# OPERATING SYSTEM

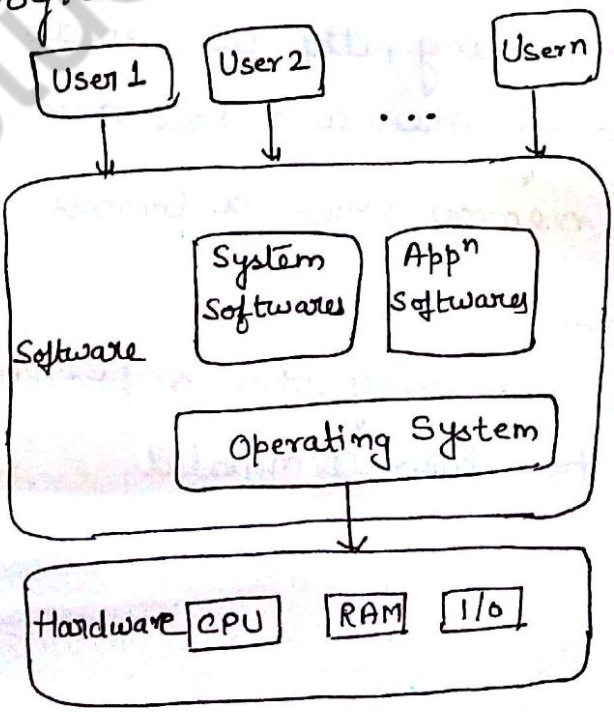
Lecture :

## UNIT-1

**INTRODUCTION :-** An Operating System (OS) is an interface between a computer user and computer hardware. An OS is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, & controlling peripheral devices like disk drives & printers.

**Some popular O.S. include :-** Linux, Windows, OS X, VMS, OS/400, AIX, z/OS, etc.

**Definition :-** An operating system is a program that acts as an interface between the user and the computer hardware & controls the execution of all kinds of programs.



Following are some important functions of an O.S.:-

→ Memory Management
→ Processor Management
→ Device Management
→ File Management
→ Security
→ Control over system performance
→ Job accounting
→ Error detecting aids
→ Coordination between other softwares & users.

## MEMORY MANAGEMENT

Memory management refers to management of Primary Memory or Main memory. Main memory is a large array of words or bytes where each word or bytes where each word or byte has its own address.

→ Keep tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
→ In multiprogramming, the OS decides which process will get memory when and how much.
→ Allocates the memory when a process requests it to do so.
→ De-allocates the memory when a process no longer needs it or has been terminated.

## PROCESSOR MANAGEMENT

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling. An OS does the following activities for processor mgmt.

→ Keeps tracks of processor & status of process. The program responsible for this task is known as traffic controller.

→ Allocates the processor (CPU) to a process.

→ De-allocates processor when a process is no longer required.

## DEVICE MANAGEMENT

An O.S. manages device communication via their respective drivers. It does the following activities for device mangement —

→ Keep tracks of all devices. Program responsible for this task is know as the I/O Controller.

→ Decides which process gets the device when and for how much time.

→ Allocates the device in the efficient way.

→ De-allocates devices.

# FILE MANAGEMENT

A file management system is normally organized into directories for easy navigation and usage. These directories may contain files & other directions.

An O.S. does the following activities for file management:-

→ Keeps track of information, location, users, status etc. The collective facilities are often known as file system.

→ Decides who gets the resources.

→ Allocates the resources.

→ De-allocates the resources.

## Other Important Activities.

→ Security :- By means of password & similar other techniques, it prevents unauthorized access to programs & data.

→ Control over system performance :- Recording delays between request for a service & response from the system.

→ Job accounting :- Keep track of time & resources used by various jobs & users.

→ Error detecting aids :- Production of dumps, traces, error messages, and other debugging & error detecting aids.

→ Coordination between other softwares & users :- Coordination & assignment of compilers, interpreters, assemblers & other software to the various users of the comp$^r$ system.

# CLASSIFICATION OF OPERATING SYSTEM

Operating systems are there from the very first comp^r generation and they keep evolving with time.

1. Batch Operating System :- The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an offline device like punch cards & submits it to the computer operator. To speed up processing, jobs with similar needs are batched together an run as a group. The programers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows-

→ Lack of interaction between the user & the job.
→ CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
→ Difficult to provide the desired priority.

2. Interactive Operating System :- An interactive operating system is one that allows the user to directly interact with the operating system whilst one or more programs are running.

There will be an user interface in place to allow this to happen. It could be a command line style of interface or it could be a graphical interface

3. Time-sharing operating systems :-

Time sharing is a technique which enables many people, located at various terminals, to a particular computer system at the same time. Time-sharing or multi tasking is a logical extention of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

⇒ The main difference between multiprogrammed Batch systems and Time sharing systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time Sharing, the objective is to minimize response time.

Advantages :-

→ Provides the advantage of quick response
→ Avoids duplication of software
→ Reduces CPU idle time

Disadvantages :-

→ Problem of reliability
→ Question of security & integrity of user programs & data
→ Problem of data comm$^n$.

4. Real time Systems:-

A real time system is defined as the data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an ~~system~~ input & display of required updated information is termed as the response time. So in this method, the response time is very less as compared to the online processing.

2 types of real time systems.

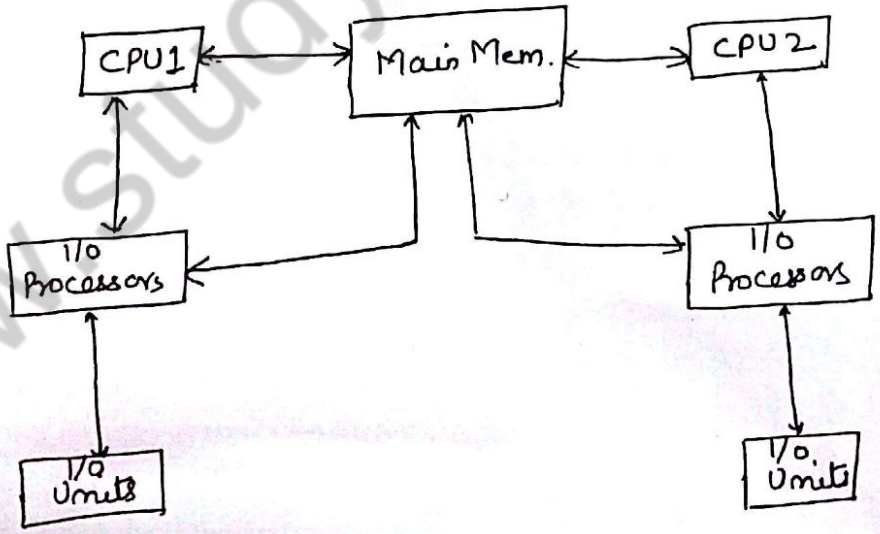
(a) Hard real time systems:- Hard r.t.s. guarantee that critical tasks complete on time. In hard rts, secondary storage is limited or missing & the data is stored in ROM. In these systems, virtual memory is almost never found.

