



**WOLLO UNIVERSITY  
COLLEGE OF INFORMATICS  
DEPARTMENT OF COMPUTER SCIENCE REGULAR**

**FUNDAMENTAL OF OPERATING SYSTEM LECTURE NOTE (CoSc 2042)  
FOR SECOND YEAR COMPUTER SCIENCE REGULAR STUDENTS**

**BY: Kibebu M.**

**2012 E.C**

# Chapter 1 :- System Software

## ➤ Unit Structure

- ❖ Objectives
- ❖ Introduction
- ❖ Definition of operating system
- ❖ Functions of Operating System
- ❖ Operating System as User Interface
- ❖ Assembler ,Compiler , Loader
- ❖ Summary
- ❖ Model questions

# Objectives

- After going through this unit, you will be able to:
  - Describe Basic Organization of Computer Systems
  - Define Operating system, functions, history and Evolution
  - Define assembler, linker, loader, compiler

# Introduction

- **An operating system** act as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
- **An operating system** is a software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

# Definition of Operating Systems

- **An Operating system** is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- A more common definition is that the **operating system** is the one program running at all times on the computer (usually called the kernel), with all else being applications programs.
- **An Operating system** is concerned with the allocation of resources and services, such as memory, processors, devices and information.
- **The Operating System** correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

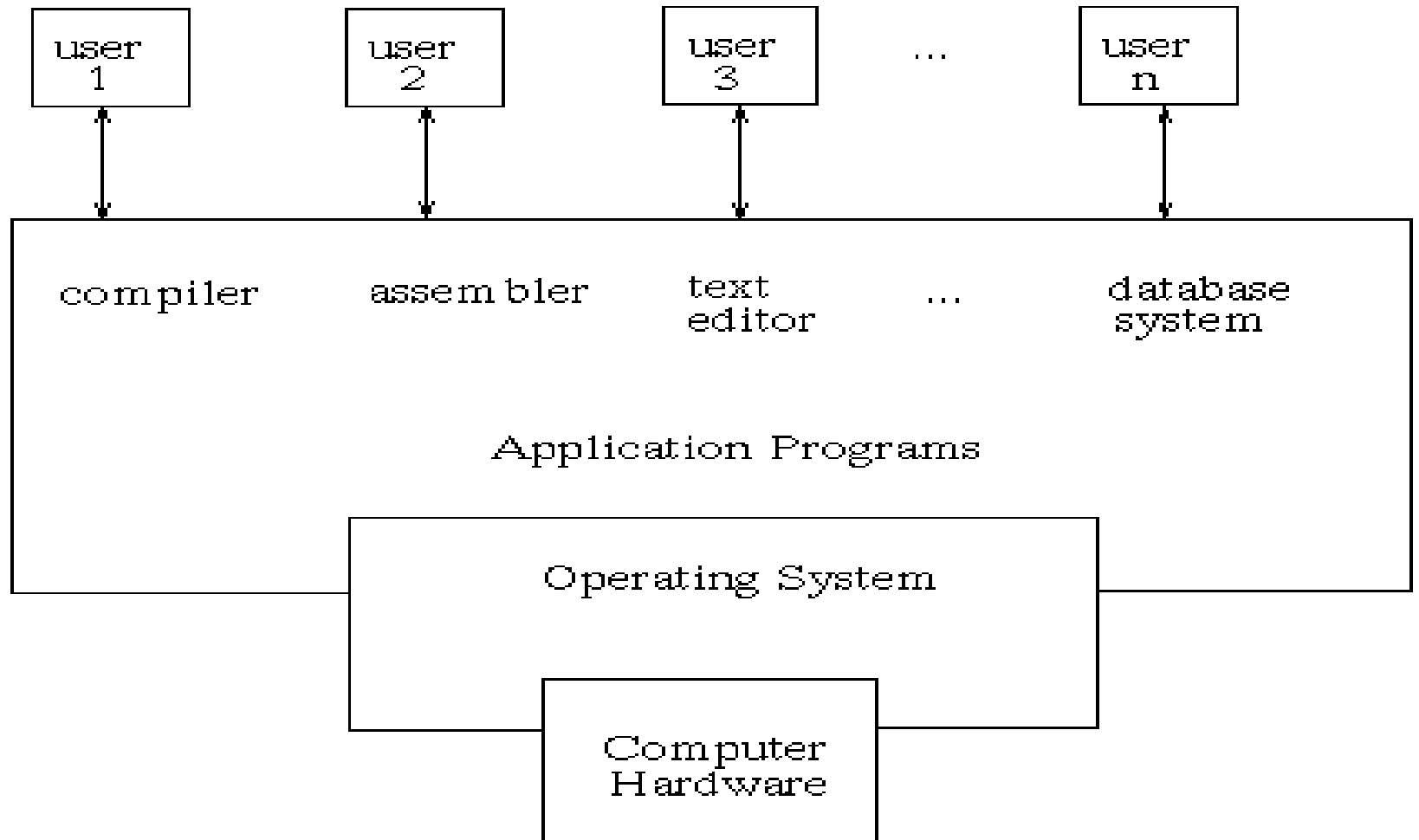
# Functions of Operating System

- Operating system performs three functions:
  - ❖ **Convenience:** An OS makes a computer more convenient to use.
  - ❖ **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
  - ❖ **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service

# Operating System as User Interface

- Every general purpose computer consists of the hardware, operating system, system programs, application programs.
- The hardware consists of memory, CPU, ALU, I/O devices, peripheral device and storage device.
- System program consists of compilers, loaders, editors, OS
- The application program consists of business program, database program.
- The fig. 1.1 shows the conceptual view of a computer system

Cont...



**Fig 1.1 Abstract view of the components a computer system**



## Cont...

1. **Hardware** - provides basic computing resources(CPU, memory ,I/O devices).
2. **Operating system** - controls and coordinates the use of the hardware among the various application programs for the various users.
3. **Applications programs** - define the ways in which the system resources are used to solve the computing problems of the users (database systems, video games, business pro grams).
4. **Users** (people, machines, other computers).

## Cont...

- **Operating system** is a set of special programs that run on a computer system that allows it to work properly.
- Operating system performs the following basic tasks :-
  - ✓ Recognizing input from the keyboard
  - ✓ Keeping track of files and directories on the disk.
  - ✓ It manage system resources.
  - ✓ Sending output to the display screen.
  - ✓ Controlling the peripheral devices.

# Goal of operating system

- OS is designed to serve the following purposes :
  - 1) . **Resource Management:** Disk, CPU cycles, etc. must be managed efficiently to maximize overall system performance. It controls the allocation and use of the computing system's resources among the various user and tasks.
  - 2) . **Resource Abstraction:** Software interface to simplify use of hardware resources . It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs
  - 3) **Virtualization:** Supports resource sharing – gives each process the appearance of an unshared resource

Cont...

- The operating system must support the following tasks. The tasks are :
  - Provides the facilities to create, modification of program and data files using and editor.
  - Access to the compiler for translating the user program from high level language to machine language.
  - Provide a loader program to move the compiled program code to the computer's memory for execution.
  - Provide routines that handle the details of I/O programming.

# System Calls // Programs

- System calls provide the interface between a process and the operating system.
- Some of system calls are assembler, compiler, interpreter and loader.

# Assembler

- Input to an assembler is an assembly language program.
- Output is an object program plus information that enables the loader to prepare the object program for execution.
- At one time, the computer programmer had at his disposal a basic machine that interpreted, through hardware, certain fundamental instructions.
- He would program this computer by writing a series of ones and zeros(machine language), place them into the memory of the machine.

# Compiler

- The high level languages – examples are FORTRAN, COBOL, ALGOL and PL/I – are processed by compilers and interpreters.
- A compiler is a program that accepts a source program in a high-level language and produces a corresponding object program.
- An interpreter is a program that appears to execute a source program as if it was machine language. The same name (FORTRAN, COBOL etc) is often used to designate both a compiler and its associated language.

# Loader

- A loader is a routine that loads an object program and prepares it for execution. There are various loading schemes: absolute, relocating and direct-linking.
- **Loader is a program that places programs into memory and prepares them for execution.**
- In a simple loading scheme, the assembler outputs the machine language translation of a program on a secondary device and a loader is placed in core.
- The loader places into memory the machine language version of the user's program and transfers control to it. Since the loader program is much smaller than the assembler, those makes more core available to user's program.



# Summary

- **An Operating system** is concerned with the allocation of resources and services, such as memory, processors, devices and information. The Operating System correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.
- **Assembler**

Input to an assembler is an assembly language program. Output is an object program plus information that enables the loader to prepare the object program for execution.
- **Loader**

A loader is a routine that loads an object program and prepares it for execution. There are various loading schemes: absolute, relocating and direct-linking. In general, the loader must load, relocate, and link the object program
- **Compilers**

A compilers is a program that accepts a source program in a high- level language and produces a corresponding object program.



# Model Question

Q. 1 Define Operating System?

Q. 2 Explain various function of operating system?

Q. 3 Explain I/O system Management?

Q. 4 Define & explain Assembler, Loader, Compiler?

# Chapter 2

## Fundamental of Operating System

### ➤ Unit Structure

- ❖ Objectives
- ❖ Introduction
- ❖ Operating System Services
- ❖ Operating System Components
- ❖ Batch System
- ❖ Time Sharing System
- ❖ Multiprogramming
- ❖ Spooling
- ❖ Properties of Operating System
- ❖ Summary
- ❖ Model Question

# Objectives

- After going through this unit, you will be able to:
  - To describe the services an operating system provides to users, processes, and other systems
  - Describe operating system components.
  - Define Batch, timesharing and multiprogramming OS

# Introduction

- **An operating system** provides the environment within which programs are executed. Internally, operating systems vary greatly in their makeup, since they are organized along many different lines.
- The design of a new operating system is a major task. It is important that the goals of the system be well defined before the design begins.
- We can view an operating system from several vantage points. One view focuses on the services that the system provides, another, on the interface that it makes available to users and programmers; a third, on its components and their interconnections.

# Operating System Services

- **An operating system** provides services to programs and to the users of those programs. It provided by one environment for the execution of programs. The services provided by one operating system is difficult than other operating system.
- Operating system makes the programming task easier.
- The common service provided by the operating system is listed below.
  1. Program execution
  2. I/O operation
  3. File system manipulation
  4. Communications
  5. Error detection

## Cont...

1. **Program execution:** Operating system loads a program into memory and executes the program. The program must be able to end its execution, either normally or abnormally.
2. **I/O Operation :** I/O means any file or any specific I/O device. Program may require any I/O device while running. So operating system must provide the required I/O.
3. **File system manipulation :** Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file.
4. **Communication :** Data transfer between two processes is required for some time. The both processes are on the one computer or on different computer but connected through computer network. Communication may be implemented by two methods:
  - a. Shared memory
  - b. Message passing.
5. **Error detection :** error may occur in CPU, in I/O devices or in the memory hardware. The operating system constantly needs to be aware of possible errors. It should take the appropriate action to ensure correct and consistent computing. Operating system with multiple users provides following services:
  1. Resource Allocation
  2. Accounting
  3. Protection



## Cont...

### 1) Resource Allocation :

- If there are more than one user or jobs running at the same time, then resources must be allocated to each of them. Operating system manages different types of resources require special allocation code, i.e. main memory, CPU cycles and file storage.
- There are some resources which require only general request and release code. For allocating CPU, CPU scheduling algorithms are used for better utilization of CPU. CPU scheduling algorithms are used for better utilization of CPU. CPU scheduling routines consider the speed of the CPU, number of available registers and other required factors.

### 2) Accounting :

- Logs of each user must be kept. It is also necessary to keep record of which user how much and what kinds of computer resources. This log is used for accounting purposes.
- The accounting data may be used for statistics or for the billing. It also used to improve system efficiency.

### 3) Protection :

- Protection involves ensuring that all access to system resources is controlled. Security starts with each user having to authenticate to the system, usually by means of a password. External I/O devices must be also protected from invalid access attempts.
- In protection, all the access to the resources is controlled. In multiprocess environment, it is possible that, one process to interface with the other, or with the operating system, so protection is required.



# Operating System Components

- Modern operating systems share the goal of supporting the system components. The system components are :
  - Process Management
  - Main-Memory Management
  - Secondary-Storage Management
  - I/O System Management
  - File Management
  - Protection System
  - Networking
  - Command-Interpreter System

Cont...

