Operating System Support
Memory Management

Contents of Lecture:

- Operating System Objectives and Functions
- Types of Operating Systems
- Memory Management

References for This Lecture:

- ✓ William Stallings, Computer Organization and Architecture Designing For Performance, 9th Edition, Chapter 8: *Operating System Support*

Operating System Objectives and Functions:

- The Operating System as resource manager:
  - ✓ A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions. The OS is responsible for managing these resources.

- The OS as a control mechanism is unusual in two respects:
  - ✓ The OS functions in the same way as ordinary computer software – it is a program executed by the processor
  - ✓ The OS frequently relinquishes control and must depend on the processor to allow it to regain control

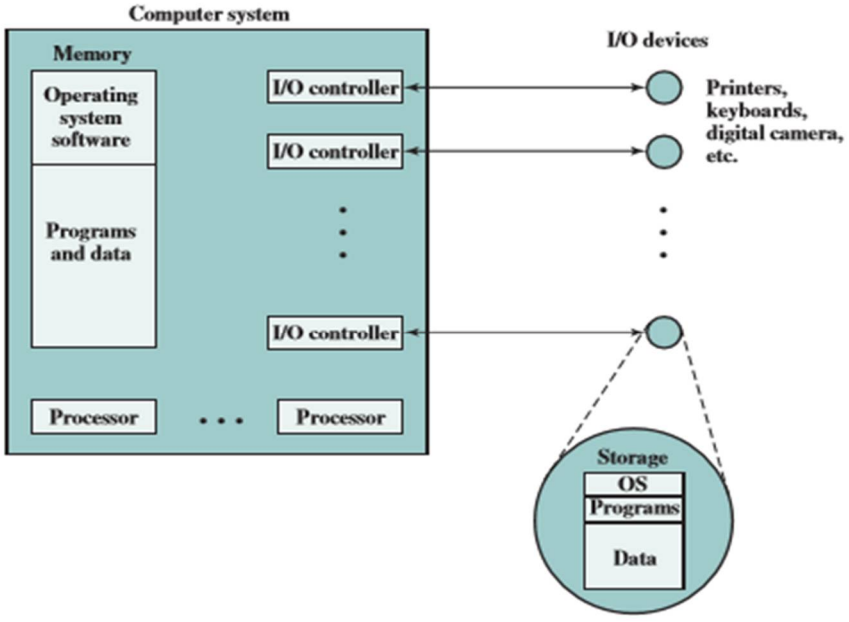


Figure 8.2 The Operating System as Resource Manager

- The main resources that are managed by the OS. A portion of the OS is in main memory. This includes the kernel, which contains the most frequently used functions in the OS and, at a given time, other portions of the OS currently in use. The remainder of main memory contains user programs and data.

- The allocation of this resource (main memory) is controlled jointly by the OS and memory-management hardware in the processor.

- The OS decides when an I/O device can be used by a program in execution, and controls access to and use of files.

- The processor itself is a resource, and the OS must determine how much processor time is to be devoted to the execution of a particular user program.

- In the case of a multiple-processor system, this decision must span all of the processors.

Types of Operating Systems:

- **Interactive system**
  - ✓ The user/programmer interacts directly with the computer to request the execution of a job or to perform a transaction
  - ✓ User may, depending on the nature of the application, communicate with the computer during the execution of the job

- **Batch system**
  - ✓ Opposite of interactive
  - ✓ The user's program is batched together with programs from other users and submitted by a computer operator
  - ✓ After the program is completed results are printed out for the user

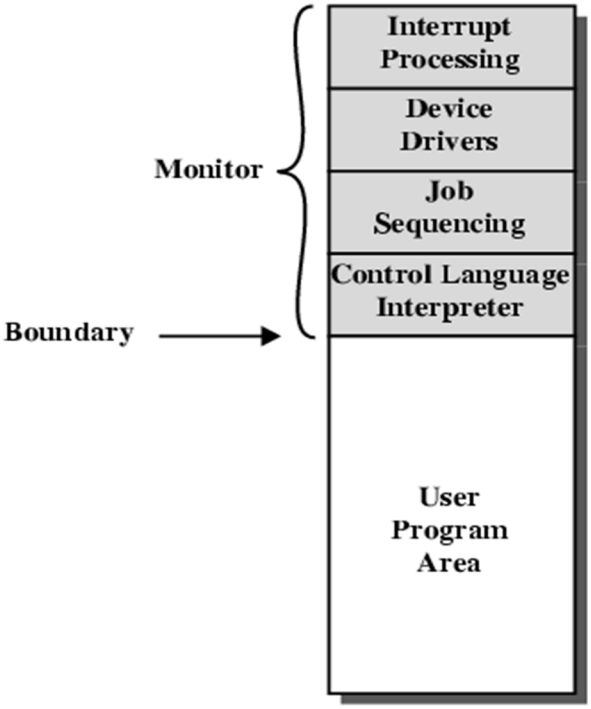
- **Multiprogramming**
  - ✓ An independent dimension specifies whether the system employs or not.
  - ✓ With multiprogramming, the attempt is made to keep the processor as busy as possible, by having it work on more than one program at a time.
  - ✓ Several programs are loaded into memory, and the processor switches rapidly among them.

