Chapter 5
Operating Systems

Outline

- Basic Operating System Functions
- Machine-Dependent Operating System Features
- Machine-Independent Operating System Features
- Operating System Design Options
- Implementation Examples

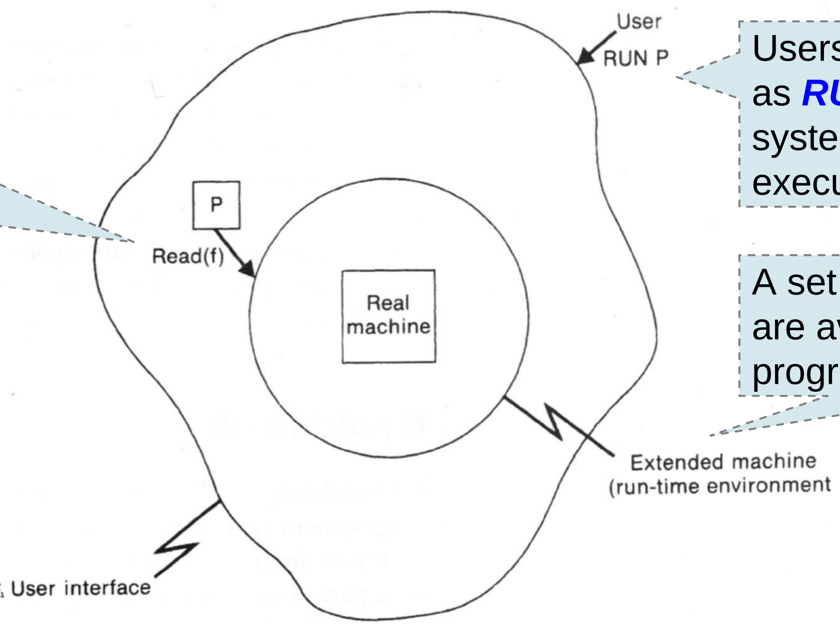
Basic Operating System Functions

Basic Concept of an Operating System

- The main purpose of operating systems:
  - To make the computer easier to use.
  - To provide an interface that is more user-friendly.
  - To provide a set of services that can aid in the performance of many common tasks.
  - To manage the resources of the computer to meet overall system goals (e.g., efficiency).

OS provides a service routine that could be invoked with a command such as ***read(f)*** to read data ***from file f***.

Govern the interactions with programmers, operators, etc. This interface may provide a ***control language*** to allow users to enter commands to invoke system functions.



Users enter a command such as ***RUN P*** to invoke the system loader to load and execute a program.

A set of service routines that are available for use during program execution.

Types of Operating Systems

- There are several ways to classify operating systems and some of the classifications overlap or fall into the same category.
  - The kind of user interface provided:
    - ***Single-job system***
      - Run one user job at a time because of the limited memory size and lack of data channels.
      - Commonly found on microcomputers.
    - ***Multiprogramming system***
      - Permit several user jobs to be executed concurrently.
      - Switch the CPU among the various user jobs by the operating system.
      - Provide a suitable run-time environment so the jobs do not interfere with each other.
    - ***Multiprocessor system***
      - Contain more than one CPU that shares a common memory.
      - Users can view the system as if it were a powerful single processor.
    - ***Network operating system***
      - Provide an interface to allow communication via the network.
      - Allow to login to remote machines and to copy files from one to another.

Types of Operating Systems (Cont.)

  - ***Distributed operating system***
    - Allow a complex type of network organization.
    - Let users view the entire network as a single system.
      - » Users are unaware of which machine on the network is actually running a program or storing data.
- The type of accesses provided to a user:
  - ***Batch processing system***
    - A job is described by a sequence of control statements stored in a machine-readable form.
    - The operating system can read and execute a series of such jobs without human intervention.
    - Goal: Make the most efficient use of the computer.
  - ***Time-sharing system***
    - Provide interactive access to a number of users with less efficient machine utilization.
    - Execute commands with a reasonably short response time.
    - Goal: Provide good response time to the interactive users.
  - ***Real-time system***
    - Response quickly to external signals such as those generated by data sensors.
    - Usually used on computers that monitor and control time-critical process such as nuclear reactor operation or spacecraft flight.
    - Goal: Provide a guaranteed response time to time-critical external events.

User Interface

- The user interface is designed to serve the needs of the various groups of people who must deal with the computer.
- The user interface of an operating system should be designed to match the needs of all the various types of users.
  - ***In simple operating systems***:
    - The interface is generally designed to be easy to use.
      - It may include a simple *command language*, a *menu* from which the user selects options, or a *graphical representation* of programs and data.
  - ***In complex systems:***
    - There may be a number of different user-interface languages.
    - There may be a more complex and more powerful command language.
    - There is usually a special language that is used to communicate with the operators of the computer.
    - Logs of system activity that can be used for performance analysis and error recovery are maintained.

Run-Time Environment

- The run-time environment
  - Contain ***a set of service routines*** that are available for use during program execution.
  - Provide facilities for *managing the resources of the computing system* and *assigning these resources to user programs*.
- Consider the I/O function under a single-job operations system on a SIC computer.
  - To perform a read operation without operating systems, the program needs to test the I/O errors and provide error-recovery routines.
  - Without the support of an operating system, the user program can simply invoke a service routine and specify the device to be used. The operating system would take care of all details (e.g., status testing and counting of bytes transferred).