# Process Management

- **A process** is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The five major activities of an operating system in regard to process management are:
  - Creation and deletion of user and system processes.
  - Suspension and resumption of processes.
  - A mechanism for process synchronization.
  - A mechanism for process communication.
  - A mechanism for deadlock handling.
- Process management is usually performed by the kernel.

Cont...

## Main-Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- The major activities of an operating system in regard to memory-management are:
  - Keep track of which part of memory are currently being used and by whom.
  - Decide which processes are loaded into memory when memory space becomes available.
  - •Allocate and deallocate memory space as needed..

Cont...

# I/O System Management

- The I/O system consists of:
  - ✓ A buffer-caching system
  - ✓ A general device-driver interface
  - ✓ Drivers for specific hardware devices (**device drivers**)
- Device Drivers
  - ✓ Must have access to I/O hardware
  - ✓ Must be able to handle interrupts.
  - ✓ Communicate with other parts of the OS (File system, Networking etc).

Cont...

# File Management

- A file is a collection of related information.
  - Commonly, files represent programs (both source and object forms) and data.
- The operating system is responsible for the following activities in connection with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage. e.g. free space allocation.
  - File backup on stable (non-volatile) storage media.



Cont...

# Protection System

- Protection refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- Operating Systems commonly control access by using permissions. All system resources have an owner and a permission associated with them. Users may be combined into groups for the purpose of protection. e.g. in UNIX every file has an owner and a group.



# **Networking(Distributed system)**

- A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock
- The processors communicate with one another through communication lines called network

Cont...

# Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management (e.g. running a program)
  - I/O handling (e.g. set terminal type)
  - secondary-storage management (e.g. format a disk)
  - main-memory management (e.g. specify virtual memory parameters)
  - file-system access (e.g. print file)
  - protection (e.g. set permissions)
  - networking (e.g. set IP address)
- The program that reads and interprets control statements is called variously:
  - command-line interpreter
  - shell (in UNIX)



# Type of Operating system

## Batch System

- **Batch operating system** is one where programs and data are collected together in a batch before processing starts.
- A job is predefined sequence of commands, programs and data that are combined in to a single unit called job.
- Memory management in batch system is very simple. Memory is usually divided into two areas : Operating system and user program area.

## Cont...

- Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e first come first served fashion.
- When job completed execution, its memory is releases and the output for the job gets copied into an output **spool for later printing.**
- Batch system often provides simple forms of file management. Access to file is serial. Batch systems do not require any time critical device management.
- Batch systems are inconvenient for users because users can not interact with their jobs to fix problems. There may also be long turn around times. Example of this system id generating monthly bank statement.

Cont...

- **Advantages of Batch System**

- Move much of the work of the operator to the computer.
- Increased performance since it was possible for job to start as soon as the previous job finished.

- **Disadvantages of Batch System**

- Turn around time can be large from user standpoint.
- Difficult to debug program.
- A job could enter an infinite loop.
- A job could corrupt the monitor, thus affecting pending jobs.
- Due to lack of protection scheme, one batch job can affect pending jobs.

# Time Sharing System

- Multi-programmed batched systems provide an environment where the various system resources (for example, CPU, memory, peripheral devices) are utilized effectively.
- Time sharing, or multitasking, is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur so frequently that the users may interact with each program while it is running.



Cont...

- If users are to be able to access both data and code conveniently, an on-line file system must be available. A file is a collection of related information defined by its creator. Batch systems are appropriate for executing large jobs that need little interaction.
- Time-sharing systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory.
- A program that is loaded into memory and is executing is commonly referred to as a process.
- When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O. I/O may be interactive; that is, output is to a display for the user and input is from a user keyboard. Since interactive I/O typically runs at people speeds, it may take a long time to completed.

# Multiprogramming

- When two or more programs are in memory at the same time, sharing the processor is referred to the multiprogramming operating system. Multiprogramming assumes a single processor that is being shared. It increases CPU utilization by organizing jobs so that the CPU always has one to execute.
- The operating system keeps several jobs in memory at a time. This set of jobs is a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the job in the memory. Multiprogrammed system provide an environment in which the various system resources are utilized effectively, but they do not provide for user interaction with the computer system. Jobs entering into the system are kept into the memory. Operating system picks the job and begins to execute one of the job in the memory.



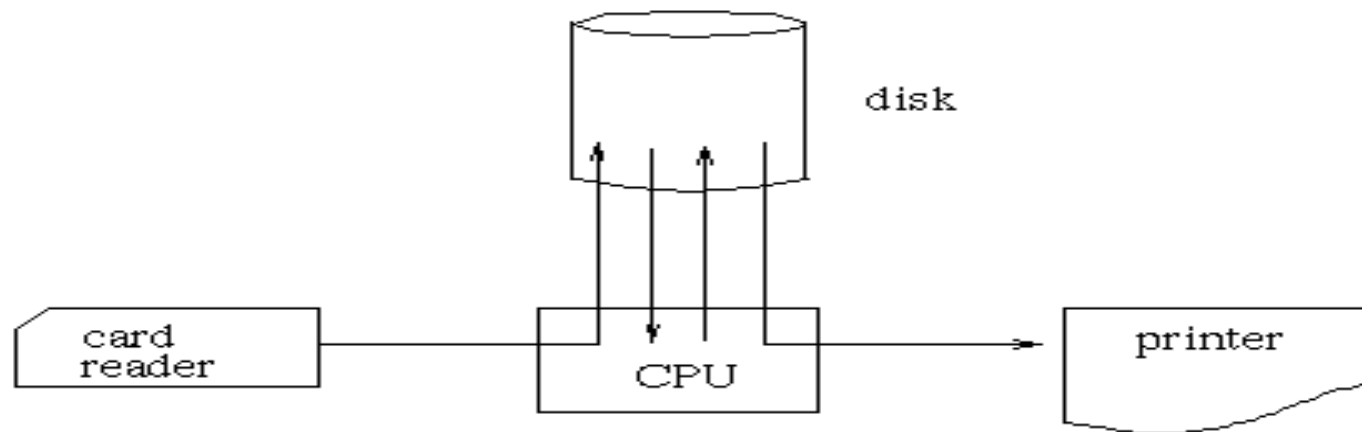
## Cont...

- Multiprogramming operating system monitors the state of all active programs and system resources. This ensures that the CPU is never idle unless there are no jobs.
- **Advantages**
  1. High CPU utilization.
  2. It appears that many programs are allotted CPU almost simultaneously.
- **Disadvantages**
  1. CPU scheduling is requires.
  2. To accommodate many jobs in memory, memory management is required.



# Spooling

- Acronym for simultaneous peripheral operations on line.
- Spooling refers to putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready.
- Spooling is useful because device access data that different rates. The buffer provides a waiting station where data can rest while the slower device catches up. Fig 2.3 shows the spooling.





## Cont...

- The most common spooling application is print spooling.
- In print spooling, documents are loaded into a buffer and then the printer pulls them off the buffer at its own rate.
- Spooling is also used for processing data at remote sites. The CPU sends the data via communications path to a remote printer.
- Spooling overlaps the I/O of one job with the computation of other jobs.
- One difficulty with simple batch systems is that the computer still needs to read the decks of cards before it can begin to execute the job. This means that the CPU is idle during these relatively slow operations. Spooling batch systems were the first and are the simplest of the multiprogramming systems.

### **Advantage of Spooling**

- 1. The spooling operation uses a disk as a very large buffer.
- 2. Spooling is however capable of overlapp

# Essential Properties of the Operating System

- **Batch** : Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering , off line operation, spooling and multiprogramming. A Batch system is good for executing large jobs that need little interaction, it can be submitted and picked up latter.
- **Time sharing** : Uses CPU s scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another i.e. the CPU is shared between a number of interactive users. Instead of having a job defined by spooled card images, each program reads its next control instructions from the terminal and output is normally printed immediately on the screen.
- **Interactive** : User is on line with computer system and interacts with it via an interface. It is typically composed of many short transactions where the result of the next transaction may be unpredictable. Response time needs to be short since the user submits and waits for the result.



## Cont...

- **Real time system** : Real time systems are usually dedicated, embedded systems. They typically read from and react to sensor data. The system must guarantee response to events within fixed periods of time to ensure correct performance.
- **Distributed** : Distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines

# Summary

- **An operating system** provides services to programs and to the users of those programs. It provided by one environment for the execution of programs. The services provided by one operating system is difficult than other operating system. Operating system makes the programming task easier.
- **Batch operating system** is one where programs and data are collected together in a batch before processing starts. In batch operating system memory is usually divided into two areas : Operating system and user program area.
- **Time sharing, or multitasking**, is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur so frequently that the users may interact with each program while it is running.
- When two or more programs are in memory at the same time, sharing the processor is referred to the **multiprogramming operating system**.
- **Spooling** is useful because device access data that different rates. The buffer provides a waiting station where data can rest while the slower device catches up.

# Model Question

Q. 1 Explain various operating system services?

Q. 2 Define Spooling? Describe Spooling process?

Q. 3 Differentiate Multitasking&&Multiprogramming?

# Chapter 3

## Process Management

### ▶ Unit Structure

- ❖ Objectives
- ❖ Concept of Process (Processes and Programs )
- ❖ Process State
- ❖ Suspended Processes
- ❖ Process Control Block
- ❖ Process Management (Scheduling Queues and Schedulers)
- ❖ Context Switching
- ❖ Operation on processes
- ❖ Co-operating Processes
- ❖ Summary
- ❖ Model Question

# Objectives

- After going through this unit, you will be able to:
  - To introduce the notion of a process – a program in execution, which forms the basis of all computation
  - To describe the various features of processes, including scheduling, creation and termination, and communication.



# Process concepts

- **A process** is an instance of a program in execution. It encompasses the static concept of program and the dynamic aspect of execution. As the process runs, its context (state) changes – register contents, memory contents, etc., are modified by execution.
- process is a sequential program in execution. A process defines the fundamental unit of computation for the computer. Components of the process are:
  - Object program
  - Data
  - Resources
  - Status of the process execution
- **Object program** i.e. code to be executed.
- **Data** is used to for executing the program.
- While executing the program, it may require some **resources**.
- **Last component**(Status of the process execution) is used for verifying the status of the process execution.
- A process can run to completion only when all requested resources have been allocated to the process. Two or more processes could be executing the same program each using their own **data** and **resources**.
- Process exists in a limited span of time.

# Processes and Programs

- **Process** is a dynamic entity, that is a program in execution. A process is a sequence of information executions. Process exists in a limited span of time. Two or more processes could be executing the same program, each using their own data and resources.
- **Program** is a static entity made up of program statement. Program contains the instructions. A program exists at single place in space and continues to exist. A program does not perform the action by itself.

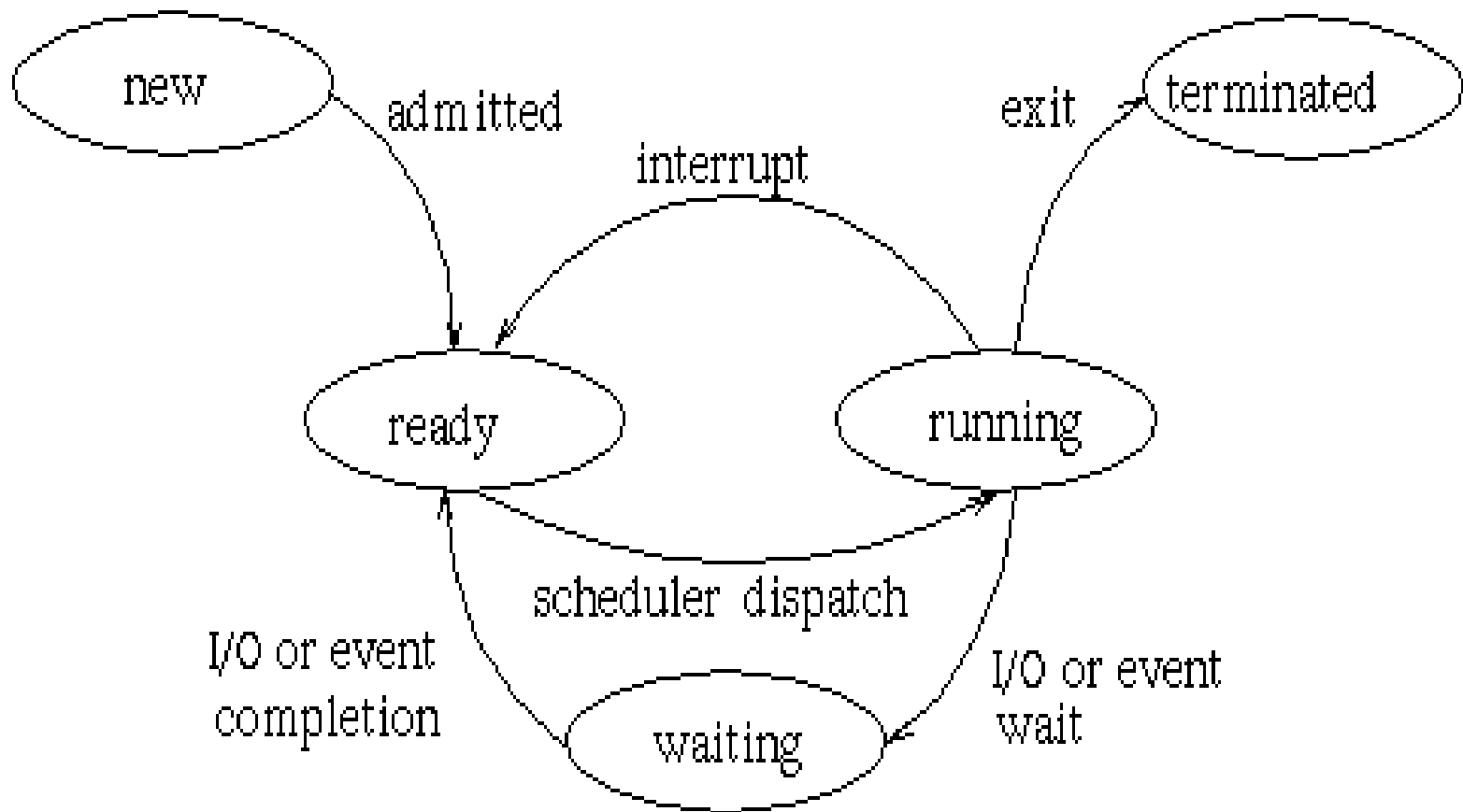
Cont...

## Process state

- When process executes, it changes state. Process state is defined as the current activity of the process.
- Process contains **five states**. Each process is in one of the states
  - New
  - Ready
  - Running
  - Waiting
  - Terminated (exit)
- **New**- a process that has just been created
- **Ready**- ready processes are waiting to have the processor. Allocating to them by the operating system so that they can run.
- **Running**- the process that is currently being executed. A running process processes all the resources needed for its execution, including the processor.
- **Waiting**- a process that can not execute until some event occurs. Such as the completion of an I/O operation. The running process may become suspended by invoking an I/O routine.

Cont...

- **Terminated**- a process that has been released from the pool of executable process by the OS.



*Fig. 3.1 shows the general form of the process state transition diagram.*

## Cont...

- When ever process changes state, the OS reacts by placing the PCB in the list that corresponds to its new state. Only one process can run on any processor at any instant and many processes may be in ready and waiting state.

### **Suspended processes:-** Characteristics of suspended processes

1. Suspended process is not immediately available for execution.
  2. The process may or may not be waiting on an event.
  3. For preventing the execution, process is suspend by OS, parent process, process itself and an agent.
  4. Process may not be removed from the suspended state until the agent orders the removal.
- **Swapping** is used to move all of a process from main memory to disk. When all the process by putting it in the suspended state and transferring it to disk.
  - Reasons for process suspension
    1. Swapping
    2. Timing
    3. Interactive user request
    4. Parent process request



## Cont...

- **Swapping** : OS needs to release required main memory to bring in a process that is ready to execute.
- **Timing** : Process may be suspended while waiting for the next time interval.
- **Interactive user request** : Process may be suspended for debugging purpose by user.
- **Parent process request** : To modify the suspended process or to coordinate the activity of various descendants.

# Process Control Block (PCB)

- **PCB** – is the data structure used by the OS. OS groups all information that needs about a particular process.
- **PCB-** Information associated with each process. which contains:
  - Process ID (name, number)
  - Process state
  - Priority, owner, etc...
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information



## Fig. 3.2 Process Control Block

Pointer	Process State
Process Number	
Program Counter	
CPU registers	
Memory Allocation	
Event Information	
List of open files	

## Cont...

- **Pointer** : Pointer points to another process control block. Pointer is used for maintaining the scheduling list.
- **Process State** : Process state may be new, ready, running, waiting and so on
- **Program Counter** : It indicates the address of the next instruction to be executed for this process.
- **Event information** : For a process in the blocked state this field contains information concerning the event for which the process is waiting.
- **CPU register** : It indicates general purpose register, stack pointers, index registers and accumulators etc. number of register and type of register totally depends upon the computer architecture.
- **Memory Management Information** : This information may include the value of base and limit register. This information is useful for deallocating the memory when the process terminates.
- **Accounting Information** : This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers
- **Process control block** also includes the information about CPU scheduling, I/O resource management, file management information, priority and so.
- **The PCB** simply serves as the repository for any information that may vary from process to process

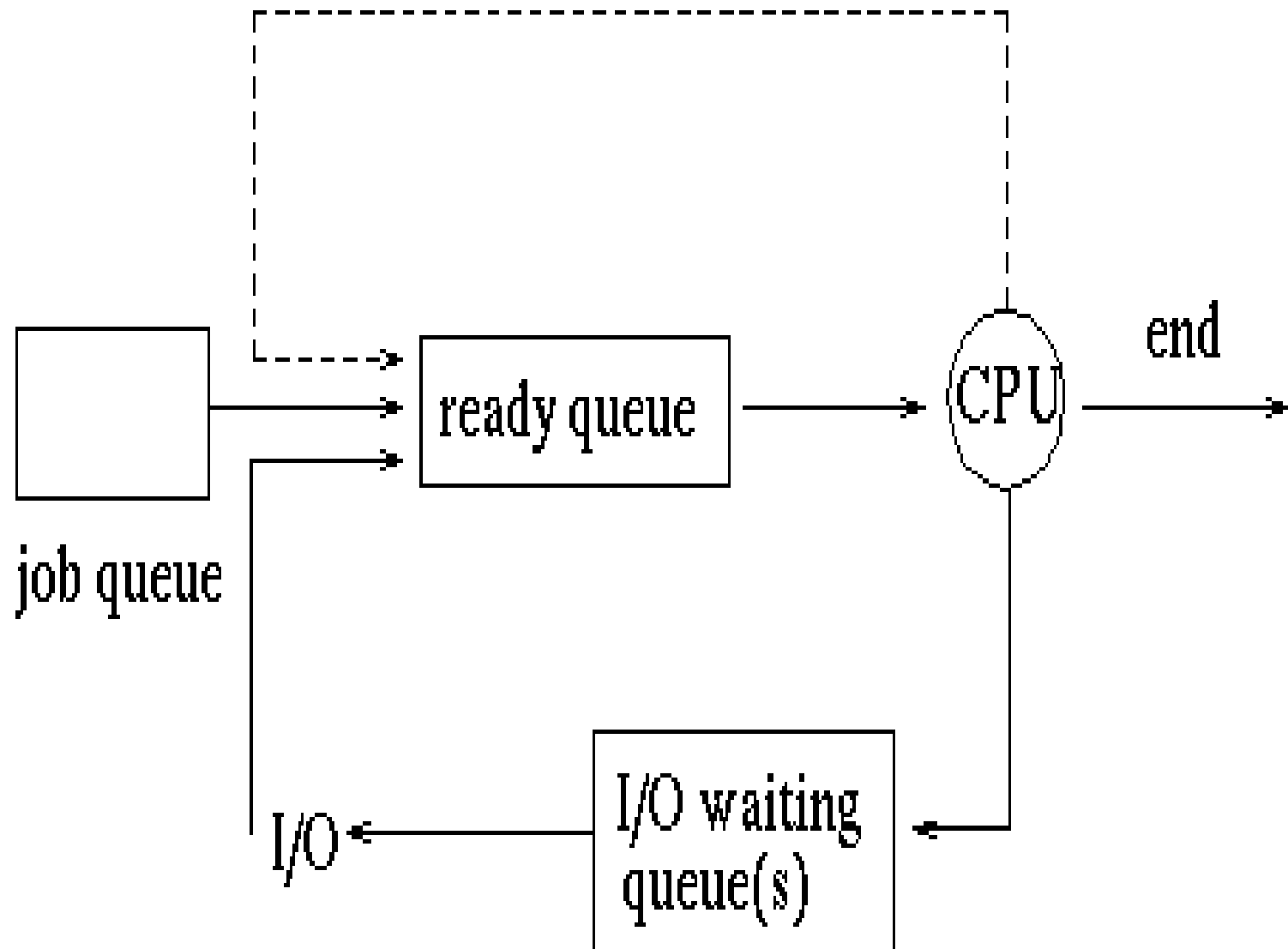
# Process Management / Process Scheduling

- **Multiprogramming operating system** allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time multiplexing.
- **The scheduling mechanism** is the part of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of particular strategy

# Scheduling Queues

- When the process enters into the system, they are put into a job queue. This queue consists of all processes in the system. The operating system also has other queues.
- Device queue is a queue for which a list of processes waiting for a particular I/O device. Each device has its own device queue.
- Fig. 3.3 shows the queuing diagram of process scheduling. In the fig 3.3,
  - ✓ **queue** is represented by rectangular box.
  - ✓ **The circles** represent the resources that serve the queues.
  - ✓ **The arrows** indicate the flow of processes in the system.

Cont...



*Fig. 3.3 Queuing Diagram*

## Cont...

- **Queues are of three types** :job queue, ready queue and set of device queues. A newly arrived process is put in the Job queue.
- Processes are waiting in ready queue for allocating the CPU.
- Once the CPU is assigned to the process, then process will execute. While executing the process, one of the several events could occur.
  1. The process could issue an I/O request and then place in an I/O queue.
  2. The process could create new sub process and waits for its termination.
  3. The process could be removed forcibly from the CPU, as a result of interrupt and put back in the ready queue.

# Two State Process Model

- **Process may be in one of two states :**
  - a) Running
  - b) Not Running
- When new process is created by OS, that process enters into the system in the running state.
- Processes that are not running are kept in queue, waiting their turn to execute.
- Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list.
- Use of dispatcher is as follows. When a process interrupted, that process is transferred in the ready queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then select a process from the queue to execute.



# Schedules

- **Schedulers are of two types.**
  1. Long Term Scheduler
  2. Short Term Scheduler

Cont...

## Long Term Scheduler

- It is also called job scheduler.
  - **Long term scheduler** determines which programs are admitted to the system for processing. Job scheduler selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduler.
  - **The primary objective of the job scheduler** is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
  - On same systems, the long term scheduler may be absent or minimal. Time-sharing operating systems have no long term scheduler. When process changes the state from new to ready, then there is a long term scheduler.
- I.e. - **Long-term scheduler** (job scheduler) - selects which processes should be brought into the ready queue.
- **Long-term scheduler** is invoked very infrequently (seconds, minutes) => (may be slow). The long-term scheduler controls the degree of multiprogramming.

Cont...

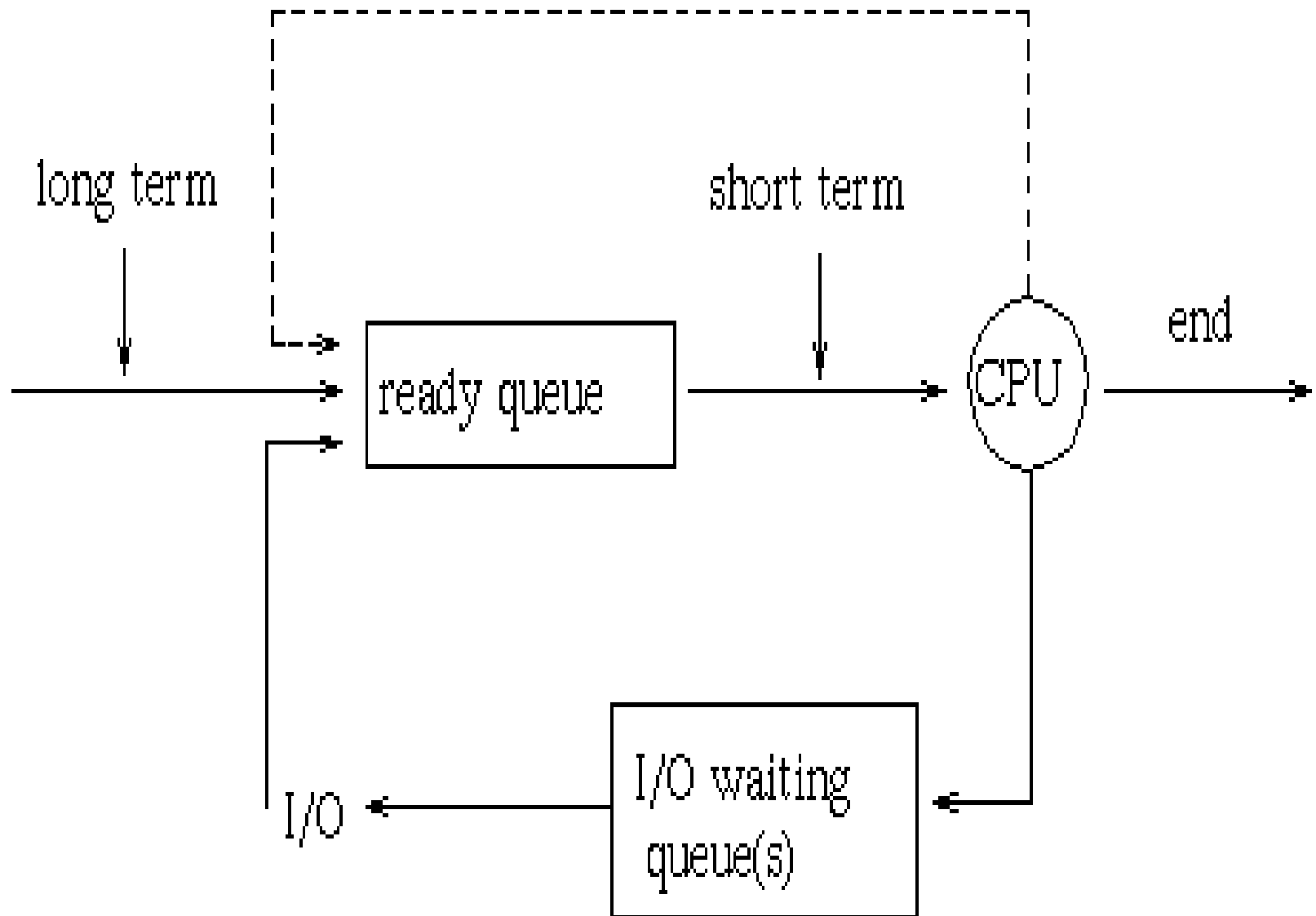
## Short Term Scheduler

- It is also called CPU scheduler. Main objective is increasing system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects from among the processes that are ready to execute and allocates the CPU to one of them.
- Short term scheduler also known as dispatcher, execute most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than long term scheduler.

I.e. - **Short-term scheduler** (CPU scheduler) - selects which process should be executed next and allocates CPU.

- **Short-term scheduler** is invoked very frequently (milliseconds)  
=> (must be fast).

Cont...



Cont...

## Context Switch

- Sometimes called process switch
- A context switch involves two processes:
  - One leaves the Running state
  - Another enters the Running state
- The status (context) of one process is saved; the status of the second process restored.
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching. Time dependent on hardware support.

# Operation on Processes

- Several operations are possible on the process. Process must be created and deleted dynamically. Operating system must provide the environment for the process operation. We discuss the two main operations on processes.
  - A. Create a process
  - B. Terminate a process

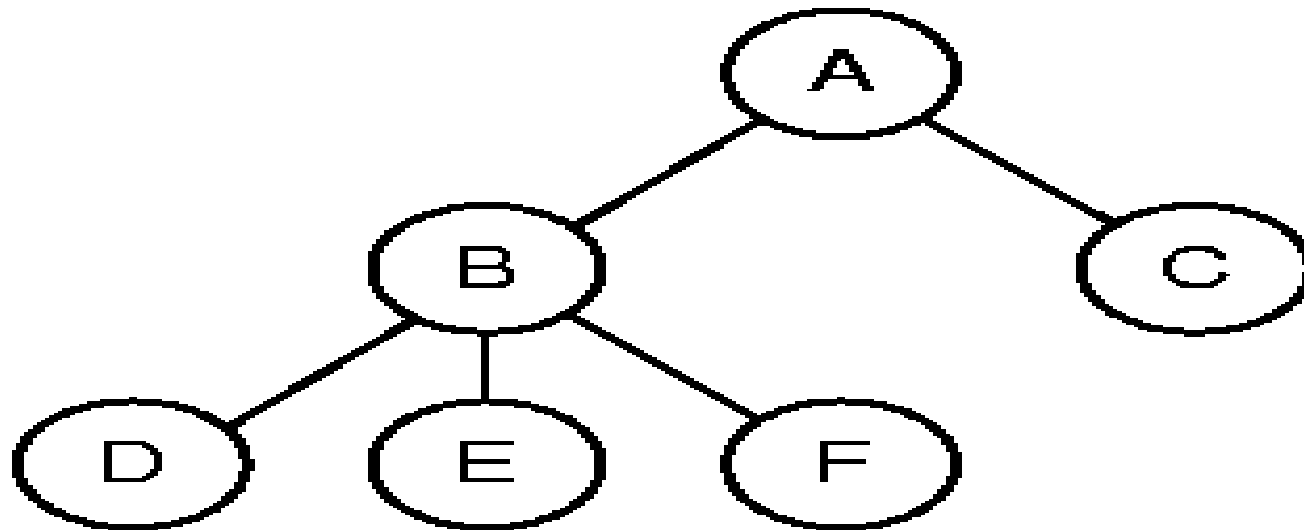
## Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- When a process creates a new process, two possibilities exist in terms of execution.
  1. The parent continues to execute concurrently with its children.
  2. The parent waits until some or all of its children have terminated

Cont...

**For address space**, two possibilities occur:

1. The child process is a duplicate of the parent process.
2. The child process has a program loaded into it.







Cont...

### **Resource sharing - 3 possibilities**

- Parent and children share all resources.
- Children share subset of parent's resources.
- Parent and child share no resources.

.

Cont...

## Process Termination

- ❑ Process executes last statement and asks the operating system to delete it (exit).
  - Output data from child to parent (via fork).
  - Process' resources are deallocated by operating system.
- ❑ Parent may terminate execution of children processes (abort).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
- ❑ Operating system does not allow child to continue if its parent terminates.
  - Cascading termination.



Cont...

Following are the resources for terminating the child process by parent process.

1. The task given to the child is no longer required.
2. Child has exceeded its usage of some of the resources that it has been allocated.
3. Operating system does not allow a child to continue if its parent terminates.

# Co-operating Processes

- **Co-operating process** is a process that can affect or be affected by the other processes while executing. If suppose any process is sharing data with other processes, then it is called co-operating process.
- Benefit of the co-operating processes are :
  1. Sharing of information
  2. Increases computation speed
  3. Modularity
  4. Convenience
- Co-operating processes share the information : Such as a file, memory etc. System must provide an environment to allow concurrent access to these types of resources. Computation speed will increase if the computer has multiple processing elements are connected together. System is constructed in a modular fashion. System function is divided into number of modules.

# Summery

- A process is a program in execution. As a process executes, it changes state. The state of a process is defined by that process's current activity.
- Each process may be in one of the following states: new, ready, running, waiting, or terminated. Each process is represented in the operating system by its own process control block (PCB).
- A process, when it is not executing, placed in some waiting queue. There are two major classes of queues in an operating system: I/O request queues and the ready queue. The ready queue contains all the processes that are ready to execute and are waiting for the CPU.
- Each process is represented by a PCB and the PCBs can be linked together to form a ready queue. Long-term(job) scheduling is the selection of processes that will be allowed to contend for the CPU. Normally, long-term scheduling is heavily influenced by resources-allocation considerations, especially memory management. Short-term(CPU) scheduling is the selection of one process from the ready queue.

## cont...

- Operating systems must provide a mechanism for parent processes to create new child processes. The parent may wait for its children to terminate before proceeding, or the parent and children may execute concurrently. There are several reasons for allowing concurrent execution: information sharing computation speedup, modularity, and convenience.
- The processes executing in the operating system may be either independent processes or cooperating processes. Cooperating processes require an inter-process communication mechanism to communicate with each other. Principally, communication is achieved through two schemes: shared memory and message passing. The shared-memory method requires communicating processes through
- The use of these shared variables. In a shared-memory system, the responsibility for providing communication rests with the application programmers: the operating system needs to provide only the shared memory. The responsibility for providing communication may rest with the operating system itself. These two schemes are not mutually exclusive and can be used simultaneously within a single operating system.



# Model questions

Q.1 Define process and programs?

Q.2 Describe Process Control Block?

Q.3 Explain Scheduling Queues?

Q.4 Explain schedulers and its types?

Q.5 Explain context switch?

Q.6 Explain operation on processes?



# Chapter 4 :- Thread management

## ➤ Unit Structure

- ❖ Objectives
- ❖ **Introduction Of Thread**
- ❖ Types of Thread (User Level Thread and Kernel Level Thread)
- ❖ Advantage of Thread
- ❖ Multithreading Models
- ❖ Difference between User Level and Kernel Level Thread
- ❖ Difference between Process and Thread
- ❖ Summary
- ❖ Model Question

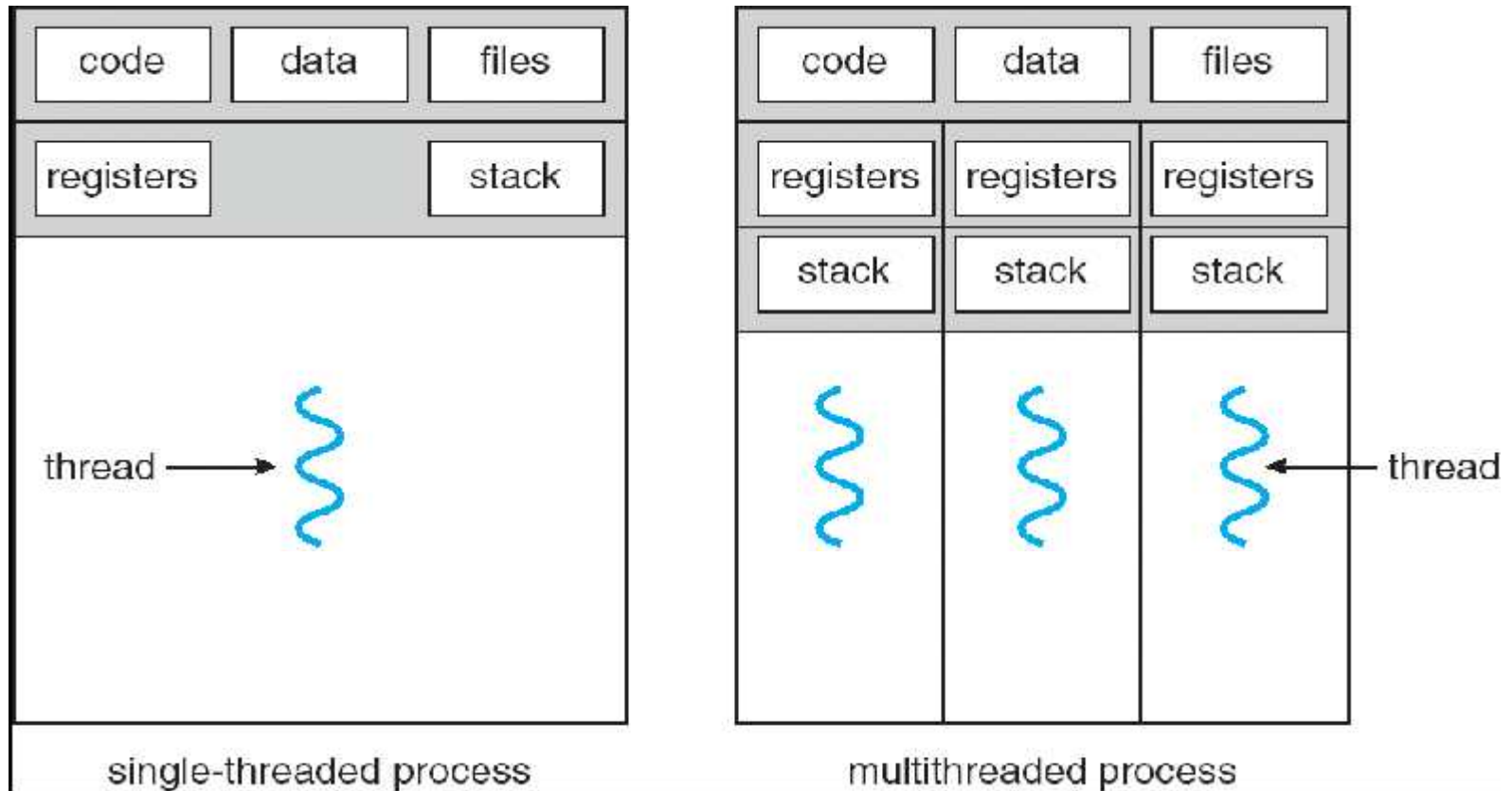
# Objective

- After going through this unit, you will be able to:
  - To introduce Thread concepts & its types.
  - To introduce Multithreading Models and Threading issues.