- The alternative is a **uniprogramming** system that works only one program at a time.

Memory Management:

- **Uni-program**
  - ✓ Memory split into two
    - ➢ One for Operating System (monitor)
    - ➢ One for currently executing program

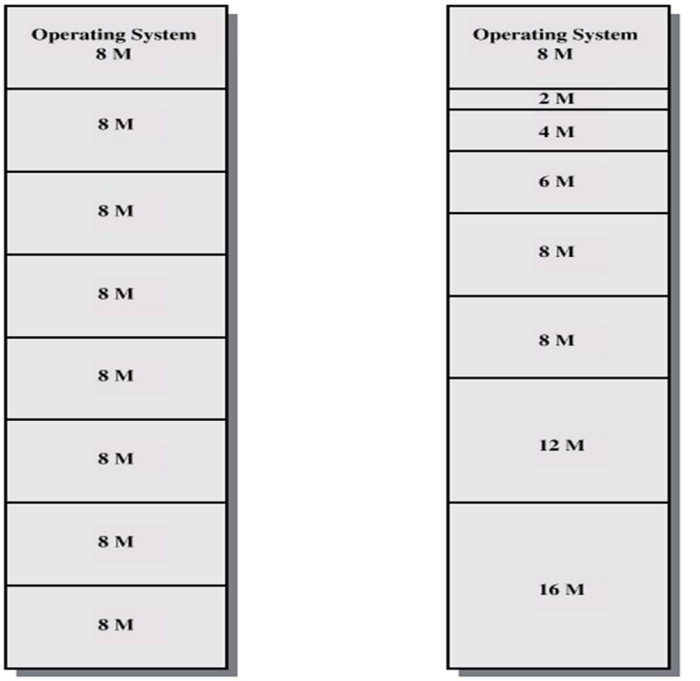
- Memory Layout for Resident Monitor:



- **Multi-program**
  - ✓ "User" part is sub-divided and shared among active processes
  - ✓ Schemes: Partitioning , Paging , Segmentation…etc.

Partitioning:

- Splitting memory into sections to allocate to processes (including Operating System)
- **Fixed-sized partitions**
  - ✓ May not be equal size
  - ✓ Process is fitted into smallest hole that will take it (best fit)
  - ✓ Some wasted memory
  - ✓ Leads to variable sized partitions



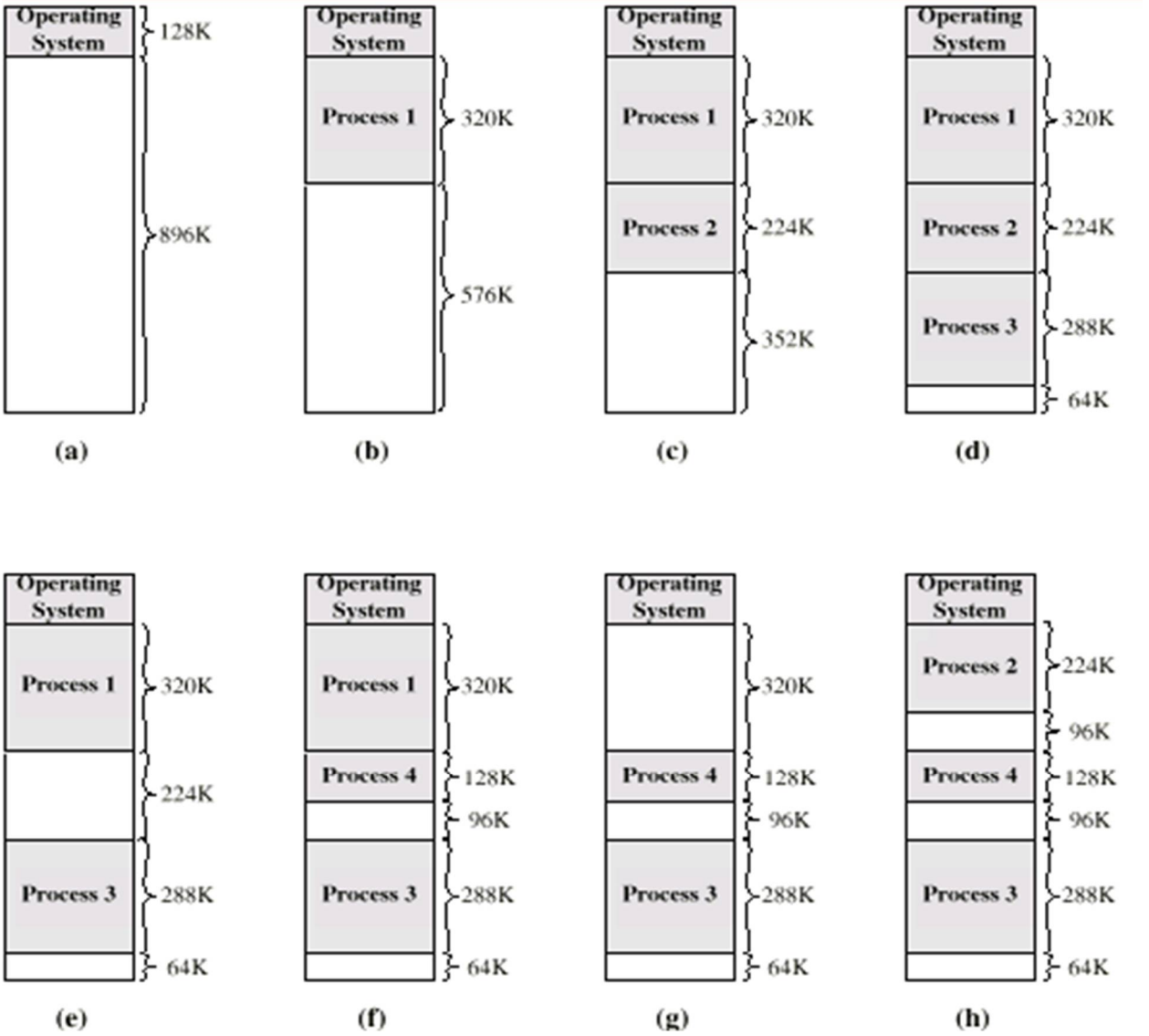
(a) Equal-size partitions (b) Unequal-size partitions