Run-Time Environment (Cont.)

- A ***service routine*** can be thought of as providing an ***extension*** to the underlying machine.
  - Service routines can be thought of defining an ***extended machine*** for use by programs during execution.
    - The extended machine is easier to use than the real machine.
    - Programs deal with the functions and capabilities provided by this extended machine.
    - I/O operations on the extended machine may appear to be less error-prone than on the real machine, because the operating system takes care of error detection and recovery.
- The run-time environment contains routines that manage the resources of the computer, allocating them to user jobs.
  - The user jobs do not need to be concerned with resource management.
  - A ***virtual machine*** is provided to let user jobs run as if they own the machine, even though the underlying real machine is being shared.

Run-Time Environment (Cont.)

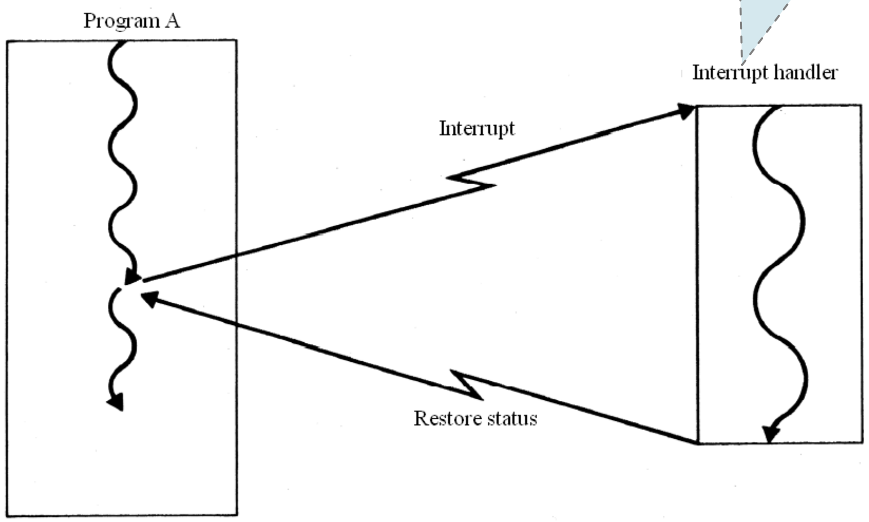
- On advanced systems, users generally request operating system functions by some special hardware instruction such as ***supervisor call (SVC)***.
  - Execution of an ***SVC instruction*** generates an ***interrupt*** that transfers control to an operating system service routine.
  - A ***code supplied by the SVC instruction*** specifies *the type of request*.
- The generation of an interrupt causes the CPU to switch from ***user mode*** to ***supervisor mode***.
  - ***In supervisor mode:***
    - All machine instructions and features can be used. (including ***privileged instructions***)
    - Most parts of the operating system are designed to run in this mode.
  - ***In user mode:***
    - Some instructions are not available. E.g., *I/O functions*, *set memory protection flags*, or *switch the CPU from one mode to another*.
- Restricting the use of such ***privileged instructions*** forces programs to make use of the services provided by the run-time environment.
  - User programs must deal with the extended machine interface, rather than utilizing the underlying hardware functions directly.
  - *Privileged instructions* and *user/supervisor modes* are a practical necessity for a system that supports more than one user at a time.

Machine-Dependent Operating System Features

Interrupt Processing

- An ***interrupt*** is a signal that causes a computer to alter its normal flow of instruction execution.
  - Such signals can be generated by different conditions, e.g., *the completion of an I/O operation*, *the expiration of a preset time interval*, or *an attempt to divide by zero*.
- Interrupt processing:
  - The *interrupt* automatically transfers controls to an ***interrupt-processing routine*** (called ***interrupt handler***) that is usually part of the operating system.
  - After completion of the interrupt processing, control can be returned to program A at the point at which its execution was interrupted.
  - E.g., Program A is interrupted by the I/O completion of other programs.
    - The interrupts is ***asynchronous*** with respect to program A.
    - A is unaffected by the interrupt, expect for *timing*.

Interrupt-processing routine



Interrupt Processing (Cont.)

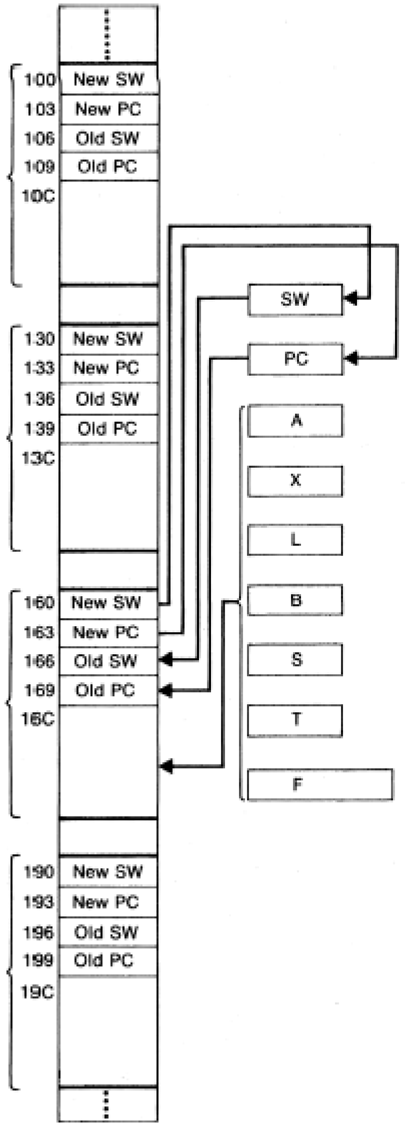
SIC/XE interrupt types

Class	Interrupt type
I	SVC
II	Program
III	Timer
IV	I/O

- SIC/XE interrupt classes
  - ***SVC interrupt***
    - The SVC interrupt is generated when a supervisor call (SVC) instruction is executed by the CPU.
    - The SVC instruction is used by programs to request operating system functions.
  - ***Program interrupt***
    - This program interrupt is generated by some condition that occurs during program execution.
      - E.g., *an attempt to divide by zero*, *an attempt to execute an illegal machine instruction*.
  - ***Timer interrupt***
    - The timer interrupt is generated by an interval timer within the CPU.
    - The timer contains a register that can be set to an initial positive value by the privileged instruction ***STI***.
      - The value in this register is automatically decremented by 1 for each millisecond of CPU time.
      - When the value reaches zero, a timer interrupt occurs.
      - The ***interval timer*** is used by the operating system to govern how long a user program can remain in control of the machine.
  - ***I/O interrupt***
    - The I/O interrupt is generated by an I/O channel or device.
    - Most such interrupts are caused by the normal completion of some I/O operation.

Interrupt Processing (Cont.)

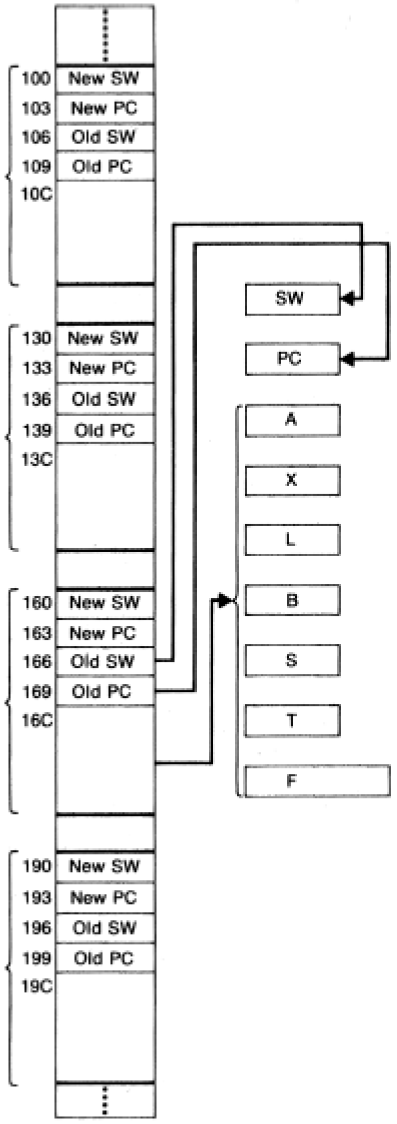
- When an interrupt occurs, the status of the CPU is saved, and control is transferred to an interrupt-processing routine.
- On a SIC/XE machine, there is a ***fixed interrupt work area*** corresponding to each class of interrupt.
- E.g., The area assigned to the timer interrupt begins at memory address 160. *The following storing and loading of registers (called* ***context switch****) are done automatically by the hardware of the machine.*
  - 1. When a timer interrupt occurs, the *contents of all registers* are stored in this work area.
  - 2. Then the first two words of the area are loaded to *the status word (SW)* and the *program counter (PC)*.



(a) (timer interrupt)

Interrupt Processing (Cont.)

- The loading of PC with a new value automatically causes a transfer of control.
  - The next instruction to be executed is taken from the address given by the new value of PC.
  - This prestored address is the starting address of the interrupt-handling routine for a timer interrupt.
  - The loading of SW also causes certain changes.
- After the handling routine returns by executing a ***Load Processor Status (LPS) instruction*** to restore the contents of SW, PC, and other registers.
  - This transfers control to the instruction following the one that was being executed when the interrupt occurred.

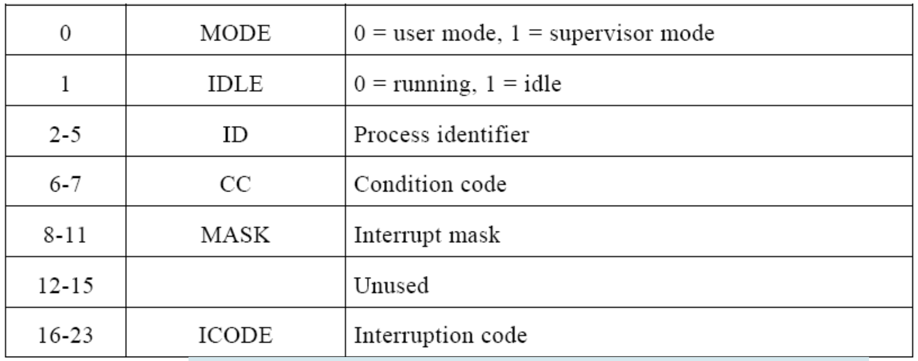


(b) LPS 166 (LPS 166)

Note: ***SW*** is similar to the ***program status word*** or ***processor status word*** in other computers.

SIC/XE Status Word

- ***MODE*** field:
  - Specify whether the CPU is in user mode or supervisor mode.
  - When an interrupt occurs, the new SW contents have MODE=1. This automatically switches the CPU to supervisor mode.
- ***ICODE*** field:
  - *Before **the old value of SW** is saved*, the ICODE field is automatically set to a value that indicates the cause of the interrupt.
    - ***For an SVC interrupt***, ICODE is set to *the value specified by the user in the SVC instruction*.
    - ***For a program interrupt***, ICODE indicates *the type of condition*, such as divided by zero.
    - ***For an I/O interrupt***, ICODE indicates gives *the number of the I/O channel* that generated the interrupt.
- ***CC*** field (condition code):
  - Saving SW automatically preserves the condition code value that was being used by the interrupted process.
- ***IDLE*** field:
  - Specify whether the CPU is executing instructions or is idle.
- ***ID*** field:
  - Identify the user program currently being executed.



0	MODE	0 = user mode, 1 = supervisor mode
1	IDLE	0 = running, 1 = idle
2-5	ID	Process identifier
6-7	CC	Condition code
8-11	MASK	Interrupt mask
12-15		Unused
16-23	ICODE	Interruption code

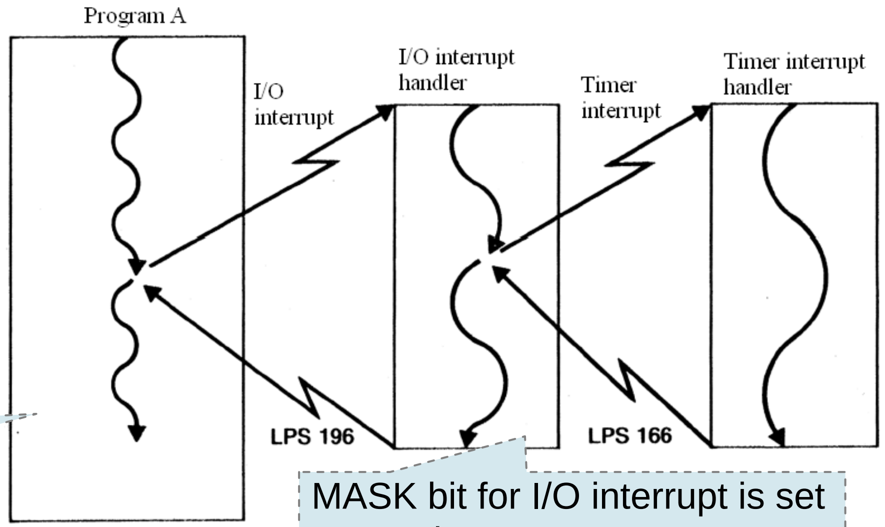
SIC/XE status word contents

SIC/XE Status Word (Cont.)

- ***Mask*** field:
  - Control whether interrupts are allowed.
    - Each bit in MASK is set to 0 to disable the corresponding interrupt. (Called *masked* or *inhibited* or *disabled*)
  - This control is necessary to prevent loss of the stored processor status information.
    - ***For example***:
      - 1. an I/O interrupt occurs. The values of SW, PC, and other registers are stored in the I/O interrupt work area, and the CPU begin to execute the I/O interrupt handler.
      - 2. If another I/O interrupt occurs before the processing of the fist one had been completed, another context switch would take place.
      - 3. This time, the values that were saved by the original interrupt would be destroyed, so it would be impossible to return control to the user program that was executing at the time of the first interrupt.
  - Interrupts that are masked are not lost because the hardware saves the signal that would have caused the interrupt. The delayed interrupt is said to be ***pending***.
  - The masking of interrupts on a SIC/XE machine is *under the control of the operating system*.
    - It depends on the value of MASK in the SW that is prestored in each interrupt work aea.

Interrupt Priority

- Each class of interrupts on a SIC/XE machine is assigned an ***interrupt priority***.
  - SVC interrupt, program interrupts, and so on.
  - For example:
    - Initially, the MASK field in the status word for each interrupt class is set.
    - The status word that is loaded in response to a ***program interrupt*** would have
      - The MASK bits for *program*, *timer*, and *I/O* interrupts set to 0.
      - The MASK bit for SVC interrupts set to 1.
  - Such an interrupt priority would cause a ***nested interrupt*** situation.



MASK bits set to 1.

MASK bit for I/O interrupt is set to 0. Others are set to 1.