(b) Soft real time systems:- Soft real time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration & planetary rovers, etc.

5. Multiprocessor Systems :-

Multiprocessing is the use of two or more CPUs within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability of a system to support more than one processor to allocate tasks between them. These multiple processors are in close comm$^{n}$ sharing the computer bus, memory & other peripheral devices. These systems are referred to as tightly coupled systems.

These types of systems are used when very high speed is required to process a large volume of data. These systems are generally used in environment like satellite control, weather forecasting etc. The basic organization of multiprocessing system –



6. Multiuser Systems :-

A multi user operating system is an O.S. that allows multiple users on different computers or terminals to access a single system with one OS on it.

Timesharing systems are multiuser systems. Most batch processing systems for mainframe computers may also be considered "multi-user", to avoid leaving the CPU idle while it waits for I/O operations to complete.

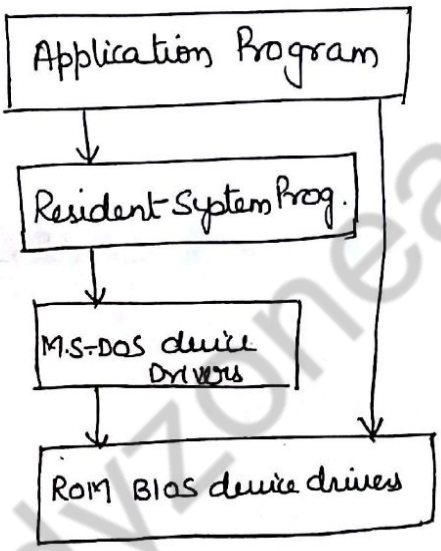
7. Multiprocess Systems :- Multiprocesses O.S. refers to

8. Multithreaded System :- Multithreading is the ability of a program or an O.S. process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programing running in the computer.

# OPERATING SYSTEM STRUCTURE

The design of an operating system architecture traditionally follows the separation of concerns principle. This principle suggest structuring the O.S. into relatively independent parts that provide simple individual features, thus keeping the complexity of the design manageable.

1. Simple Structure :-




→ In MS DOS, applications may bypass the O.S.
→ O.S. such as MS-DOS and the original UNIX did not have well defined structures
→ There was no CPU Execution Mode (user & kernal), and so errors in applications could cause the whole system to crash.

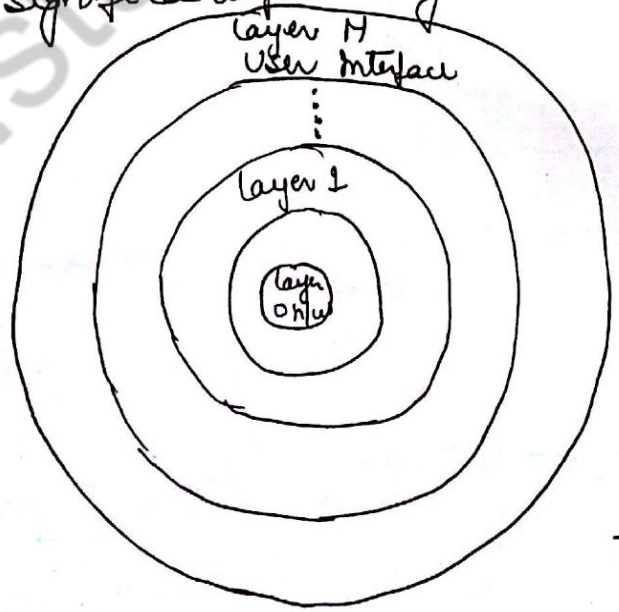
## 2. Layered Approach

→ Another approach is to break the OS into a number layers, each of which rests on the layer below it, and relies solely on the services provided by the lower layer.

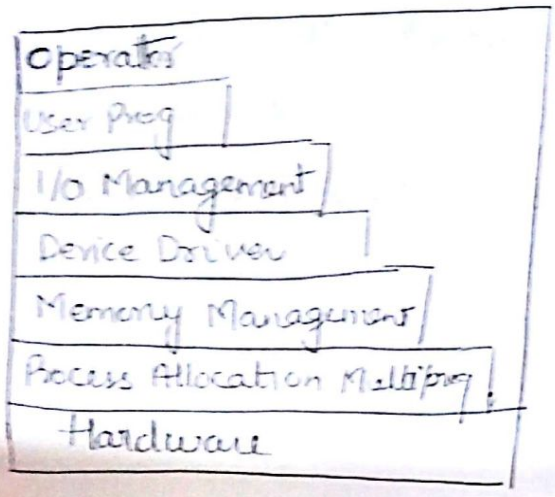
→ This approach allows each layer to be developed & debugged independently, with the assumption that all lower layers have already been debugged & are trusted to deliver proper services.

→ The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer, & so many

→ Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.




A layered O.S.




Layered Architecture

## 3. Microkernals

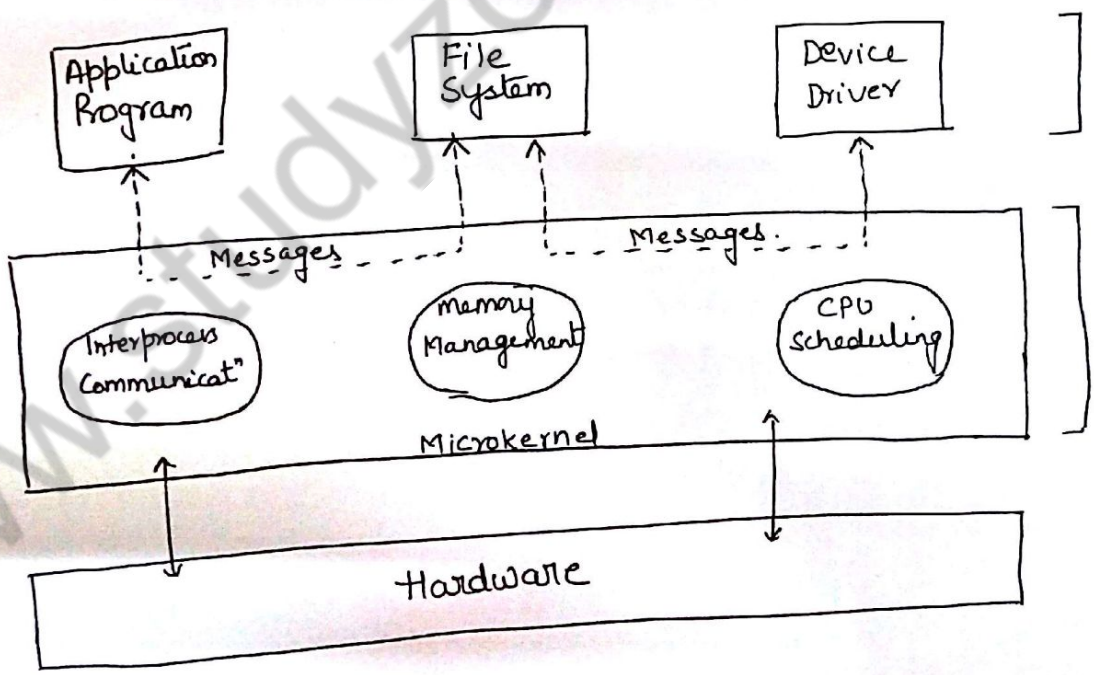
The basic idea behind micro kernals is to remove all non-essential services from the kernal, and implement them as system applications instead, thereby making the kernal as small and efficient as possible.

→ Most micro kernels provide basic process & memory management & message passing between other services & not much more.

→ Security & protection can be enhanced, as most services as performed in user mode not kernal mode.

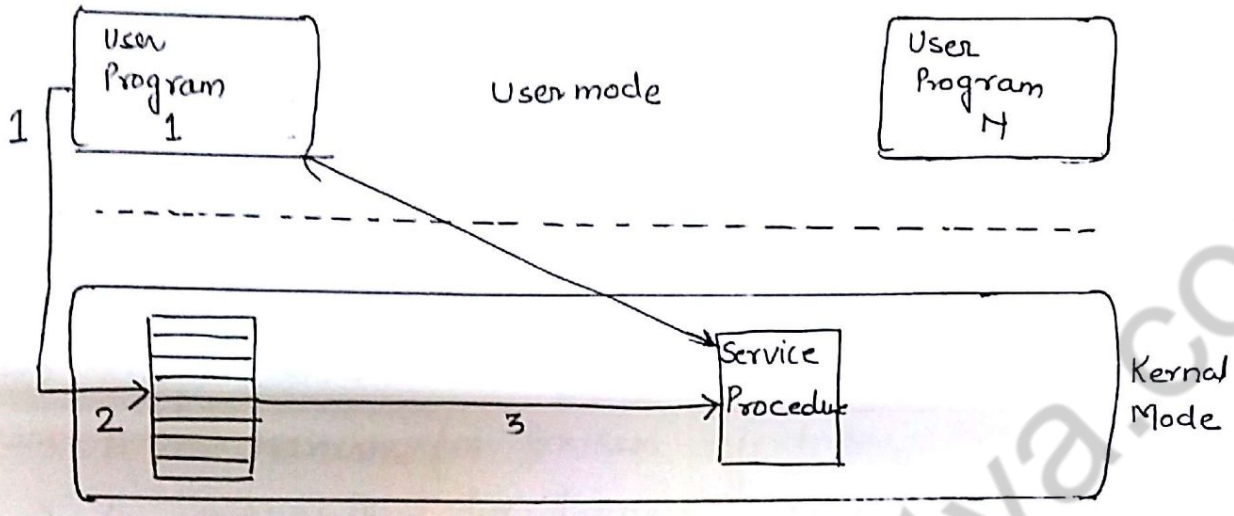
→ System expansion can also be easier as it involves adding more system applications, not rebuilding a new kernal.

Eg. of micro kernal. QNX, a real time O.S. for embedded system.




Architecture of a typical microkernal

4. Monolithic Architecture of OS




Monolithic Structure of OS

It is the oldest architecture used for developing OS. Operating system resides on kernal for anyone to execute. System call is involved i.e. switching from user mode to kernal mode and transfer control to O.S. [Event 1]

Many CPUs have 2 modes have 2 modes,

Kernal Mode for O.S. in which all instructions are allowed & User mode for user program in which I/O devices and certain other instruction are not allowed.. Then OS then examine the parameter of the call to determine which system call is to be carried out [Event 2]. Next, the OS index's into a table that contains procedure that carries out system call [Event 3] Finally, it is called when the work has been completed & the system call is finished, control is given back to the user mode [Event 4].

# SYSTEM COMPONENTS

Even though, not all systems have the same structure many modern OS share the same goal of supporting the following types of system components.

## 1. Process Management :-

→ Creation & deletion of user & system processes.
→ Suspension & resumption of processes.
→ A mechanism for process synchronization
→ A mechanism for process communication.
→ A mechanism for deadlock handling.

## 2. Main Memory Management

→ Keep track of which part of memory are currently being used & by whom
→ Decide which process are loaded into memory when memory space becomes available.
→ Allocate & deallocate memory space as needed.

## 3. File Management

→ The creation & deletion of files.
→ The creation & deletion of directories
→ The support of primitives for manipulating files & directories.
→ The mapping of files onto secondary storage.
→ The backup of files on stable storage media.

4. I/O System Management

I/O subsystem hides the peculiarities of specific hardware devices from the user. Only the device drivers know the complexities of the specific device to whom it is assigned.

5. Secondary Storage Management :-

→ Managing the free space available.
→ Allocation of storage space when new files are written.
→ Scheduling the requests for memory access.

6. Networking

A distributed systems is a collection of processors that do not share memory, peripheral devices, or a clock. The processors communicate with one another through comm^n lines called n/w. The comm^n - n/w design must consider routing & connection strategies, and the problems of contention & security.

7. Security :-

Security refers to the mechanism for controlling the access of programs, processes or users to the resources defined by a computer system.

# OPERATING SYSTEM - SERVICES

An O.S. provides services to both the users & to the programs.

→ It provides programs an environment to execute

→ It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system.

→ Program Execution
→ I/O operations
→ File system manipulation
→ Communication
→ Error Detection
→ Resource Allocation
→ Protection.

## 1. Program Execution

→ Loads a program into memory
→ Executes the program.
→ Handles program's execution
→ Provides a mechanism for process synchronizat$^{n}$
→ Provides a mechanism for process comm$^{n}$
→ Provides mechanism for deadlock handling.

2. I/O Operation

An I/O subsystem comprises of I/O devices & their corresponding drivers softwares. Drivers hide the peculiarities of specific hardware devices from the users.

An OS manages comm$^n$ b/w user & device drivers.

→ I/O operation means read or write operation with any file or any specific I/O device.

→ O.S. provides the access to the required I/O device when required.

3. File System Manipulation

→ Program needs to read a file or write a file.

→ The O.S. gives the permission to the program for operation on file.

→ Permission varies from read-only, read-write, denied and so on.

→ O.S. provides an interface to the user to create/delete files.

→ O.S. provides an interface to the user to create/delete directories.

→ O.S. provides an interface to create the backup of file system.

3. Communication

→ Two process often require data to be transferred between them.

→ Both the processes can be on one computer or on different computers, but are connected through a computer network.

→ Comm$^n$ may be implemented by two methods, either by shared or by message passing.

→ Comm$^n$.

4. Error Handling

Error may occur in CPU, in I/O devices or in memory h/w.

→ The OS constantly checks for possible errors.

→ The OS takes an appropriate action to ensure correct & consistent computing.

5. Resource Management

→ The O.S. manages all kinds of resources using schedulers.

→ CPU scheduling algorithms are used for better utilization of CPU.

6. Protection

→ The O.S. ensures that all access to system resources is controlled.

→ The O.S. ensures that external I/O devices are protected from unvalid access attempts

→ The O.S. provides authentication features authentication for each user by means of passwords.

# RE-ENTRANT KERNEL

A reentrant kernel enables processes to give away the CPU while in kernel mode, not hindering other processes from also entering kernal mode. Thus in re-entrant kernal several processes may be executing in Kernal Mode at the same time.

A typical use is IO wait. The process want to read a file. It calls a kernel fun" for this side. Inside the kernel fun", the disk controller is asked for the data. Getting the data will take some time & the fun' is blocked during that time. With a reentrant Kernel, the scheduler will assign CPU to another process until an interrupt from the disk controller indicates that the data is available & the earlier thread can be resumed. This process can still access IO, like user input.

## How to acheive re-entrancy ?

First way of providing re-entrancy is to write functions so that they modify only local variables and do not alter global data structures. Such

functions are referred to as "Re-entrant functions". A Kernel that implements 'reentrant routines" are reffered to as "Re-entrant Kernels"

Other than using 're-entrant functions", a Kernel can also use "Locking" methods to ensure that only one process can execute a non-re-entrant function at a time.

## Difference between Non-reentrant & Re-entrant Kernel

In Non-reentrant kernal mode, every single process acts on its own set of memory locations & thus cannot interfere with the others.

On a 're-entrant kernel', when a h/w interrupt occurs, it is able to suspend the current running process, even if the process is in Kernel Mode. This improves the throughput of the device controllers that issue interrupts. What exactly happens is this: When a device issues an interrupt, it waits for the CPU to acknowledge. If the Kernel answers quickly, the device controller will be to perform other tasks while the CPU handles the interrupt.

# Concurrent Process. UNIT-2

## Process Concept :-

→ A process is an executing program, including the current values of the program counter, registers and variables

→ Difference between a process and a program is a group of instructions whereas the process is the activity.

→ Process can be described :

- I/O Bound Process - spends more time doing I/O then computation.
- CPU Bound Process :- spends more time doing computation.

## Cooperating Processes

→ Processes executing concurrently in the O.S. may be either independent processes or cooperating processes.

→ A process is independent if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent.

→ A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly any process that shares data with other processes is a cooperating process.

→ Advantages of process cooperation :-

(i) Information sharing

(ii) Computation speedup

(iii) Modularity (dividing the system functions into separate processes or threads.

(iv) Convenience (an individual user may work on many tasks at the same time.

## Process Synchronization :

→ Concurrent access to shared data may result in data inconsistency.

→ Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

→ Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer & is decremented by the consumer after it consumes the buffer.

### Critical Section Problem :

→ Consider a system consisting n processes { P0, P1 ... Pn }

→ Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file & so on.

→ The important feature of the system is that, when one process is executing in its C.S., no other process is to be allowed to execute in its C.S.

→ That is, no two processes are executing in their C.S.s at the same time (mutually exclusive)

→ The section of code implementing this request is the entry section. The C.S. may be followed by an exit section. The remaining code is the remainder section.

```
do {
    Entry Section
        Critical Section
    exit Section
        remainder section
} while(true);
```

## Solution to C.S. Problem

A. Mutual Exclusion :- If process $P_i$ is executing in its critical section, then no other processes can be executed in their critical sections.

B. Progress :- If no process is executing in its critical section and there exists some processes that wish to enter their C.S., then the selection of the processes that will enter the C.S. next cannot be postponed indefinitely.

C. Bounded Waiting :- A bound must exists on the number of times that other processes are allowed to enter their C.S, after a process has madey made a request to enter its C.S. and before that request is granted.

→ Assume that each process executes at a non zero speed.

→ No assumption concerning relative speed of the N processes.

Finally, we need four conditions to hold to have a good solution:

1. No two processes may be simultaneously inside their C.S.
2. No assumptions may be made about speeds or the number of processors.
3. No processes running outside its C.S. may block other processes.
4. No process should have to wait forever to enter its C.S.

Peterson's solution :-

```
do {
    flag[i] = TRUE
    turn = j;
    while (flag[j] && turn == j);

    critical section

    flag[i] = FALSE;

    remainder section
} while (TRUE);
```

## Dekker's Solution :-

```
Do {
flag[0] := true
  while flag[1] = true {
   if turn ≠ 0 {
    flag[0] := false
   while turn ≠ 0 {
   }
   flag[0] := true.
   }
 }
// critical section
....
turn := 1
flag[0] := false
// remainder section
} while (TRUE);
```

## Peterson's Solution

```
do {
  flag[i] = TRUE
   turn = j;
  while (flag[j] && turn == j);

  critical section

  flag[i] = FALSE

   remainder section

  } while (TRUE);
```

## Test & Set Operation :

```
do {
   waiting[i] = TRUE;
      key = TRUE;
   while (waiting[i] && key)
      key = TestandSet(&lock);
   waiting[i] = FALSE;
   // critical section
   j = (i+1) % n;
   while ((j != i) && !waiting[j])
      j = (j+1) % n
   if (j == i)
      lock = FALSE;
   else
      waiting[j] = FALSE;
   // remainder section
} while (TRUE);
```

## Semaphores :-

- Semaphores provide solution to critical section problem.
- Definition :- Semaphore S is an integer variable, apart from initialization is accessed only through two automatic operation wait() & signal().

| wait(S)<br>{<br>While(S<0)<br>;<br>S--; // no operation<br>} | Signal(S)<br>{<br>S++;<br>} |
|---|---|

## Mutex Implementation with semaphore with no busy time :-

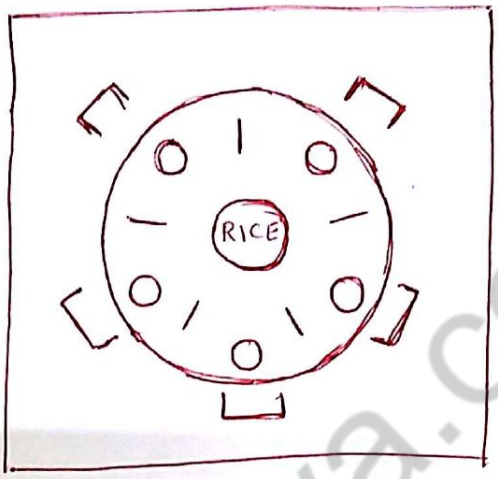
Must guarantee that not two processes can execute wait() & signal() on the same semaphore at the same time.

```
Do {
 wait(mutex)
   critical section
 signal(mutex)
    remainder section
} while(1);
```

| wait(S) {<br>value --;<br>if(value<0) {<br>add this process to waiting queue<br>block(); }<br>} | Signal(S) {<br>value++;<br>if(value <= 0) {<br>remove a process P from the waiting queue<br>wakeup(P);<br>} } |
|---|---|

## Classical Problem in Concurrency :-

### Dining Philosopher Problem :-




Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by 5 chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks.

Shared Data :-

- Bowl of rice (data set)
- Semaphore chopstick [5] initialized to 1.

The structure of Philosopher (i) :-

```
while (true) {
    wait (chopstick [i]);
    wait (chopstick [(i+1) % 5]);

    // eat
    signal (chopstick [i]);
    signal (chopstick [(i+1) % 5]);

    // think
}.
```

## Sleeping Barber's Problem

A Barber shop has a single room with one barber chair and n customer chairs. Customers (i.e. independent processes) enter the waiting room one at a time if empty chairs are available, otherwise they go elsewhere. Each time the barber (i.e. independent process) finishes a haircut the customer leaves to go elsewhere, & among the waiting customers, the one who has been waiting the longest (i.e. a FIFO waiting queue) move to the barber's chair to get a hair cut.

If the barber discovers the waiting room is empty, (s)he falls asleep in the waiting room. An arriving customer finding the barber asleep wakes the barber (i.e. a rendezvous takes place) and has a haircut; otherwise the arriving customer waits.

Using semaphores & shared variables only (but not critical region statements or monitors), give two algorithms, one for the barber, and another for an arbitrary customer, that implement the sleeping barber problem. Explain your algorithms, and explicitly specify any assumptions you make about the model.

# Interprocess Communication

## Cooperating Processes :-

⇒ An independent process is one that cannot affect or be affected by the execution of other processes.

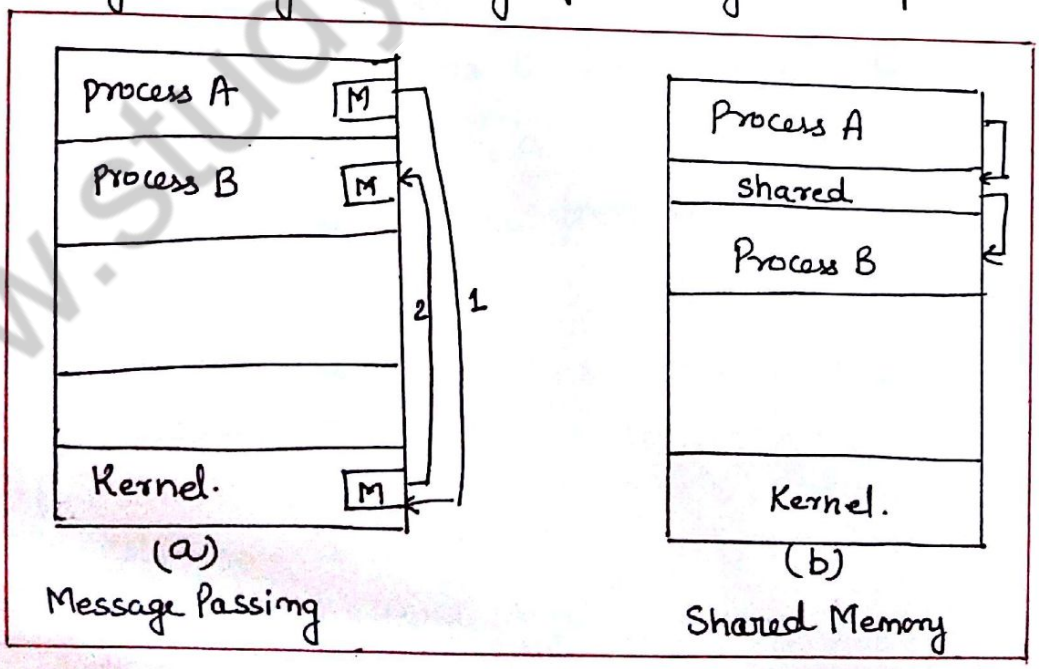
⇒ A cooperating process can affect or be affected by the execution of another process in the system.

### Advantages of cooperating processes

→ Information sharing (of the same piece of data)
→ Computation speed up (break a task into smaller subtasks)
→ Modularity (dividing up the system functions)
→ Convenience (to do multiple tasks simultaneously)

### Two fundamental models of interprocess communication

→ Shared Memory (a region of memory is shared)
→ Message Passing (exchange of messages b/w processes).



## Shared Memory Systems

→ Shared Memory requires communicating processes to establish a region of shared memory.

→ Information is exchanged by reading & writing data in the shared memory.

→ A common paradigm for cooperating processes is the producer-consumer problem.

→ A producer process produces inf$^{n}$ that is consumed by a consumer process.

- unbounded-buffer places no practical limit on the size of the buffer.
- bounded-buffer assumes that there is a fixed buffer size.

## Message Passing Systems :-

→ Mechanism to allow process to communicate & to synchronize their actions.

→ No address space needs to be shared; this is particularly useful in a distributed processing environment (eg. a chat program).

→ Message passing facility provides two operations :

- send (message) - msg size can be fixed or variable
- receive (message)

→ If P and Q wish to communicate, they need do :

- establish a communicat$^{n}$ link b/w them.
- exchange msg via send/receive.

## Logical Implementation of comm^n link.

→ Direct or indirect communicat^n
→ synchronous or asynchronous communication
→ Automatic or explicit buffering

### Direct Communication

→ Process must name each other explicitly:
- send (P, message) – send a msg to process P
- receive (Q, message) – receive a msg from process P

→ Properties of communication link
- Link are established automatically between every pair of processes that want to communicate.
- A link is associated with exactly one pair of communicating processes.
- Between each pair there exists exactly one link.
- The link may be unidirectional, but is usually bi-directional.

→ Disadvantages:-
- Limited modularity of the resulting process definitions
- Hard-coding of identifiers are less desirable than indirection techniques.

## Indirect Communication :-

Messages are directed and received from mailboxes (also referred to as ports)

→ Each mailbox has a unique id.

→ Processes can communicate only if they share a mailbox
- send (A, message) - send message to mailbox A
- receive (A, message) - receive msg from mailbox A.

→ Properties of comm^n link
- Link is established between a pair of processes only if both have shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of processes, there may be many different links, with each link corresponding to one mailbox.

→ For shared mailbox, messages are received based on the following methods:
- Allow a link to be established with atmost two processes.
- Allow at most one process at a time to execute a receive() operation.
- Allow the system to select arbitrary which process will receive the message. (eg round robin tech).

## Synchronization :-

→ Message passing may be either blocking or non-blocking.

→ Blocking is considered synchronous.

↪ • Blocking send has the sender block until the message is received.

° Blocking receive has the receiver block until a message is available.

→ Non blocking is considered asynchronous.

° Non-blocking send has the sender send the message & continue.

° Non-blocking receives has the receiver receive a valid message or null.

→ When the send() and receive() are blocking, we have a rendezvous btw the sender & receiver.

## Buffering

whether communication is direct or indirect, msg exchanged by communicating processes resides in a temporary queue. These queues can be implemented in 3 ways.

1 Zero capacity : the queue has a maximum length of zero. Sender must block until the recipient receives the msg.

2. Bounded capacity : the queue has the finite length of n. Sender must wait if queue is full.

3. Unbounded capacity : the queue length is unlimited Sender never blocks.

## Process Creation:-

Parent process create children processes, which in turn create other processes, forming a tree of processes.

### Resource sharing:-

1. Parent and children share all resources.
2. Children share subset of parent's resources.
3. Parent and child share no resources.

### Execution:-

1. Parent and children execute concurrently.
2. Parent waits until children terminate.

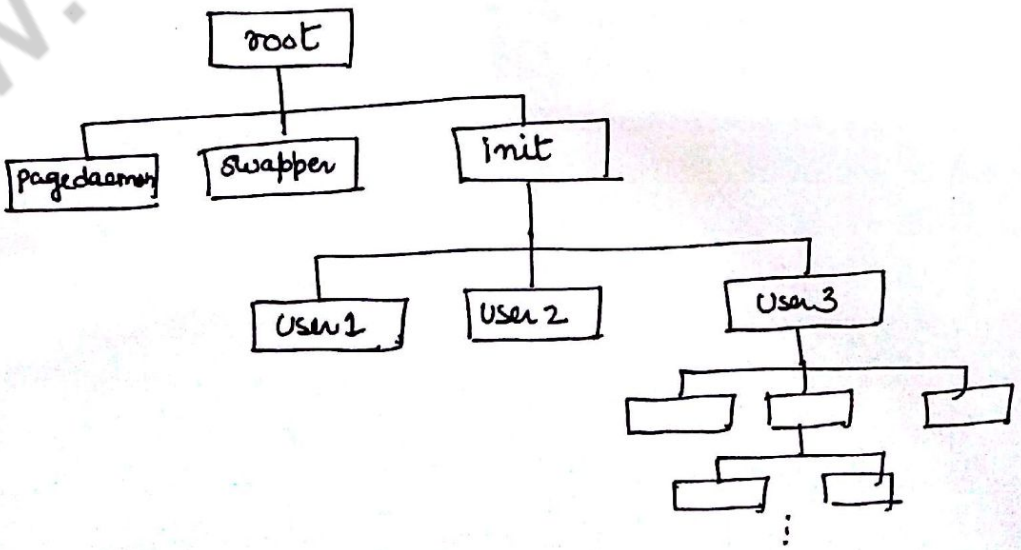
### Address Space

1. Child duplicate of parent.
2. Child has a program loaded to it.

### UNIX examples:-

1. "fork" system call creates new processes.
2. "exec" system call used after a fork to replace the process' memory space with a new program.

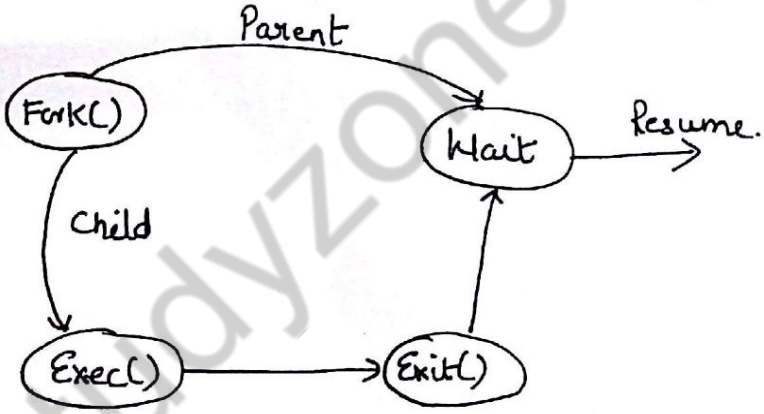
### Process tree on a Unix System:-



## Process Termination

Process executes last statement and asks the operating system to decide it (exit).

1. Output data from child to parent (via wait).
2. Process resources are de allocated by operating system. Parent may terminate execution of children processes (abort).
3. Child has exceeded allocated resources.
4. Task assigned to child is no longer required.
5. Parent is existing.
6. Operating system does not allow child to continue if its parent terminates.

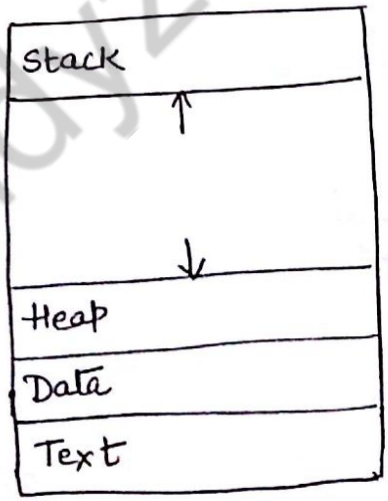


# PROCESS CONCEPT UNIT-3

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 entry which represents the basic unit of work to be implemented in the system".

To put in simple words - when a program is loaded into the memory and it becomes a process, it can be divided into four sections - stack, heap, text and data. The following image shows a simplified layout of a process inside main memory.




Stack:- The process stack contains the temperory data such as method/function parameters, return address and local variables.

Heap:- This is dynamically allocated memory to a process during its runtime.