❖ **Variable Sized Partitions:**

- ✓ Allocate exactly the required memory to a process
- ✓ This leads to a hole at the end of memory, too small to use
  - ➢ Only one small hole - less waste
- ✓ When all processes are blocked, swap out a process and bring in another
- ✓ New process may be smaller than swapped out process
- ✓ Another hole
- ✓ Eventually have lots of holes (fragmentation)
- ✓ Solutions:
  - ➢ Coalesce - Join adjacent holes into one large hole
  - ➢ Compaction - From time to time go through memory and move all hole into one free block (c.f. disk de-fragmentation)

❖ **Dynamic Partitioning:**

- ✓ No guarantee that process will load into the same place in memory
- ✓ Instructions contain addresses
  - ➢ Locations of data
  - ➢ Addresses for instructions (branching)
- ✓ Logical address - relative to beginning of program
- ✓ Physical address - actual location in memory (this time)
- ✓ Automatic conversion using base address



Swapping:

- **Problem:**
  - ✓ We have a long-term queue of process requests, typically stored on disk. These are brought in, one at a time, as space becomes available. As processes are completed, they are moved out of main memory.
  - ✓ Now the situation will arise that none of the processes in memory are in the ready state (e.g., all are waiting on an I/O operation).
  - ✓ CPU remains idle.

- **Solution:**
  - ✓ If none of the processes in memory are ready (i.e. all I/O blocked)
    - ➢ Swap out a blocked process to intermediate queue
    - ➢ Swap in a ready process or a new process
  - ✓ Execution then continues with the newly arrived process.
  - ✓ But swapping is an I/O process...

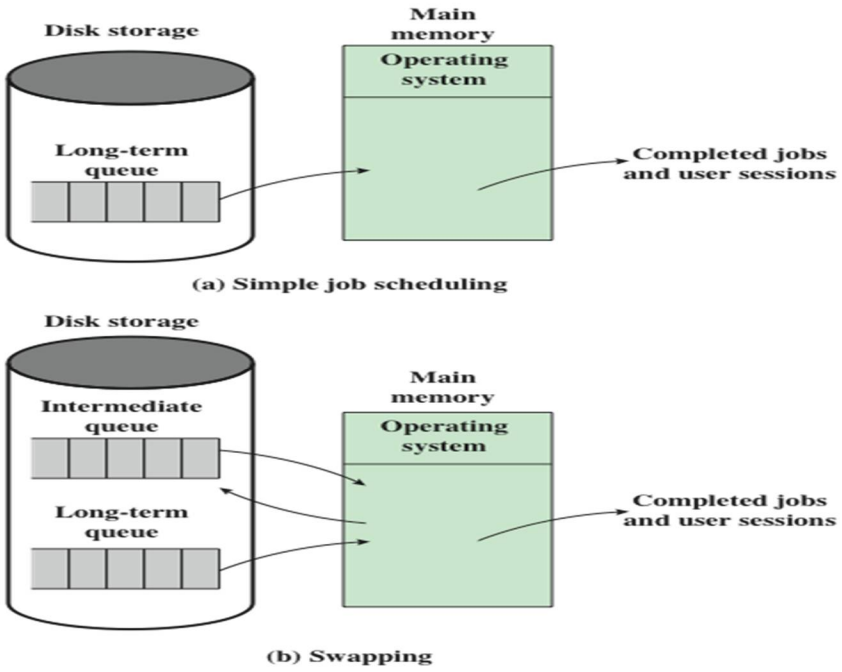


Figure 8.12 The Use of Swapping

Paging:

- Split memory into equal sized, small chunks -page frames
- Split programs (processes) into equal sized small chunks - pages
- Allocate the required number page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames
- Use page table to keep track
- Logical and Physical Addresses – Paging: