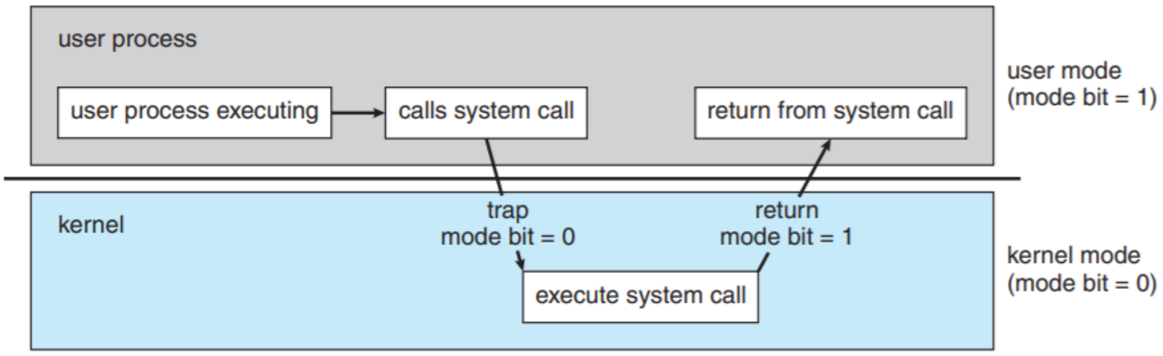
UNIT 1:

1. **DEFINE OS & explain OS services.**
   An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs.

2. **Explain the dual mode operation of OS and its need.**
   In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.
   we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode).
   A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfil the request.
   Transformation from user to kernel mode is as shown below.



   At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.
   The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/O control, timer management, and interrupt management.

3. **List the activities of OS in connection with process and memory management.**

Process Management:

A process needs certain resources—including CPU time, memory, files, and I/O devices—to accomplish its task. These resources are either given to the process when it is created or allocated to it while it is running. When the process terminates, the operating system will reclaim any reusable resources.
a program by itself is not a process. A program is a passive entity, like the contents of a file stored on disk, whereas a process is an active entity. A single-threaded process has one program counter specifying the next instruction to execute. The execution of such a process must be sequential. The CPU executes one instruction of the process after another, until the process completes.
A process is the unit of work in a system. A system consists of a collection of processes, some of which are operating-system processes and the rest of which are user processes. All these processes can potentially execute concurrently—by multiplexing on a single CPU, for example:
• Scheduling processes and threads on the CPUs.
• Creating and deleting both user and system processes.
• Suspending and resuming processes.
• Providing mechanisms for process synchronization.
• Providing mechanisms for process communication.

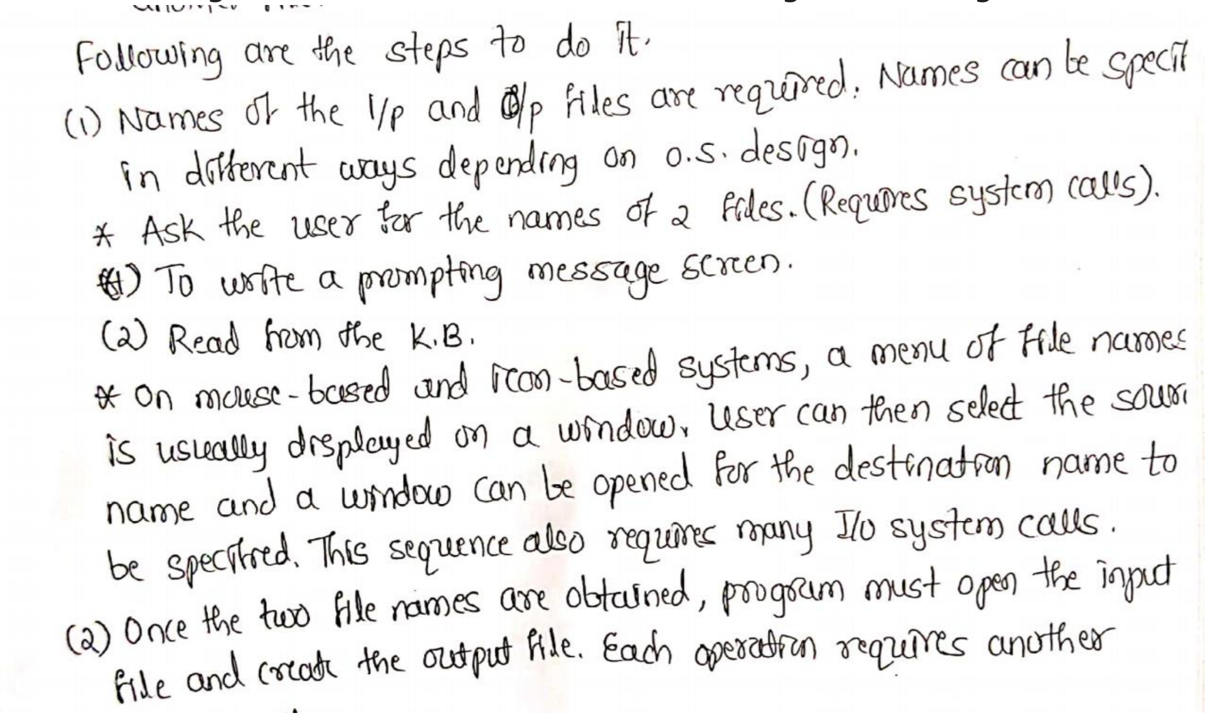
Memory Management:

Main memory is a large array of bytes, ranging in size from hundreds of thousands to billions. Each byte has its own address. Processor reads instructions from main memory during the instruction-fetch cycle and both reads and writes data from main memory during the data-fetch cycle (on a von Neumann architecture). After execution the program terminates and the memory space will be freed so that the next program can be loaded and executed.
To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.
In selecting a memory-management scheme for a specific system, we must take into account many factors—especially the hardware design of the system. Each algorithm requires its own hardware support.
The operating system is responsible for the following activities in connection with memory management:
• Keeping track of which parts of memory are currently being used and who is using them.
• Deciding which processes (or parts of processes) and data to move into and out of memory.
• Allocating and deallocating memory space as needed.

4. **What is Distributed System, explain different types of networks used in distributed system.**
   A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains. Distributed systems depend on networking for

   their functionality. Networks vary by the protocols used, the distances between nodes, and the transport media.
   Networks are characterized based on the distances between their nodes. A local-area network (LAN) connects computers within a room, a building, or a campus. A wide-area network (WAN) usually links buildings, cities, or countries. A global company may have a WAN to connect its offices worldwide, for example. These networks may run one protocol or several protocols. The continuing advent of new technologies brings about new forms of networks. For example, a metropolitan-area network (MAN) could link buildings within a city. Bluetooth and 802.11 devices use wireless technology to communicate over a distance of several feet, in essence creating a personal-area network (PAN) between a phone and a headset or a smartphone and a desktop computer.

5. **What is System Call? Explain set of system calls from one file to another file.**
   A **system call** is a request made by a program to the **operating system**. It allows an application to access functions and commands from the **operating system's** API. **System calls** perform **system**-level operations, such as communicating with hardware devices and reading and writing files.


Following are the steps to do it.
(1) Names of the I/p and O/p files are required. Names can be specif
in different ways depending on O.S. design.
* Ask the user for the names of 2 files. (Requires system calls).
(1) To write a prompting message screen.
(2) Read from the K.B.
* On mouse-based and icon-based systems, a menu of file names
is usually displayed on a window. User can then select the sourc
name and a window can be opened for the destination name to
be specified. This sequence also requires many I/O system calls.
(2) Once the two file names are obtained, program must open the input
file and create the output file. Each operation requires another

system call.

There are also possible error conditions for each operation. i.e

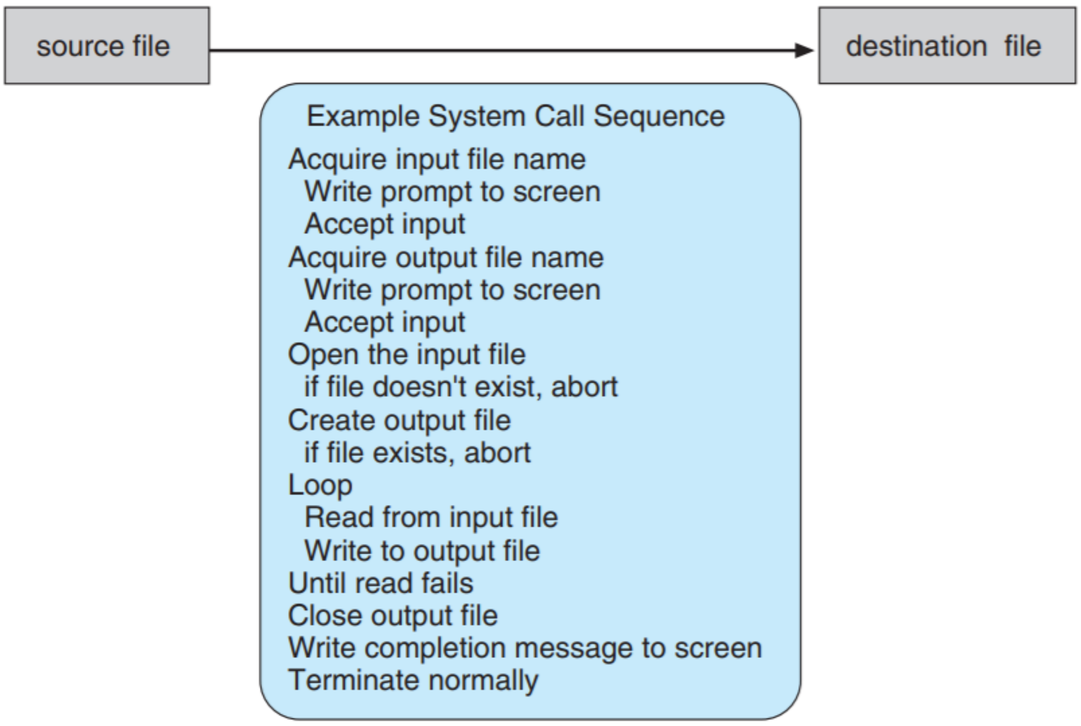
* It may not find a file with that name or the file is protected against access. Then program should print a message on the console (another sequence of system calls), then terminate the program (" " " " ") abnormally.

(3) If the i/p file exists, we must create new o/p file. If the file with same name exists, this may cause the program to abort (a system call) or we have to delete the existing file (system call) and create a new one (system call). In an interactive system, ask the user to replace the existing file (system call) or to abort the progra

(4) Both the files are setup. Data from i/p file will be read (System and write to the o/p file (system call). Each read and write opera should return status information regarding possible error condition

(5) After the entire file is copied, program may close the file (s.c), write a message on the console or terminal (s.c)

From the above example, we can see that even simple programs may make heavy use of O.S. Hence systems execute thousands of system calls per second. This system call sequence is as shown below.



Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

6. **Explain different types of System calls.**

   • Process control: This system calls perform the task of process creation, process termination, etc.
   - ◦ end, abort
   - ◦ load, execute
   - ◦ create process, terminate process
   - ◦ get process attributes, set process attributes
   - ◦ wait for time ◦ wait event, signal event
   - ◦ allocate and free memory

   • File management: File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

   - ◦ create file, delete file
   - ◦ open, close
   - ◦ read, write, reposition
   - ◦ get file attributes, set file attributes

   • Device management: Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
   - ◦ request device, release device
   - ◦ read, write, reposition
   - ◦ get device attributes, set device attributes
   - ◦ logically attach or detach devices

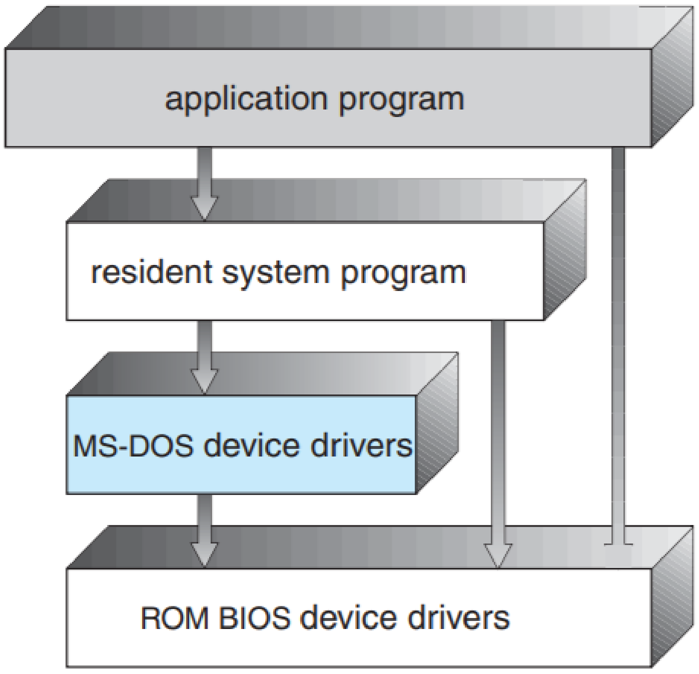
   • Information maintenance: It handles information and its transfer between the OS and the user program.
   - ◦ get time or date, set time or date
   - ◦ get system data, set system data
   - ◦ get process, file, or device attributes
   - ◦ set process, file, or device attributes

   • Communications: These types of system calls are specially used for inter process communications.

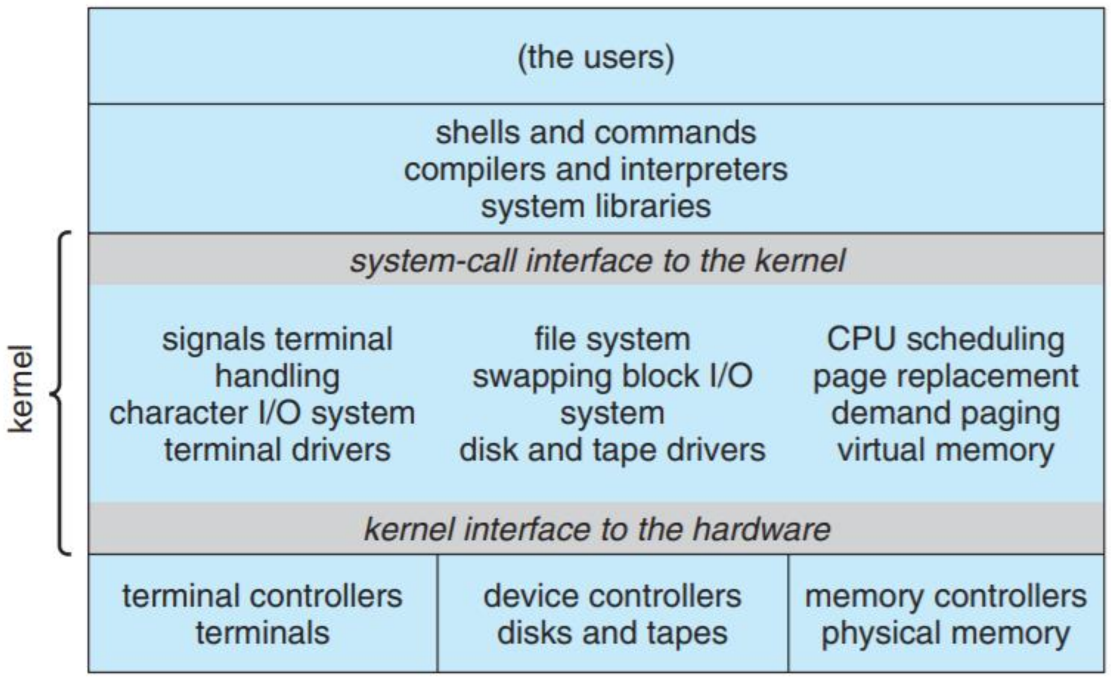
   - ◦ create, delete communication connection
   - ◦ send, receive messages
   - ◦ transfer status information
   - ◦ attach or detach remote devices.

7. **Explain OS structures.**
   A common approach is to partition the task into small components, or modules, rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.

Simple Structure: Many operating systems do not have well-defined structures. Such systems started as small, simple, and limited systems. Ex: MS-DOS.
It was written to provide the most functionality in the least space, so it was not carefully divided into modules. Fig. below shows its structure,



MS-DOS Structure: In MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail.
Another example of limited structuring is the original UNIX operating system. Like MS-DOS, UNIX initially was limited by hardware functionality. It consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved. We can view the traditional UNIX operating system as being layered to some extent, as shown in Fig. Everything below the system-call interface and above the physical hardware is the kernel. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls. Taken in sum, that is an enormous amount of functionality to be combined into one level. This monolithic structure was difficult to implement and maintain.



Traditional UNIX system structure.

UNIT – 2:

1. **What is process? Explain process state diagram.**

A process is basically a program in execution. The execution of a process must progress in a sequential fashion. A process is defined as an entity which represents the basic unit of work to be implemented in the system.

Process State:

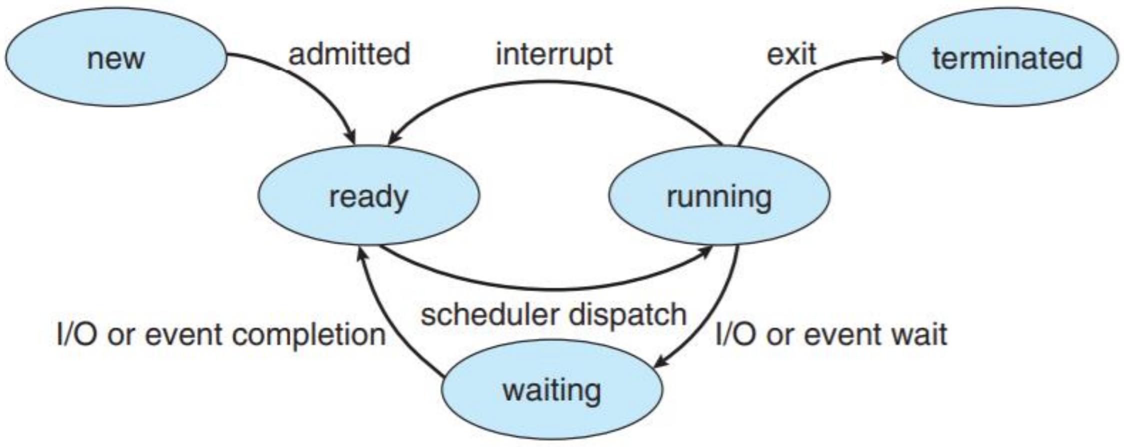


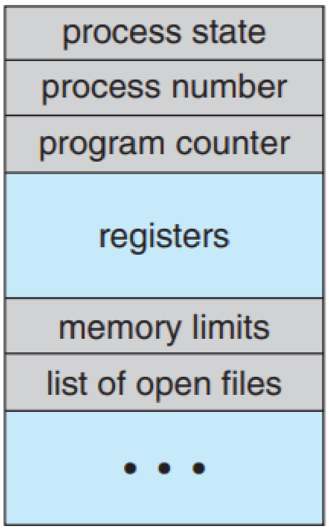
Figure 3.2 Diagram of process state.

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

• New. The process is being created.
• Running. Instructions are being executed.
• Waiting. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
• Ready. The process is waiting to be assigned to a processor.
• Terminated. The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Certain operating systems also more finely delineate process states. It is important to realize that only one process can be running on any processor at any instant. Many processes may be ready and waiting, however. The state diagram corresponding to these states is presented in Figure 3.2.

2. **What is PCB? Explain its components.**
   A **process control block (PCB)** is a data structure used by computer operating systems to store all the information about a process. It is also known as a process **descriptor**.



- Process state: The state may be new, ready, running, waiting, halted, and so on.
- Program counter: The counter indicates the address of the next instruction to be executed for this process.
- CPU registers: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs.
- CPU-scheduling information: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- Memory-management information: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.
- Accounting information: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

- I/O status information: This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

3. **Write a note on different types of schedulers.**

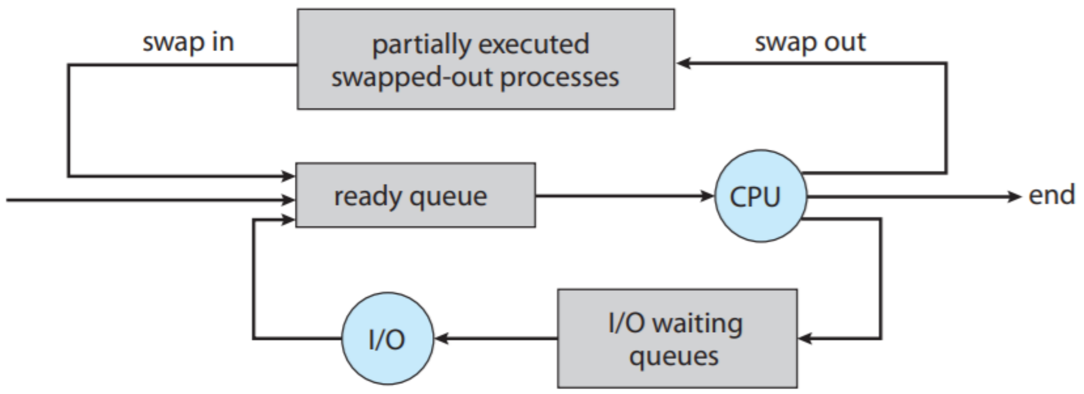
   • **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

   - Sometimes the only scheduler in a system
   - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)

   • **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue

- Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the degree of multiprogramming

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good process mix
- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: swapping



Addition of medium-term scheduling to the queueing diagram.

4. Explain Scheduling cafeterias.

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favour 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.

- CPU utilization. We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from

  40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).
- Throughput. If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second.
- Turnaround time. From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of

submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

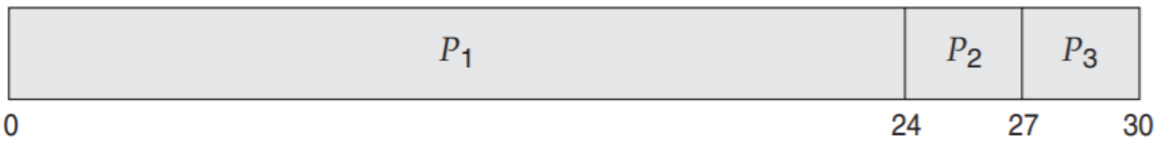
- Waiting time. The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.
- Response time. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device.