# Introduction of Thread

- **A thread** is a flow of execution through the process code, with its own program counter, system registers and stack. Threads are a popular way to improve application performance through parallelism. A thread is sometimes called a light weight process.
- **Threads** represent a software approach to improving performance of operating system by reducing the over head thread is equivalent to a classical process. Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
- Fig. 4.1 shows the single and multithreaded process.

## Cont....



- Threads have been successfully used in implementing network servers. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors

## Cont...

- A thread (or lightweight process) is a basic unit of CPU utilization; it consists of:
  - program counter
  - register set
  - stack space
- A thread shares with its peer threads its:
  - code section
  - data section
  - operating-system resources
- A traditional or heavyweight process is equal to a task with one thread. In a task containing multiple threads, while one server thread is blocked and waiting, a second thread in the same task could run.
  - Cooperation of multiple threads in same job confers higher throughput and
  - improved performance.
- Applications that require sharing a common buffer (producer-consumer problem) benefit from thread utilization.
- Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.

# Types of Thread

- Threads is implemented in two ways :
  1. User Level
  2. Kernel Level

## User Level Thread

- In a user thread, all of the work of thread management is done by the application and the kernel is not aware of the existence of threads.
- The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.
- The application begins with a single thread and begins running in that thread.
- User level threads are generally fast to create and manage



Cont...

**Advantage of user level thread over Kernel level thread :**

1. Thread switching does not require Kernel mode privileges
2. User level thread can run on any operating system.
3. Scheduling can be application specific.
4. User level threads are fast to create and manage.

**Disadvantages of user level thread :**

1. In a typical operating system, most system calls are blocking.
2. Multithreaded application cannot take advantage of multiprocessing.



# Kernel Level Threads

- In Kernel level thread, thread management done by the Kernel. There is no thread management code in the application area.
- Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- The Kernel maintains context information for the process as a whole and for individuals threads within the process.
- Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Cont...

- **Advantages of Kernel level thread:**

1. Kernel can simultaneously schedule multiple threads from the same process on multiple process.
2. If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
3. Kernel routines themselves can multithreaded.

- **Disadvantages:**

1. Kernel threads are generally slower to create and manage than the user threads.
- 2 Transfer of control from one thread to another within same process requires a mode switch to the Kernel.



Cont...

- **Advantages of Thread**

1. Thread minimize context switching time.
  2. Use of threads provides concurrency within a process.
  3. Efficient communication.
  4. Economy- It is more economical to create and context switch threads.
  5. Utilization of multiprocessor architectures –
- The benefits of multithreading can be greatly increased in a multiprocessor architecture.

# Multithreading Models

- Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach.
- In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process

## Cont... Difference between User Level and Kernel Level Thread

User Level Threads	Kernel Level Thread
<ul style="list-style-type: none"><li>• User level thread are faster to create and manage</li><li>• Implemented by a thread library at the user level.</li><li>• User level thread can run on any operating system.</li><li>• Support provided at the user level called user level thread.</li><li>• Multithread application cannot take advantage of multiprocessing.</li></ul>	<ul style="list-style-type: none"><li>• Kernel level thread are slower to create and manage.</li><li>• Operating system support directly to Kernel threads.</li><li>• Kernel level threads are specific to the operating system.</li><li>• Support may be provided by kernel is called Kernel level threads.</li><li>• Kernel routines themselves can be multithreaded</li></ul>

## Cont... **Difference between Process and Thread**

### **Process**

- Process is called heavy weight process.
- Process switching needs interface with operating system.
- In multiple process implementation each process executes the same code but has its own memory and file resources.
- If one server process is blocked no other server process can execute until the first process unblocked.
- Multiple redundant process uses more resources than multiple threaded.
- In multiple process each process operates independently of the others.

### **Thread**

- Thread is called light weight process.
- Thread switching does not need to call a operating system and cause an interrupt to the Kernel.
- All threads can share same set of open files, child processes.
- While one server thread is blocked and waiting, second thread in the same task could run.
- Multiple threaded process uses fewer resources than multiple redundant process.
- One thread can read, write or even completely wipe out another threads stack

# Summary

- **A thread** is a flow of control within a process. A multithreaded process contains several different flows of control within the same address space. The benefits of multithreading include increased responsiveness to the user, resource sharing within the process, economy, and scalability issues such as more efficient use of multiple core.
- **User level threads** are threads are visible to the programmer and are unknown to the kernel. The operating system kernel supports and manages kernel level threads. In general, user level threads are faster to create and manage than are kernel threads, as no intervention from the kernel is required. Three different types of models relate user and kernel threads: the many-to-one model maps many user threads to a single thread. The one to one model maps each user thread to a corresponding kernel thread. The many to many model multiplexers many user threads to a smaller or equal number of kernel threads.



# Model Question

Q1 Define thread and its types in detail?

Q2 Differentiate user level thread and kernel level thread?

Q3 Explain various Multithreaded Model?

Q4 Differentiate process and thread

# Chapter Five: CPU Scheduling

## Introduction

- CPU scheduling is the basis of multi programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive
- When a computer is multi-programmed, it frequently has multiple processes computing for the CPU at the same time.
- This situation occurs whenever two or more processes are simultaneously in the ready state.
- If only one CPU is available, a choice has to be made which process to run next.
- The part of the operating system that makes the choice is called the **scheduler** and the algorithm it uses is called the **scheduling algorithm**.



## CPU Scheduling

- Back in the old days of batch systems with input in the form of card images on a magnetic tape, the scheduling algorithm was simple: just run the next job on the tape.
- With timesharing systems, the scheduling algorithm became more complex because there were generally multiple users waiting for service.

## Basic Concepts

- The idea of multiprogramming is relatively simple. process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU would then just sit idle.
- Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use.



Cont...

- **CPU - I/O Burst Cycle**

The success of CPU scheduling depends on the following observed property of processes: Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states.

- **Context Switch**

To give each process on a multi programmed machine a fair share of the CPU, a hardware clock generates interrupts periodically. This allows the operating system to schedule all processes in main memory (using scheduling algorithm) to run on the CPU at equal intervals. Each switch of the CPU from one process to another is called a **context switch**.



Cont...

- **Preemptive Scheduling**

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for example, I/O request, or invocation of wait for the termination of one of the child processes).
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (for example, completion of I/O).
4. When a process terminates.

## CPU Scheduling :- Scheduling Criteria

- Different CPU scheduling algorithms have different properties and may favor one class of processes over another.
- In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.
- Many criteria have been suggested for comparing CPU scheduling algorithms.
- Criteria that are used include the following:
  - CPU utilization** - keep the CPU as busy as possible
  - Throughput** - number of processes that complete their execution per time unit
  - Turnaround time** - amount of time to execute a particular process
  - Waiting time** - amount of time a process has been waiting in the ready queue
  - Response time** - amount of time it takes from when a request was submitted until the first response is produced, not output (for time sharing environment)
  - Fairness** – comparable processes should get comparable services (gives each process a fair share of the CPU).



## **CPU Scheduling:- Optimization**

Max CPU utilization

Max throughput

Minimum turnaround time

Minimum waiting time

Minimum response time



## CPU Scheduling

- **Scheduling Algorithms**

### **1. First-Come, First-Served (FCFS) Scheduling**

- The simplest of all scheduling algorithm is non-preemptive FCFS scheduling algorithm.
- Processes are assigned the CPU in the order they request it.
- When the first job enters the system from the outside in the morning, it is started immediately and allowed to run as long as it wants too.
- As other jobs come in, they are put onto the end of the queue.
- When the running process blocks, the first process on the queue is run next.
- When a blocked process becomes ready, like a newly arrived job, it is put on the end of the queue.

# CPU Scheduling

- Example:
- *Process Burst time*
- $P_1 = 24, P_2 = 3, P_3 = 3$
- Suppose that the processes arrive in the order:  $P_1, P_2, P_3$ .
- A diagram to show this schedule is:



Waiting time for:

$P_1 = 0$

$P_2 = 24$

$P_3 = 27$

Average waiting time:  $(0 + 24 + 27)/3 = 17$



## **CPU Scheduling**

### **2. Shortest Job First (SJF) Scheduling**

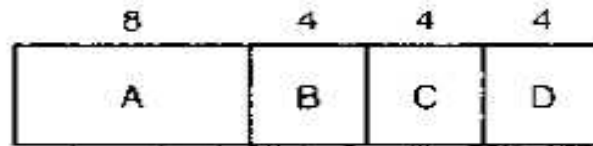
It is a non preemptive batch algorithm that assumes the run times are known in advance.

Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.

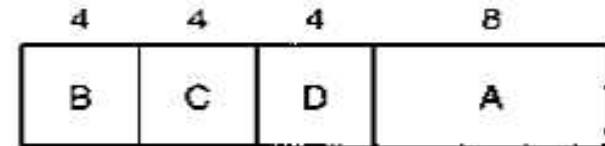
When several equally important jobs are sitting in the input queue waiting to be started, the scheduler picks the shortest job first.

For example: here we found four jobs A, B, C, and D with run times of 8, 4, 4, and 4 minutes respectively.

## CPU Scheduling



(a)



(b)

An example of shortest job first scheduling. (a) Running four jobs in the original order. (b) Running them in shortest job first order

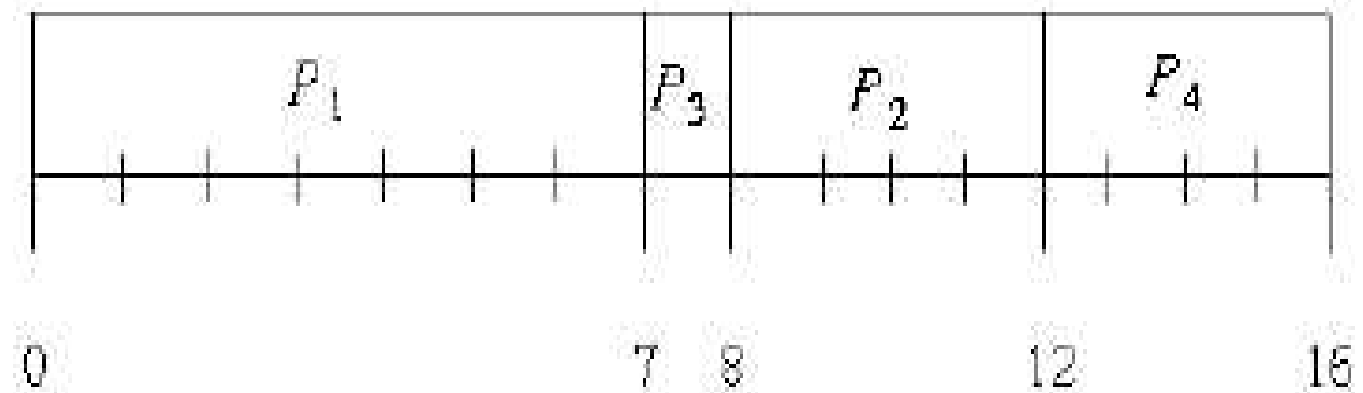
In the case of (a) turnaround time for A is 8 mints, For B is 12 mints. For C is 16 mints and for D is 20 mints. For an average of 14 mints. For the case of (b) ) turnaround for A is 4 mints. For B is 8 mints. For C is 12 mints and for D is 20 mints. For an average of 11 mints.

Shortest Job First is provably Optimal.. It is worth pointing out that shortest job first is only optimal when all the jobs are available simultaneously. Eg five processes having run times of 2, 4, 1, 1, 1 respectively. Their arrival times are 0, 0,3,3, and 3. check the difference of the average turn around time if they run in the order of A, B, C, D and E and in the order of B, C, D, E, A.