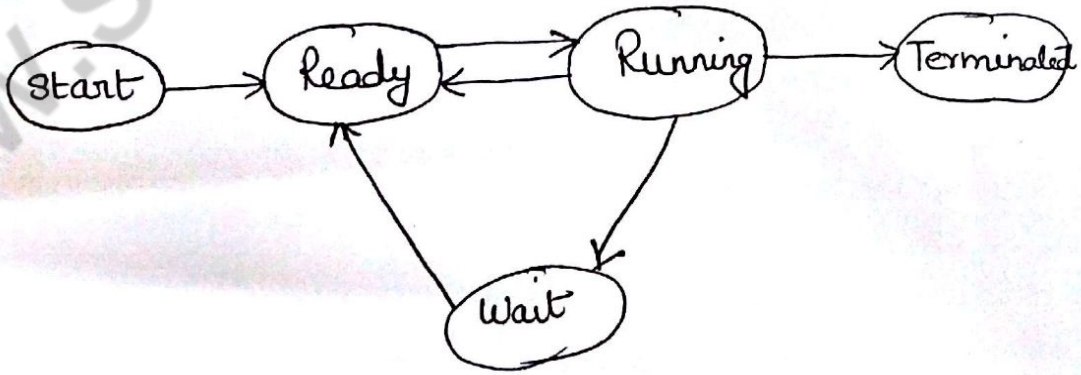
Text:- This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

Data:- This section contains the global & static variables.

Program:- A computer program is a collection of instructions that perform a specific task when executed by a computer. "A process is a dynamic instance of a program".

A part of a computer program that performs a well defined task is known as an algorithm. A collection of computer programs, libraries & related data are referred to as a software.

Process Life Cycle :-




Process States

When a process executes, it passes through diff states. These stages may differ in different OS.

Start :- This is the initial state when a process is first started / created.

Ready :- The process is waiting to be assigned to the processor. Ready processes are waiting to have the processor allocated to them by the O.S. so that they can run. Processes may come to this state after "Start" state or while running it by but interrupt by the scheduler to assign CPU to some other processes.

Running :- Once the process has beenassigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Waiting :-

Processes move into waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

Terminated :- Once the process finishes its exe", or it is terminated by the O.S., it is moved to the terminated state where it waits to be removed from main memory.

# Process Control Block (PCB)

A Process Control Block is a data structure maintained by the operating System for every process. The PCB is identified by an integer process id (PID). A PCB keeps all information needed to keep track of a process as listed below in table :-

1. Process State :- The current state of the process ie. whether it is ready, running, waiting or whatever.
2. Process Privileges :- This is req. to allow/disallow access to system resources.
3. Process ID :- Unique identification for each of the process in the O.S.
4. Pointer :- A pointer to parent process.
5. Program Counter :- is a pointer to the address of the next instruction to be executed for this process.
6. CPU registers :- Various CPU registers where process needs to be stored for exec" for running state.
7. CPU scheduling information :- Process priority & other scheduling

| Process ID |
|---|
| State |
| Pointer |
| Priority |
| Program Counter |
| CPU registers |
| I/O Information |
| Accounting Informat" |
| : |
| etc : |

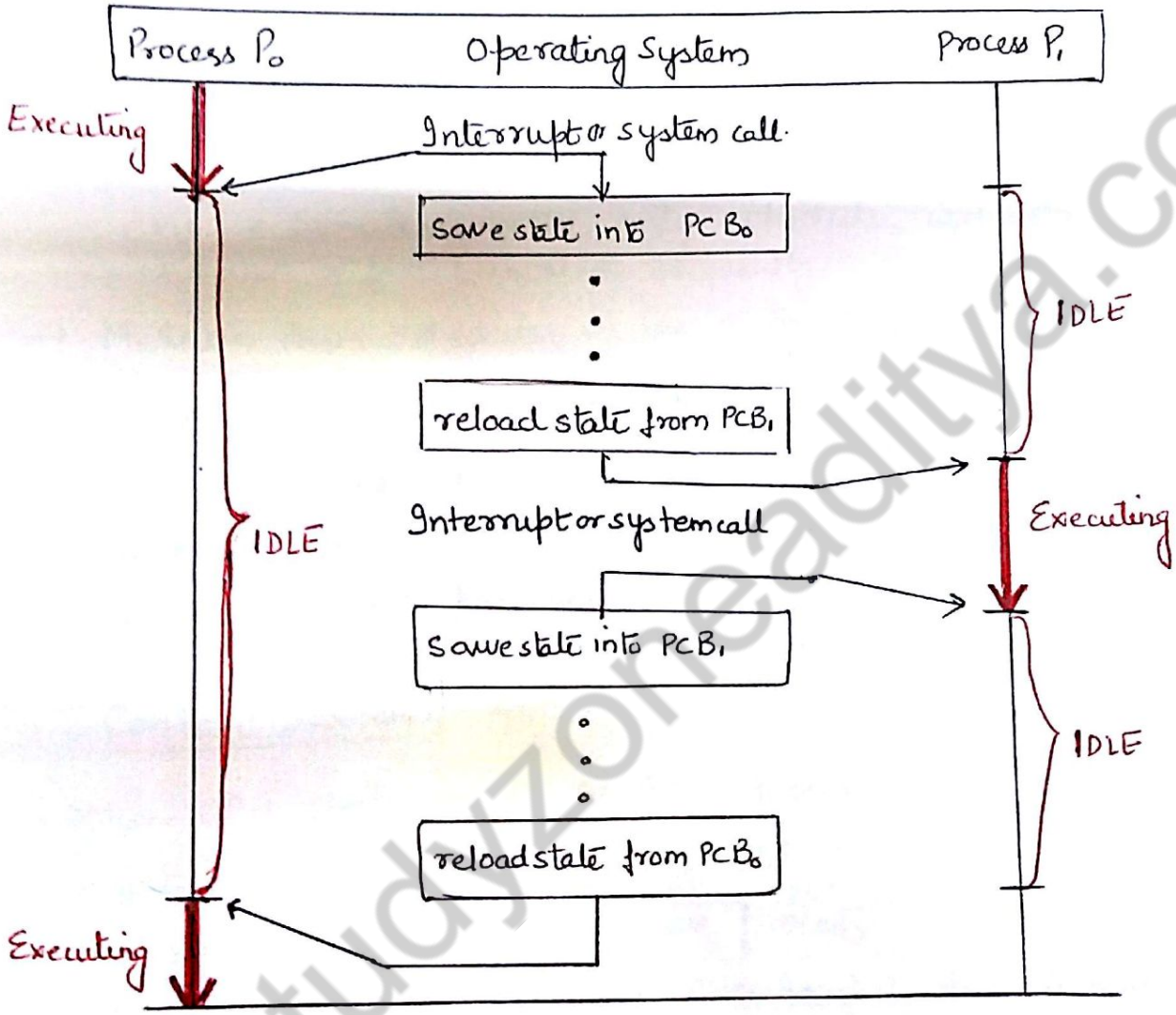
Simplified Diag of PCB

8. Memory Management information :- This includes the information of page table, memory limits, segment table depending on memory used by the OS.