Process Scheduling

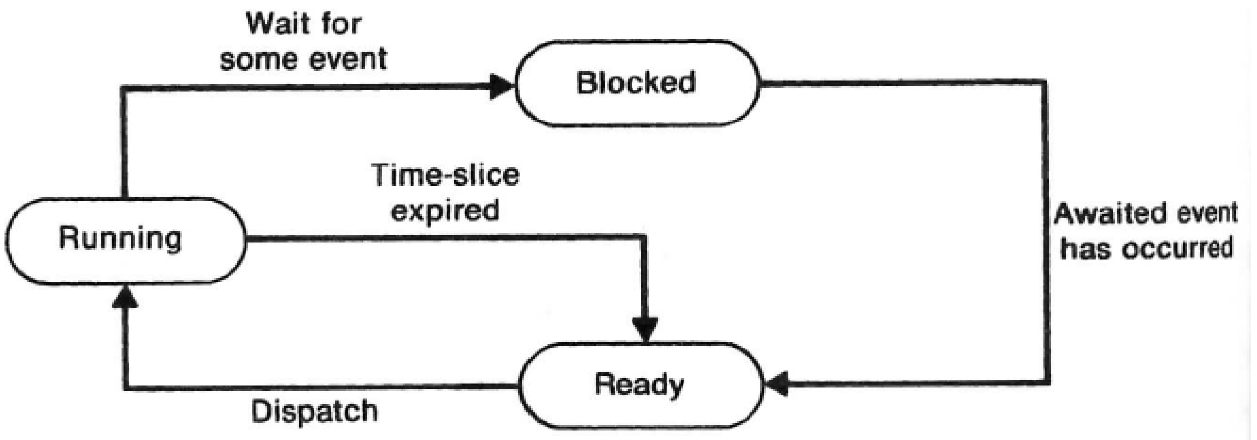
- A ***process*** (sometimes called a ***task***) is defined as ***a program in execution***.
  - The CPU is assigned to processes by the operating system in order to perform computing work.
  - In a single-job process scheduling, only one user process at a time.
  - In a multiprogramming system, many independent processes competing for control of the CPU.
- Process scheduling is the management of the CPU by switching control among the various competing processes according to some scheduling policy.
- In most cases, a ***process*** corresponds to a ***user job***. However,
  - Some systems allow one user job to create several different processes that are executed concurrently.
  - Some systems allow one program to be executed by several independent processes.

Process Scheduling (Cont.)

- A process is created when a user job begins execution, and this process is destroyed when the job terminates.
- During the period of its existence, the process can be considered to be in one of three states:
  - ***Running***:
    - A process is running when it is executing instructions using the CPU.
  - ***Blocked***:
    - A process is blocked if it must wait for some event to occur before it can continue execution.
    - E.g., a process might be blocked because it must wait for the completion of an I/O operation before proceeding.
  - ***Ready***:
    - Processes that are neither blocked nor running are said to be ***ready***.
    - Ready processes are candidates to be assigned the CPU when the currently running process gives up control.

Process Scheduling (Cont.)

- At any particular time, there can be no more than one process in the running state.
- When the operating system transfers control to a user process, it sets the ***interval timer*** to specify a ***time-slice***.
  - A time-slice is a maximum amount of CPU time the process is allowed to user before giving up control.
  - The selection of a process is called ***dispatching***. The part of the opearting



Process Scheduling (Cont.)

- Each time a process leaves the running state, its *current status must be saved*.
- This status must be restored the next time the process is dispatched so that the switching will have no effect on the results of the computation being performed.
- The status information for each process is saved by the operating system in a ***process status block (PSB)*** for that process.
  - A PSB is created when a process first begins execution of the process state. It contains
    - An indication of the *process state* (running, ready, or blocked).
    - An area that is used to save *all machine registers* (including SW and PC).
    - A variety of *other information* (e.g., variables).

Process Scheduling - Dispatching

- It is about to switch from one process to another. The status information of the previously running process needs to be saved.
  - If that process lost control of the CPU because its *time-slice* expired, the status information can be found in the *timer-interrupt work area*.
  - If that process gives up control (via an SVC request) because it needed to wait for the occurrence of some event, the status information can be found in the SVC-interrupt work area.
  - It is possible that no process was previously running process.
- After saving the status of the previously running process, the dispatcher selects a new process to receive control.
  - The dispatcher sets the interval timer to specify the time-slice.
  - Then the dispatcher switches control by using the *LPS instruction* to load the status information saved in the PSB for that process.
  - If there is no process in the ready state, set the CPU to idle.

Process Scheduling – Dispatching (Cont.)

```
procedure DISPATCH

    update PSB of previously Running process (if any)
    select next Ready process to receive control
    if a Ready process was found then
       begin
          mark selected process as Running
          set interval timer for desired time-slice using STI
          switch control to selected process using LPS
       end
    else
       place CPU in idle status using LPS
```

Process Scheduling – Dispatching (Cont.)

- There are several different methods for selecting the next process to be dispatched:
  – ***Round robin***:
    - The dispatcher cycles through the PSBs, selecting the next process that is in the ready state.
    - Each process dispatched is given the same length time-slice as all other processes.
  – ***Priority scheme***:
    - 1. Each user job has its predefined priority.
      - The goal of such a system is to provide the desired level of service for each class of job.
    - 2. the priorities are assigned by the operating system.
      - The assignment of priorities is made in an effort to improve the overall system performance.
    - 3. Assign different time-slices to different processes in conjunction with the priority system.

Process Scheduling – Wait & Signal

- When a running process reaches a point at which it must wait for some event to occur, the process informs the operating system by making a **WAIT (SVC 0)** service request.
- The occurrence of an event on which other processes may be waiting is communicated to the operating system by a **SIGNAL (SVC 0)** request.
- The event to be awaited or signaled is specified by giving the address of an ***event status block (ESB)*** that is associated with the event.
  - The ESB contains a ***flag bit ESBFLAG*** that records whether or not the associated event has occurred.
  - The ESB contains ***a pointer to ESBQUEUE***, a lost of all processes currently waiting for the event.