## CPU Scheduling

### Example of SJF

Process	Arrival time	CPU time
$P_1$	0	7
$P_2$	2	4
$P_3$	4	1
$P_4$	5	4



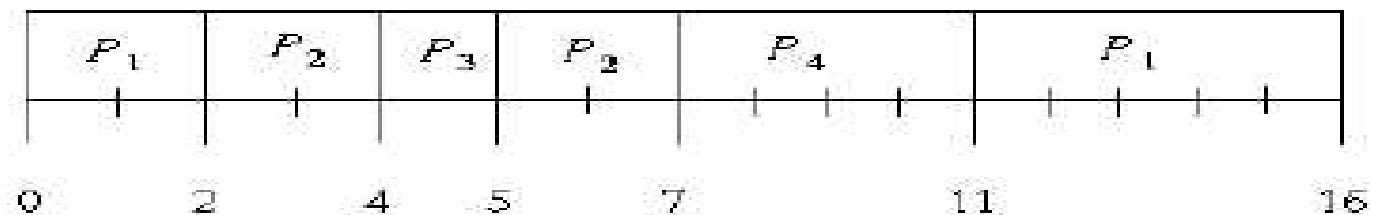
$$\text{Average waiting time} = (0 + 6 + 3 + 7)/4 = 4$$

# CPU Scheduling

## 3 Shortest Remaining Time Next

A preemptive version of shortest job first is **shortest remaining time next**. With this algorithm, the scheduler always chooses the process whose remaining run time is the shortest. Again here, the run time has to be known in advance. When a new job arrives, its total time is compared to the current process' remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job started. This scheme allows new short jobs to get good service.

SRTF (preemptive)



$$\text{Average waiting time} = (9 + 1 + 0 + 2)/4 = 3$$

# CPU Scheduling

## 4. Round-Robin (RR) Scheduling

One of the oldest, simplest, fairest, and mostly used algorithm

Each process is assigned a time interval called its **quantum**, which it is allowed to run.

If the process is still running at the end of the quantum, the CPU is preempted and given to another process.

It is easy to implement, all the scheduler needs to do is maintain a list of runnable processes.

When the process uses up its quantum, it is put on the end of the list.

The only interesting issue with round robin is the length of the quantum



## CPU Scheduling

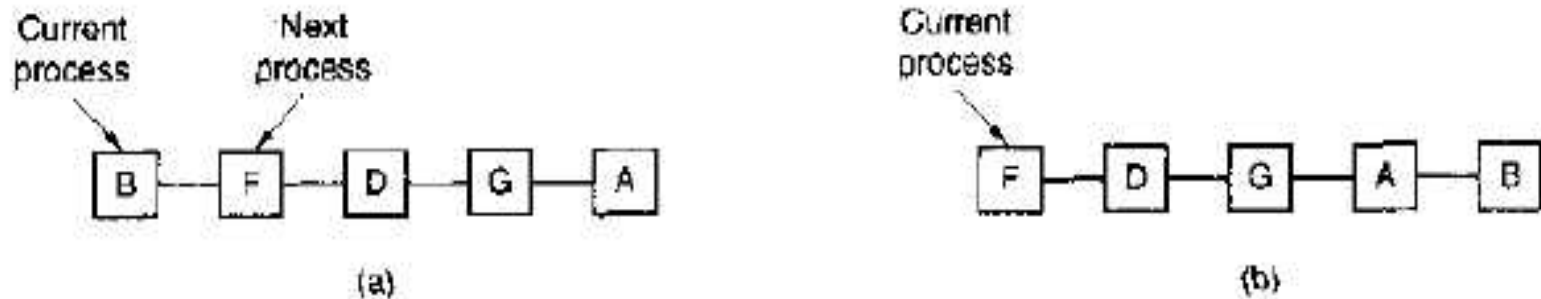


Fig. Round-Robin scheduling (a) the list of runnable processes. (b) the list of runnable processes after B uses up its quantum.

Setting the quantum too short causes too many processes switches and lowers the CPU efficiency. But setting it too long may cause poor response to short interactive request. A quantum around 20-50 msec. is often a reasonable compromise

## CPU Scheduling

Example: with time quantum = 20

Process	CPU times
$P_1$	53
$P_2$	17
$P_3$	68
$P_4$	24

$P_1$	$P_2$	$P_3$	$P_4$	$P_1$	$P_3$	$P_4$	$P_1$	$P_3$	$P_3$	
0	20	37	57	77	97	117	121	134	154	162

- Typically, higher average turnaround than SRTF, but better response.

# CPU Scheduling

## 5. Priority Scheduling

Round-Robin scheduling makes the implicit assumption that all processes are equally important.

The need to take external factors into account leads to priority scheduling.

The basic idea is each process is assigned a priority, and the runnable process with the highest priority is allowed to run.

(smallest integer means highest priority).

It can be preemptive or non-preemptive.

Problem: starvation (or indefinite blocking) – low priority processes may never execute.

Solution: aging as time progresses increase the priority of the process.

## CPU Scheduling

- Summary
- FCFS simple but causes short jobs to wait for long jobs.
- SJF is optimal giving shortest waiting time but need to know length of next burst.
- SJF is a type of priority scheduling - may suffer from starvation
  - prevent using aging
- RR is gives good response time, it is preemptive. Problem selecting the quantum.
- Multiple queue Algorithms use the best of each algorithm by having more than one queue. Feedback queues allow jobs to move from queue to queue.

# CPU Scheduling

- Exercise

Suppose that the ff jobs arrive as indicated for scheduling and execution on single CPU.

Job	Arrival time	Size (msec.)	Priority
J1	0	12	1 (Gold)
J2	2	4	3 (Bronze)
J3	5	2	1 (Gold)
J4	8	10	3 (Bronze)
J5	10	6	2 (Silver)

1. Draw a Gantt chart showing FCFS and calculate Avg. wait.Time and Avg. turn around time
2. Draw a Gantt chart showing non preemptive SJF and calculate Avg. wait.Time and Avg. turn around time
3. Draw a Gantt chart showing SRTF and calculate Avg. wait.Time and Avg. turn around time
4. Draw a Gantt chart showing RR (q=4 msec.) and calculate Avg. wait.Time and Avg. turn around time
5. Draw a Gantt chart showing Preemptive Priority and calculate Avg. wait. time and Avg. turn around time

# Chapter 6:- Concurrency Control

## ➤ Unit Structure

- ❖ Objectives
- ❖ Principal of Concurrency
- ❖ Race Condition
- ❖ Critical section
- ❖ Critical section problem and solution
- ❖ Summary
- ❖ Model Question

# Objectives

- After going through this unit, you will be able to:
  - To introduce the concurrency control and Race condition, critical section problem, where solutions can be used to ensure the consistency of shared data.
  - To present both software and hardware solutions of the critical section problem.



# Principle of concurrency

- **Concurrent processes** executing in the operating system may be either:-
  - ❖ **Cooperating processes** or
  - ❖ **Independent processes**
- A process is **independent** if it cannot affect or be affected by the other processes executing in the system.
- On the other hand, a process is **cooperating** if it can affect or be affected by the other processes executing in the system.
- **Cooperating processes** may either directly share a logical address space(that is, both code and data), or be allowed to share data only through files. The former case is achieved through the use of **lightweight processes or threads**.

## Cont...

- Processes (or threads) that **cooperate** to solve problems must exchange information. Two approaches:
  - Shared memory** (Shared memory is more efficient (no copying), but isn't always possible)
  - Message passing** (copying information from one process address space to another)
- There are several reasons for providing an environment that allows process cooperation
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience

# Race condition

- When several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called **a race condition**. this condition may result in **data inconsistency**.
- **A race condition** occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.

## Cont...

- Suppose that two processes, P1 and P2, share the global variable a. At some point in its execution, P1 updates a to the value 1, and at some point in its execution, P2 updates a to the value 2. Thus, the two tasks are in a race to write variable a. In this example the "loser" of the race (the process that updates last) determines the final value of a.
- Therefore Operating System Concerns of following things
  1. The operating system must be able to keep track of the various processes
  2. The operating system must allocate and deallocate various resources for each active process.
  3. The operating system must protect the data and physical resources of each process against unintended interference by other processes.
  4. The functioning of a process, and the output it produces, must be independent of the speed at which its execution is carried out relative to the speed of other concurrent processes.

## Cont...

- Process Interaction can be defined as
  - Processes unaware of each other
  - Processes indirectly aware of each other
  - Processes directly aware of each other
- Concurrent processes come into conflict with each other when they are competing for the use of the same resource.
- Two or more processes need to access a resource during the course of their execution. Each process is unaware of the existence of the other processes. There is no exchange of information between the competing processes.

# Critical-Section

- A **critical section** is the code that accesses shared data or resources.
- The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is **mutually exclusive in time**. Otherwise result critical-section problem
- The critical-section problem is to design a protocol that the processes can use to cooperate.
- Critical section problem is design an algorithm that allows at most one process into the critical section at a time.

-



Cont...

- A solution to the critical-section problem must satisfy the following three requirements:
  1. **Mutual Exclusion:** If process  $P_i$  is executing in its critical section, then no other processes can be executing in their critical sections.
  2. **Progress:** If no process is executing in its critical section and there exist some processes that wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision of which will enter its critical section next, and this selection cannot be postponed indefinitely.
  - 3 **Bounded Waiting:** There exist a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.





# Model Question

Q.1 Explain in brief race condition?

Q.2 Define the term critical section?

Q.3 What are the requirement for critical section problem?

# CHAPTER 7 :- DEADLOCK

## ➤ Unit Structure

- Introduction
- Deadlock Characterization
- Method for Handling Deadlock
  - Deadlock Prevention Recovery
  - Avoidance and Protection
  - Deadlock Detection
- Summary
- Model Question

# Objectives

- After going through this unit, you will be able to:
  - To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks.
  - To present number of different methods for preventing or avoiding deadlocks in a computer system.

# Introduction

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a wait state. It may happen that waiting processes will never again change state, because the resources they have requested are held by other waiting processes. This situation is called a **deadlock**.
- If a process requests an instance of a resource type, the allocation of any instance of the type will satisfy the request. If it will not, then the instances are not identical, and the resource type classes have not been defined properly.
- A process must request a resource before using it, and must release the resource after using it. A process may request as many resources as it requires to carry out its designated task.

## Cont...

- Under the normal mode of operation, a process may utilize a resource in only the following sequence:
  1. **Request:** If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.
  2. **Use:** The process can operate on the resource.
  3. **Release:** The process releases the resource

# Deadlock Characterization

- In a deadlock, processes never finish executing and system resources are tied up, preventing other jobs from ever starting.
- **Necessary Conditions**

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

## Cont...

1. **Mutual exclusion:** At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2. **Hold and wait :** There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
3. **No preemption :** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process, has completed its task.
4. **Circular wait:** There must exist a set  $\{P_0, P_1, \dots, P_n\}$  of waiting processes such that  $P_0$  is waiting for a resource that is held by  $P_1$ ,  $P_1$  is waiting for a resource that is held by  $P_2$ , ...,  $P_{n-1}$  is waiting for a resource that is held by  $P_n$ , and  $P_n$  is waiting for a resource that is held by  $P_0$ .



# Methods for Handling Deadlocks

➤ Principally, there are three methods for dealing with the deadlock problem. Those are:-

- ❖ **Deadlock prevention:-** by making at least one of the necessary condition can not hold.
- ❖ **Deadlock avoidance:-** using protocol to ensure that the system will never enter deadlock state.
- ❖ **Deadlock detection:-** allowing the system to enter in to dead lock state and recover.

# Deadlock Prevention

- For a deadlock to occur, each of the four necessary-conditions must hold. By ensuring that at least on one these conditions cannot hold, we can prevent the occurrence of a deadlock.

**Mutual Exclusion** – not required for sharable resources; must hold for non sharable resources

**Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.

**No Preemption** – If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

**Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

# Deadlock Avoidance

- Requires that the system has some additional a priori information available.
  - Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
  - The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
  - Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

# Deadlock Detection

➤ If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system must provide:

An algorithm that examines the state of the system to determine whether a deadlock has Occurred.

An algorithm to recover from the deadlock

# Summary

- A deadlocked state occurs when two or more processes are waiting indefinitely for an event that can be caused only one of the waiting processes. There are three principal methods for dealing with deadlocks:
  - Use some protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
  - Allow the system to enter a deadlocked state, detect it, and then recover.
  - Ignore the problem altogether and pretend that deadlocks never occur in the system.
- Deadlock occur only when some process makes a request that cannot be granted immediately.

# Model Question

Q.1 Write a short note on deadlock?

Q.2 Explain the characteristic of deadlock?

Q.3 Describe various methods for deadlock prevention?

Q.4 Explain how deadlocks are detected and corrected?

Q.5 What are the difference between a deadlock prevention and deadlock Avoidance?