5. Any, one/two scheduling algorithm (with gantt chart, TA time, avg time, waiting time)

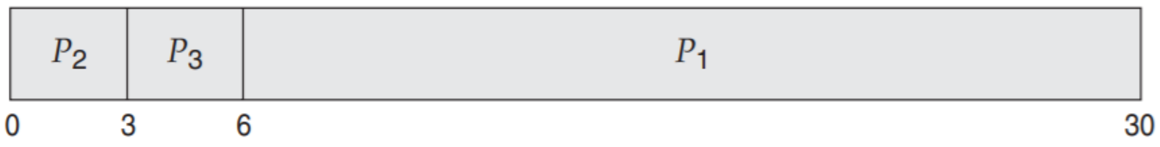
i. First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P1 , P2 , P3 The Gantt Chart for the schedule is:



Waiting time for P1 = 0; P2 = 24; P3 = 27
• Average waiting time =(P1+P2+P3)/3 = (0 + 24 + 27)/3 = 17ms
• Turn Around Time for P1=24, P2=27, P3=30 Average Turn Around time = (Total Turn Around Time)/(Total number of processes) = (24+27+30) / 3= 27ms
Suppose that the processes arrive in the order: P2, P3 , P1
• The Gantt chart for the schedule is:



• Waiting time for P1 = 6; P2 = 0; P3 = 3
• Average waiting time = (6 + 0 + 3)/3 = 3ms
Much better than previous case
• Turnaround time for P1=30, P2=3, P3=6
• Average Turnaround time = (30 + 3 + 6)/3 = 13ms
• Convoy effect - short process behind long process

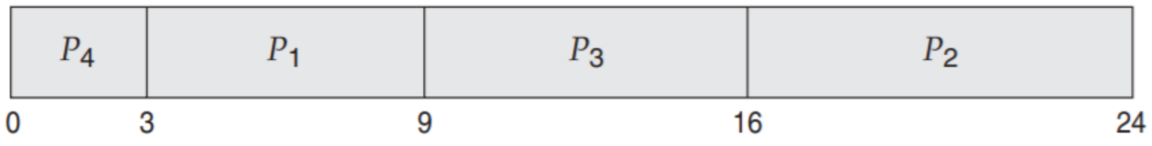
->Consider one CPU-bound and many I/O-bound processes

ii. Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



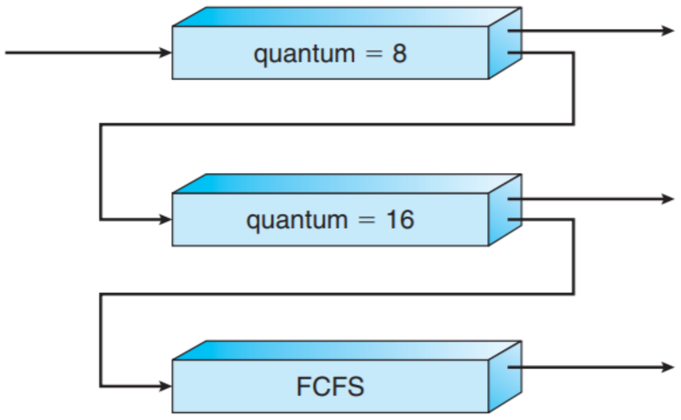
  - Waiting time for P1=3, P2=16, P3=9, P4=0;
  - Average waiting time = (3 + 16 + 9 + 0) / 4 = 7ms
  - Turnaround time for P1=9, P2=24, P3=16, P4=3
  - Average turnaround time = (9+24+16+3)/4 = 52/4= 13ms

6. Explain multilevel queue & Multilevel feedback queue with example.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

Multilevel feedback Queue:

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - Number of queues.
  - Scheduling algorithms for each queue.
  - Method used to determine when to upgrade a process.
  - Method used to determine when to demote a process.
  - Method used to determine which queue a process will enter when that process needs service.



Example of Multilevel Feedback Queue:

- Three queues:
  - Q0 – RR with time quantum 8 milliseconds
  - Q1 – RR time quantum 16 milliseconds
  - Q2 – FCFS
- Scheduling
  - A new job enters queue Q0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q1.
  - At Q1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is pre-empted and moved to queue.