Process Scheduling – Wait & Signal (Cont.)

Requested by a running process

```
procedure WAIT(ESB)

    if ESBFLAG = 1 then {event has already occurred}
        return control to requesting process using LPS
    else
        begin
            mark requesting process as Blocked
            enter requesting process on ESBQUEUE
            DISPATCH
        end
```

(a)

Requested by a process that detects that some event corresponding to ESB has occurred.

```
procedure SIGNAL(ESB)

    ESBFLAG := 1  {indicate that event has occurred}
    for each process on ESBQUEUE do
        begin
            mark process as Ready
            remove process from ESBQUEUE
        end
    return control to requesting process using LPS
```

(b)

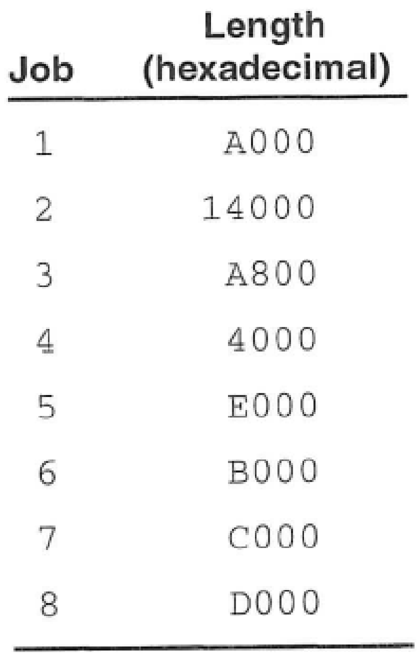
If the dispatching method being used is based on priorities, the SIGNAL algorithm would invoke the dispatcher to transfer control to the highest-priority process that is currently ready.
This is called ***preemptive process scheduling***: It permits a process that becomes ready to seize control from a lower-priority process that is currently running.

Management of Real Memory

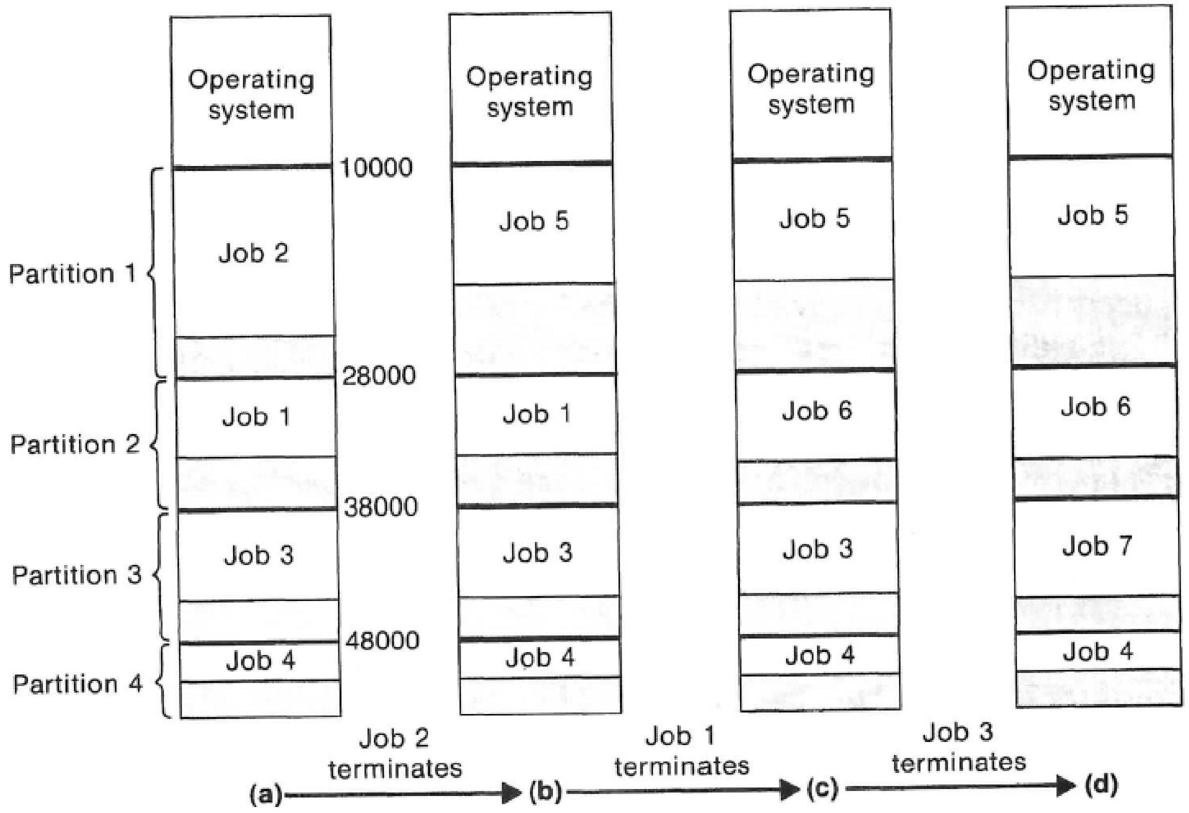
- Many systems divide memory into ***partitions***, with each process being assigned to a different partition.
  - ***Fixed partition:***
    - Partitions are predefined in size and position.
  - ***Variable partition:***
    - Partitions are allocated dynamically according to the requirements of the jobs being executed.