# UNIT 8: Memory and Device Management

## 5.1 Memory Management

### Introduction

- ❑ Memory is an important resource that must be carefully managed.
- ❑ The average home computer nowadays has a thousand times as much memory as the IBM 7094.
- ❑ Programs expand to fill the memory available to hold them.
- ❑ Ideally every programmer like an infinitely large, infinitely fast, nonvolatile and inexpensive memory.
- ❑ Unfortunately technology does not provide such memories.
- ❑ Consequently, most computers have a memory hierarchy.
  - ❑ Small amount, very fast, expensive and volatile (cache memory)
  - ❑ Tens of megabytes, medium speed, volatile (RAM)
  - ❑ Tens or hundreds of gigabytes, slow, cheap and nonvolatile (disk storage)

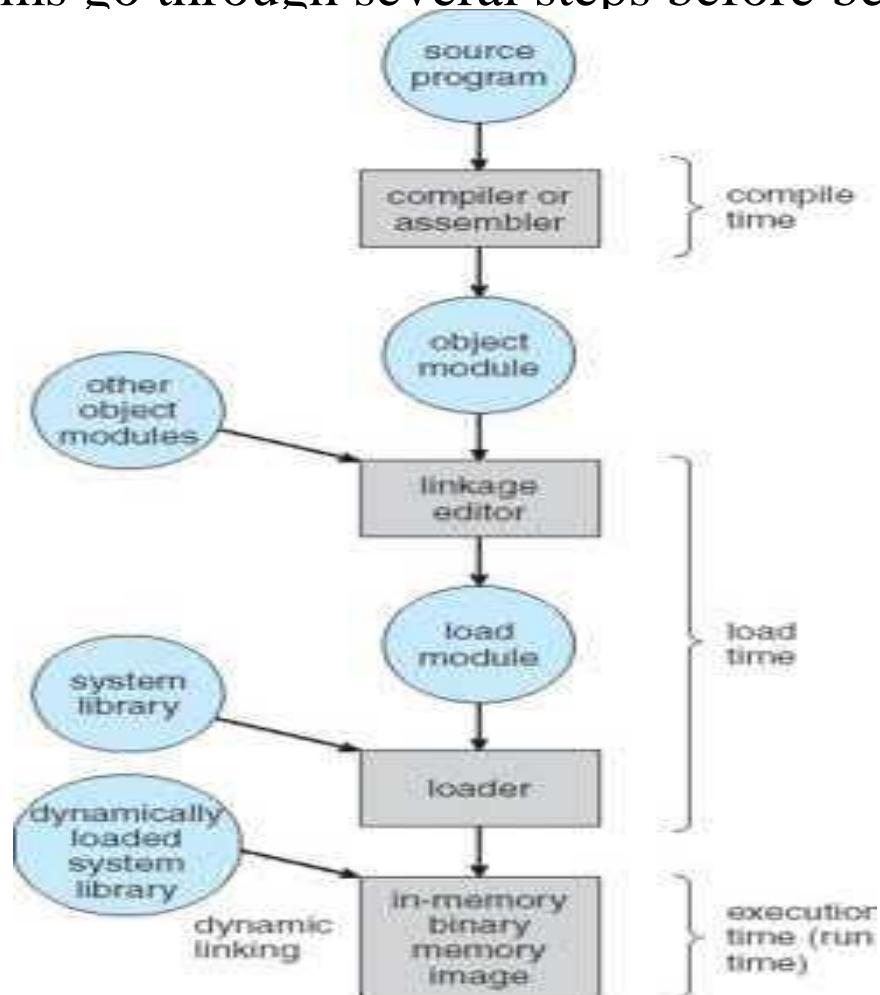


## Cont...

- ❑ It is the job of the operating system to coordinate how these memories are used.
- ❑ In this chapter we will study how OS manage memory.
- ❑ The part of the OS that manages the memory hierarchy is called the **memory manager**.
- ❑ It keeps track of which parts of memory are in use and which parts are not in use, to allocate memory to processes when they need it and deallocate it when they are done, and to manage swapping b/n main memory and disk when main memory is too small to hold all the processes
- ❑ There are a number of d/t memory management schemes ranging from simple to highly sophisticated.

## Cont...

- ❑ Program must be brought into memory and placed within a process for it to be executed.
- ❑ User programs go through several steps before being executed.



## Cont...

Address binding of instructions and data to memory addresses can happen at three stages:

- ❑ Compile time: If memory location known a priori, absolute code can be generated;
- ❑ must recompile code if starting location changes.
- ❑ Load time: Must generate relocatable code if memory location is not known at compile time.
- ❑ Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).
- ❑ Dynamic Loading - routine is not loaded until it is called.
  - o Better memory-space utilization; unused routine is never loaded.
  - o Useful when large amounts of code are needed to handle infrequently occurring cases.
- ❑ Dynamic Linking - linking postponed until execution time.
  - o Small piece of code, stub, used to locate the appropriate memory-resident library routine.
  - o Stub replaces itself with the address of the routine, and executes the routine.
  - o Operating system needed to check if routine is in processes' memory address.

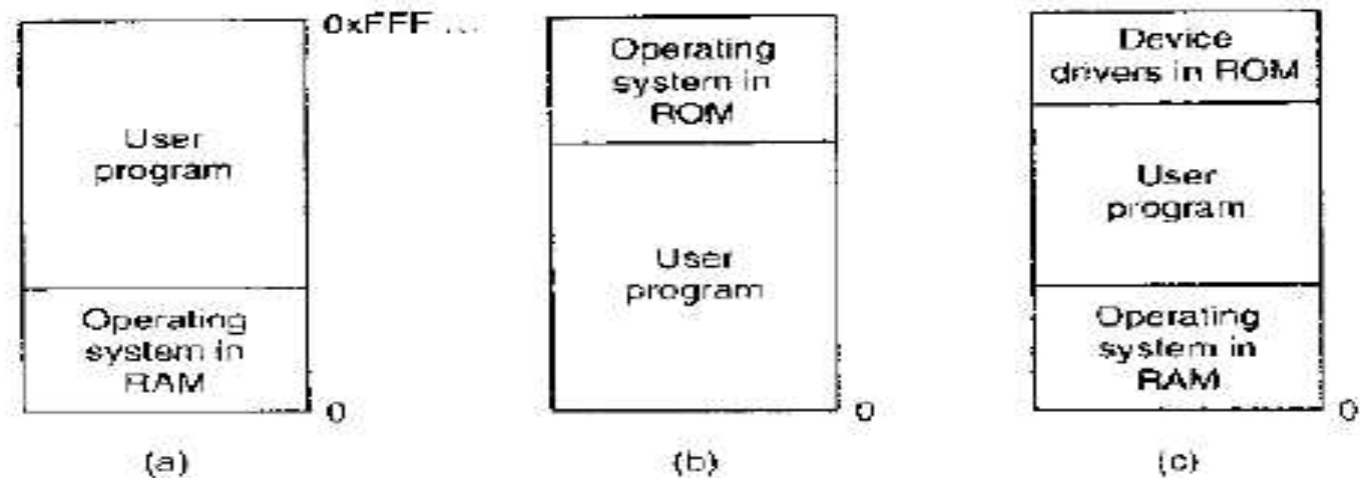
## Cont...

### Basic memory management

- ❑ Memory management can be divided into two classes: those that move processes back and forth b/n main memory and disk during execution (swapping and paging) and those that do not.

### Mono-programming without swapping or paging

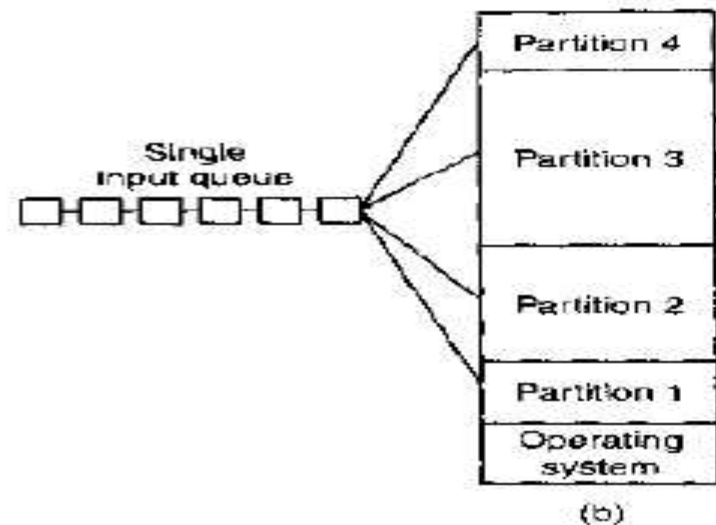
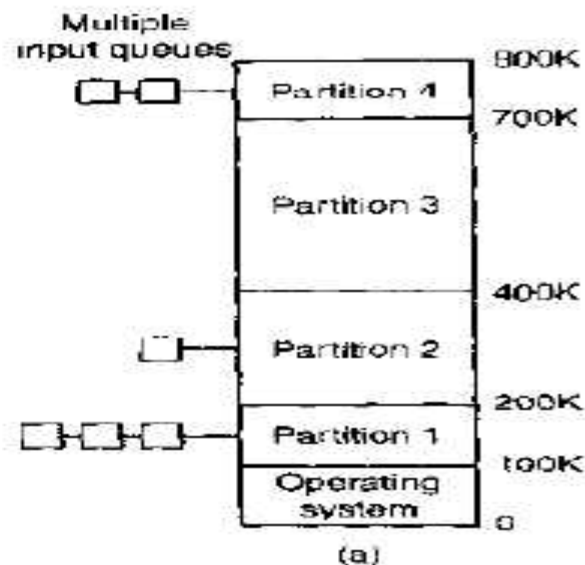
- ❑ The simplest possible memory management scheme is to run just one program at a time, sharing the memory b/n that program and the OS.



## Cont...

### Multiprogramming with fixed partitions

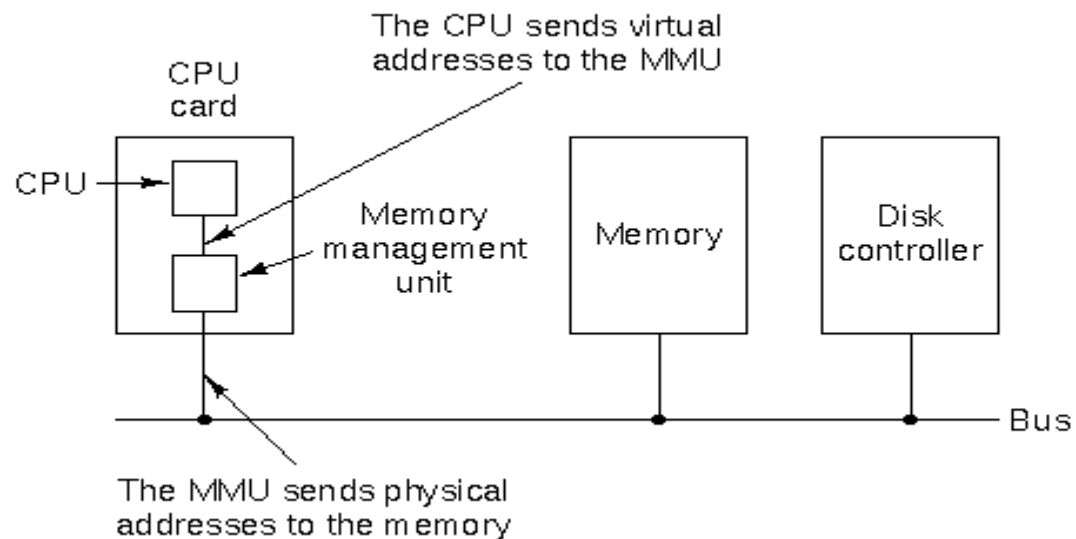
- ❑ Most modern systems allow multiple processes to run at the same time.
- ❑ The easiest way to achieve multiprogramming is simply to divide memory up into 'n' (possible unequal) partitions.
- ❑ When a job arrives, it can be put into the input queue for the smallest partition large enough to hold it.
- ❑ Any space in a partition not used by a job is lost.



## Cont...

### Logical versus Physical Address Space

- ❑ The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
- ❑ Logical address - generated by the CPU; also referred to as virtual address.
- ❑ Physical address - address seen by the memory unit.
- ❑ Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.
- ❑ Memory-management unit (MMU) - hardware device that maps virtual to physical address.



