First-fit**
  - Begins to scan memory from the beginning and chooses the first available block that is large enough

- **Next-fit**
  - Begins to scan memory from the location of the last placement and chooses the next available block that is large enough

(3) **Buddy System – a hybrid**

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- Memory blocks are available of size $2^k$ words, $L \leq K \leq U$, where
  - $2^L$ = smallest size block that is allocated
  - $2^U$ = largest size block that is allocated; generally $2^U$ is the size of the entire memory available for allocation

**Figure 7.6** Example of Buddy System

<table>
<thead>
<tr>
<th>Request</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 K</td>
<td>A = 128K</td>
</tr>
<tr>
<td>240 K</td>
<td>128K</td>
</tr>
<tr>
<td>64 K</td>
<td>128K</td>
</tr>
<tr>
<td>256 K</td>
<td>128K</td>
</tr>
<tr>
<td>Release B</td>
<td>A = 128K</td>
</tr>
<tr>
<td>Release A</td>
<td>128K</td>
</tr>
<tr>
<td>75 K</td>
<td>128K</td>
</tr>
<tr>
<td>Release C</td>
<td>A = 128K</td>
</tr>
<tr>
<td>Release D</td>
<td>128K</td>
</tr>
</tbody>
</table>

**Figure 7.7** Tree Representation of Buddy System
Relocation

- When the fixed partition scheme is used, we can expect a process will always be assigned to the same partition
  - A simple relocating loader can be used
  - When the process is first loaded, all relative memory references in the code are replaced by absolute main memory addresses, determined by the base address of the loaded process

- In the case of a equal-sized partitions and a single process queue for unequal-size partitions, a process may occupy different partitions during the course of its life
  - When a process image is first created, it is loaded into some partition in main memory; Later, the process may be swapped out
  - When it is subsequently swapped back in, it may be assigned to a different partition than the last time
  - The same is true for dynamic partitioning

- When compaction is used, processes are shifted while they are in main memory
  - Thus, the locations referenced by a process are not fixed
  - They will change each time a process is swapped in or shifted

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data to memory

- Relative
  - A particular example of logical address, in which the address is expressed as a location relative to some known point

- Physical or Absolute
  - Actual location in main memory