Fixed Partitions

- Load each incoming job into the smallest free partition in which it will fit.



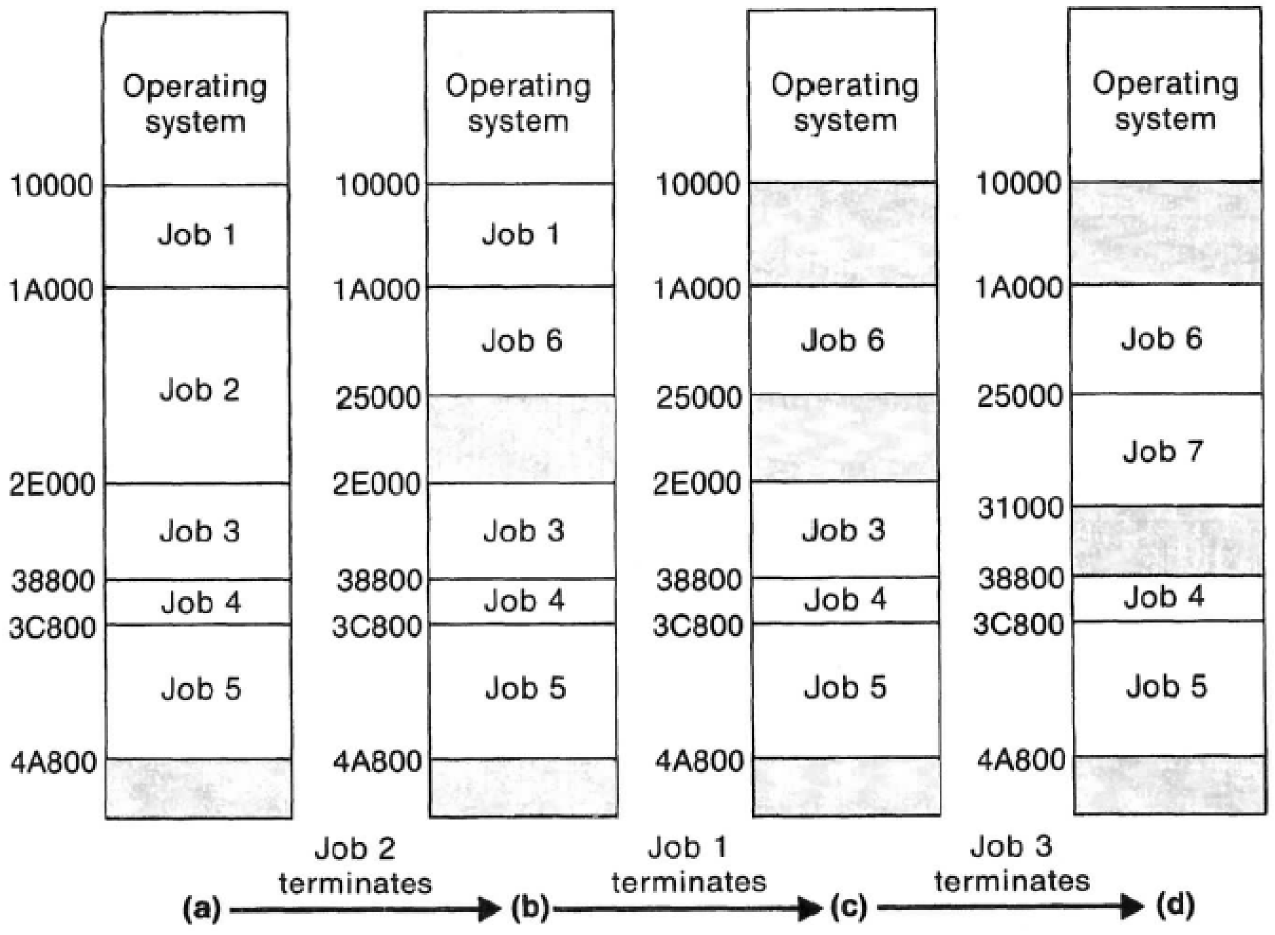
Job	Length (hexadecimal)
1	A000
2	14000
3	A800
4	4000
5	E000
6	B000
7	C000
8	D000