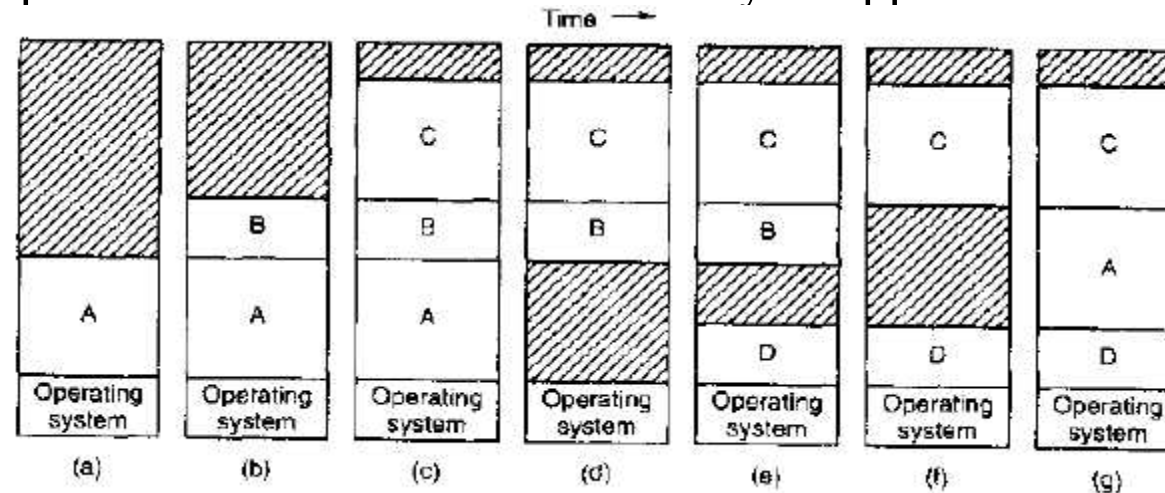


## Cont...

- In MMU scheme, the value in a relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with logical addresses; it never sees the real physical addresses.

## Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- Backing store - fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.



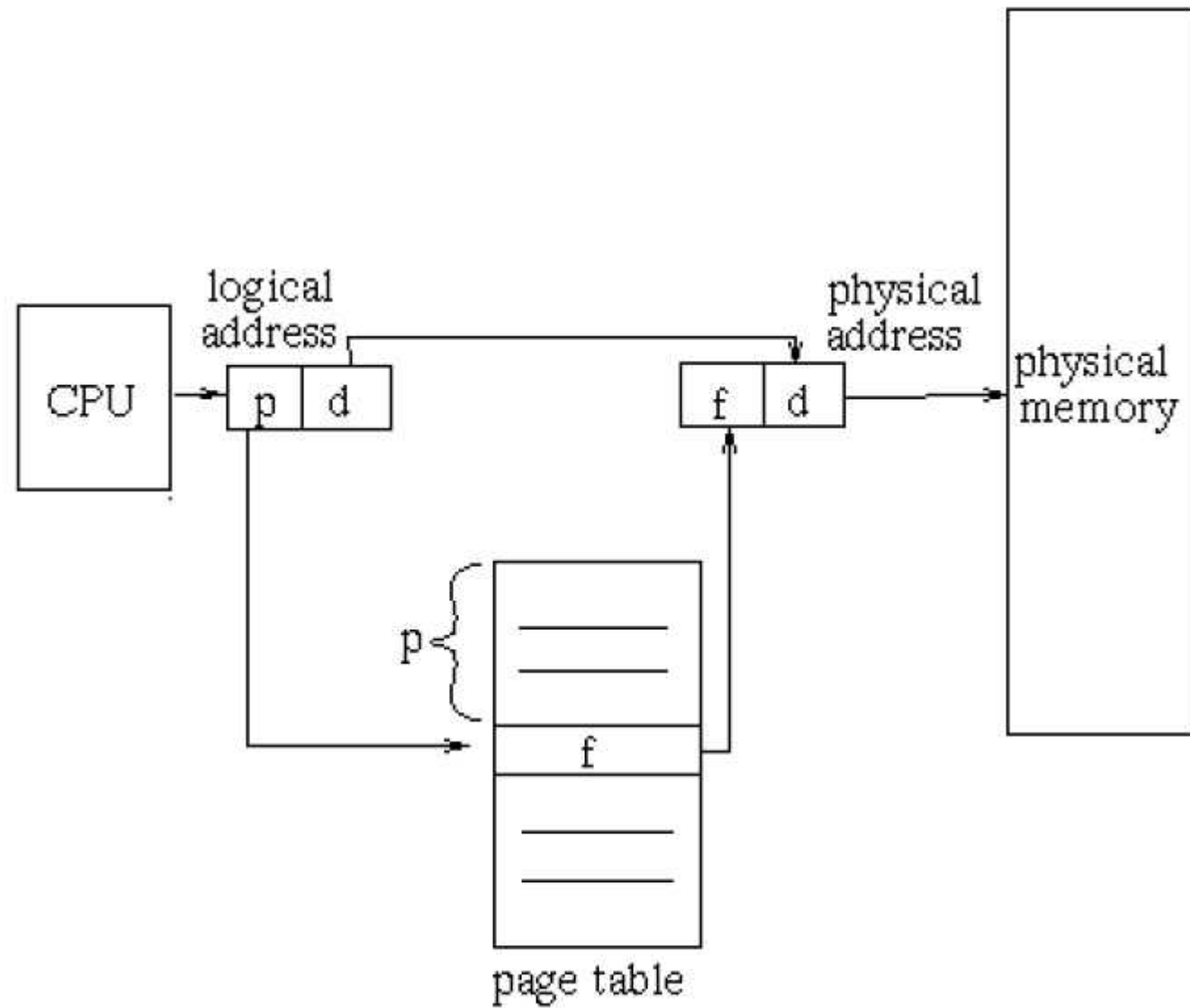


## Cont...

### Paging

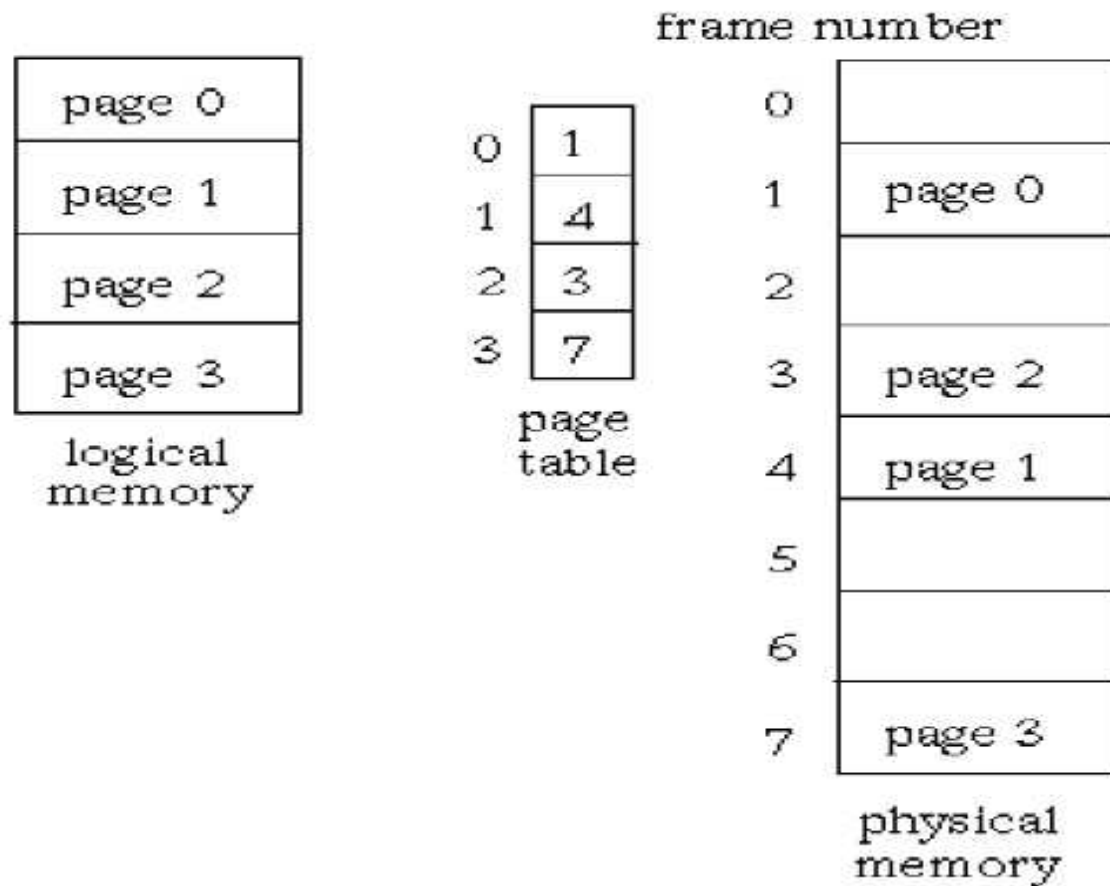
- ❑ Divide physical memory into fixed-sized blocks called **frames** (**size is power of 2**, between 512 bytes and 8192 bytes).
- ❑ Divide logical memory into blocks of same size called **pages**.
- ❑ Keep track of all free frames.
- ❑ To run a program of size  $n$  pages, need to find  $n$  free frames and load program.
- ❑ Set up a page table to translate logical to physical addresses.
- ❑ Address generated by CPU is divided into:
  - ❑ Page number ( $p$ ) - used as an index into a page table which contains base address of each page in physical memory.
  - ❑ Page offset ( $d$ ) - combined with base address to define the physical memory address that is sent to the memory unit.

## Cont...



## Cont...

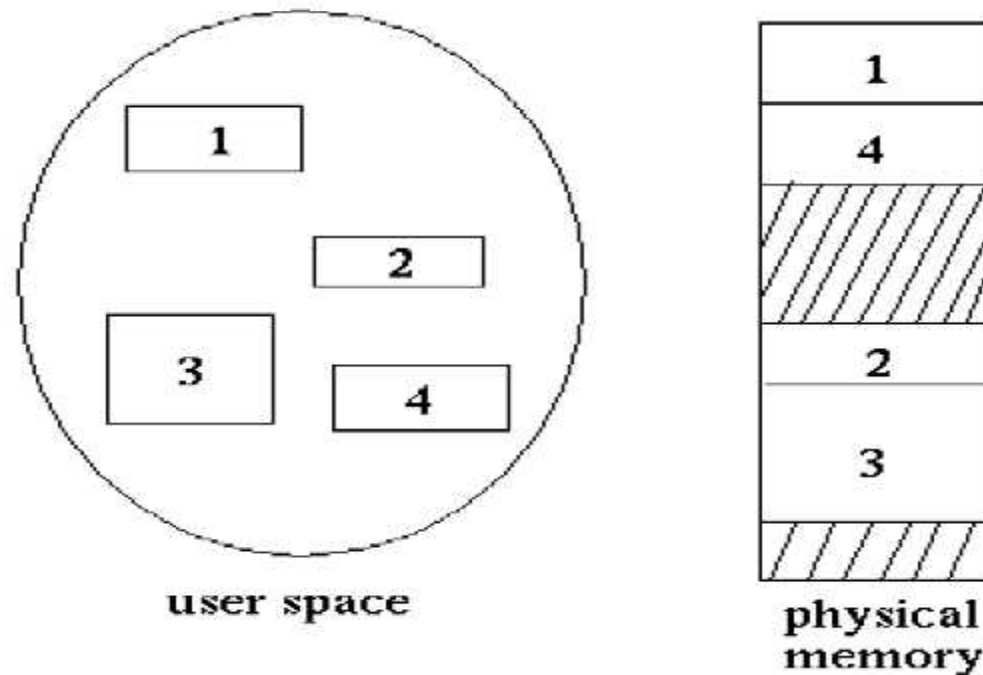
- ❑ Separation between user's view of memory and actual physical memory reconciled by address translation hardware; logical addresses are translated into physical addresses.



## Cont...

### Segmentation

- ❑ memory-management scheme that supports user view of memory.
- ❑ A program is a collection of segments. A segment is a logical unit such as: Code, local variables, global variables, stack
- ❑ • Example



## Cont...

- ❑ Logical address consists of a two tuple:
- ❑ <segment-number, offset>
- ❑ A segment table maps two-dimensional user defined addresses into one dimensional
- ❑ physical addresses; each entry in the table has:
  - ❑ base - contains the starting physical address where the segments reside in memory.
  - ❑ limit - specifies the length of the segment. as: Code, local variables, global variables, stack