Variable Partitions

Problem: fragmentation

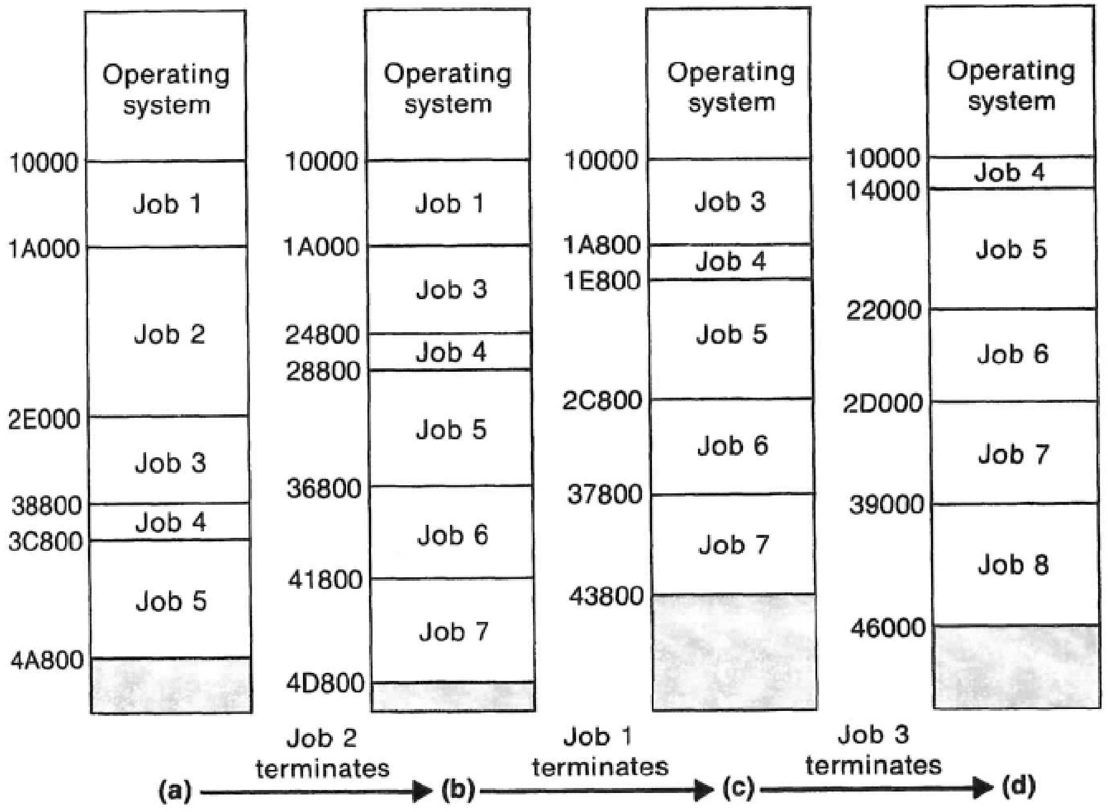
Job	Length (hexadecimal)
1	A000
2	14000
3	A800
4	4000
5	E000
6	B000
7	C000
8	D000



Allocation strategy:
- First fit
- Best fit
- Next fit

Relocatable Partition

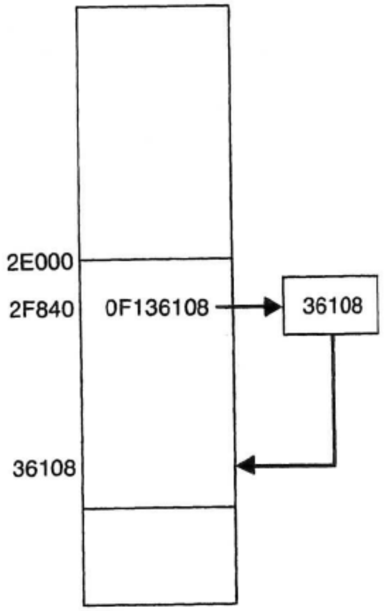
- Remaining partitions are moved as far as possible.



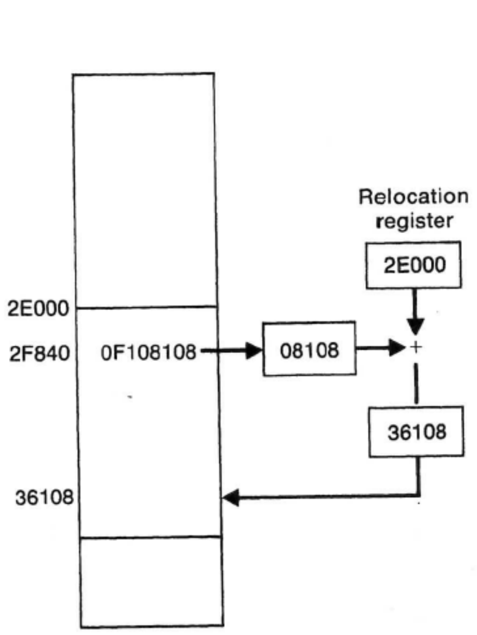
Relocation Register

Loc	Source statement			Object code
0000	P3	START	0	
		.		
1840		+STA	BUFF2	0F108108
		.		
8108	BUFF2	. . .		
		.		
		END		

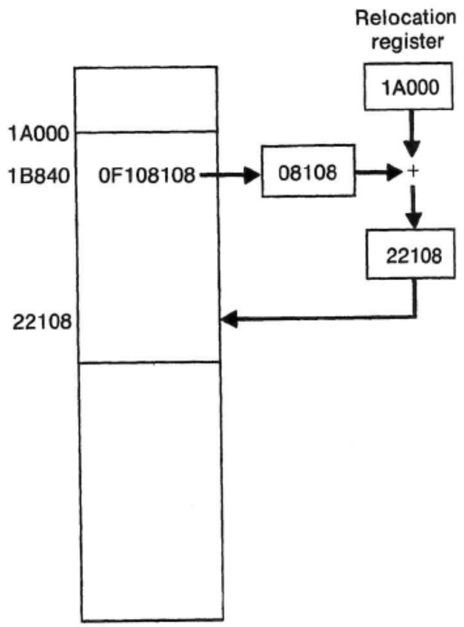
(a)



(b)



(c)



(d)