9. Accounting information :- This includes the amount of CPU used for process execution, time limits, execution ID etc.

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

# CPU Switch From Process to Process.



## Schedulers

→ Long term scheduler :- or (job scheduler) - selects which processes should be brought into the ready queue.
→ must be slow (invoked less frequently).

→ Short term scheduler :- or (CPU scheduler) - selects which process should be executed next and allocates CPU. • Sometimes the only scheduler in the system.
→ must be fast (invoked frequently).

→ Medium term scheduler :- is a part of swapping. It removes the process from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in charge of handling the swapped out-processes.

## Context Switch

→ 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.

## Process Scheduling

The process scheduling is the activity 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 a particular strategy.

Process scheduling is an essential part of a Multiprogramming O.S. Such OS systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using multiplexing.

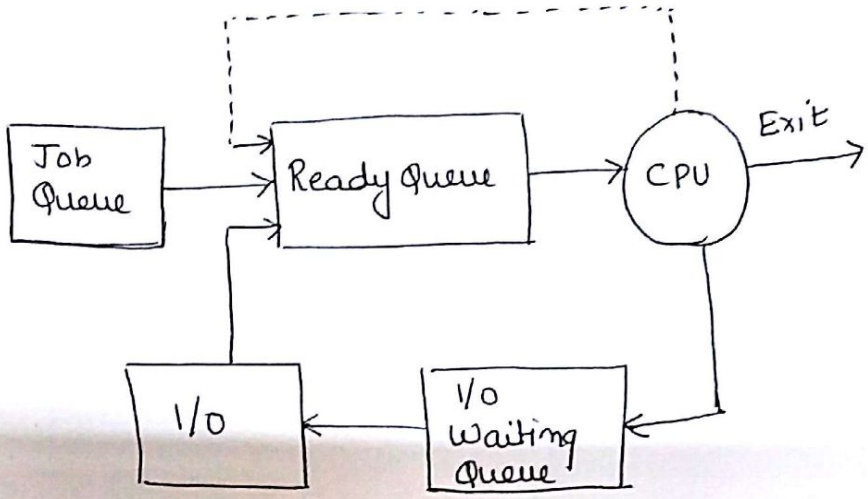
## Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue & move to its new state.

→ **Job Queue**:- This queue keeps all the processes in the system.

→ **Ready Queue**:- This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

→ **Device queues**:- The process




## Process Scheduling Queues.

| S.No | Long Term Scheduler | Short-Term Scheduler | Medium-Term Scheduler |
|---|---|---|---|
| 1. | It is a job scheduler | It is a CPU scheduler | It is a process swapping scheduler |
| 2. | Speed is lesser than S.T. scheduler | speed is fastest among other two | speed is in b/w both short & long term scheduler |
| 3. | It controls the degree of multi-programming | It provides lesser control over degree of multiprogramming | It reduces the degree of multiprog-ramming. |
| 4. | It is almost absent or minimal in time sharing system | It is also minimal in time sharing system | It is a part of time sharing systems |
| 5. | It selects processes from pool & loads them into memory for execution | It select those processes which are ready to execute | It can re-introduce the process into memory and execut$^n$ can be continued. |

## Scheduling Criteria

Different CPU-scheduling algorithms have different properties and may favour one class of process or another.

Many criteria have been suggested for comparing CPU scheduling algorithms. The characteristics used for comparison can be make a substantial difference in the determination of best algorithm.

**CPU Utilization :-** We want to keep the CPU as busy as possible CPU utilization may range 0 to 100 percent. In real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

**Throughput :-** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes completed per time unit called throughput. For long processes, this rate may be 1 process per hour for short transactions, throughput might be 10 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 turn around time.

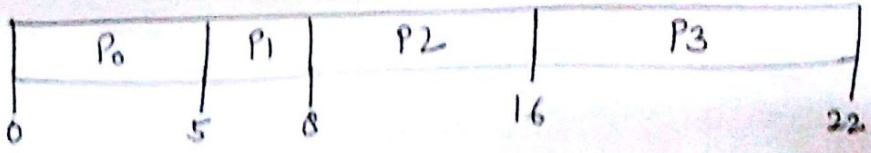
## Process Scheduling Algorithms

A process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. These algorithms are either preemptive or non-preemptive. Non preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

### 1. First Come First Serve (FCFS)

→ Jobs are executed on first come, first serve basis.
→ It is a non-preemptive, scheduling algo.
→ Easy to understand and implement.
→ Its implementation is based on FIFO queue.
→ Poor in performance as average wait time is high.

| Process | Arrival time | Execute Time | Service Time |
|---|---|---|---|
| $P_0$ | 0 | 5 | 0 |
| $P_1$ | 1 | 3 | 5 |
| $P_2$ | 2 | 8 | 8 |
| $P_3$ | 3 | 6 | 16 |



Wait time for each process is as follows:-

| Process | Wait Time : Service time - Arrival time |
|---|---|
| $P_0$ | 0 - 0 = 0 |
| $P_1$ | 5 - 1 = 4 |
| $P_2$ | 8 - 2 = 6 |
| $P_3$ | 16 - 3 = 13 |

Average W.T = (0+4+6+13)/4 = 5.75

## 2. Shortest Job Next:- (SJN).

→ This is also known as shortest job first or SJF

→ This is a non-preemptive scheduling algo.

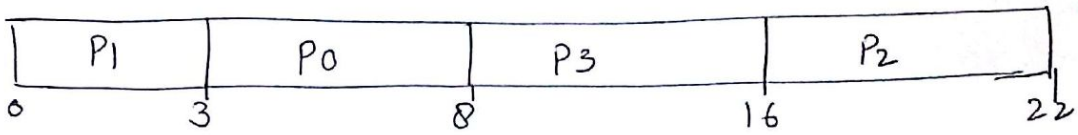
→ Best approach to minimize waiting time.

→ Easy to implement in Batch systems where required CPU time is known in advance.

→ Impossible to implement in interactive systems where required CPU time is not known.

→ The processor should know in advance how much time process will take.

| Process | Arrival time | Execut^n Time | Service time |
|---|---|---|---|
| $P_0$ | 0 | 5 | 3 |
| $P_1$ | 1 | 3 | 0 |
| $P_2$ | 2 | 8 | 16 |
| $P_3$ | 3 | 6 | 8 |



Wait time for each process –

$P_0$ – 3-0 = 3

$P_1$ – 0-0 = 0

$P_2$ – 16-2 = 14

$P_3$ – 8-3 = 5

Average W.T. = (3+0+14+5) /4 = 5.50

## 3. Priority Based Scheduling :-

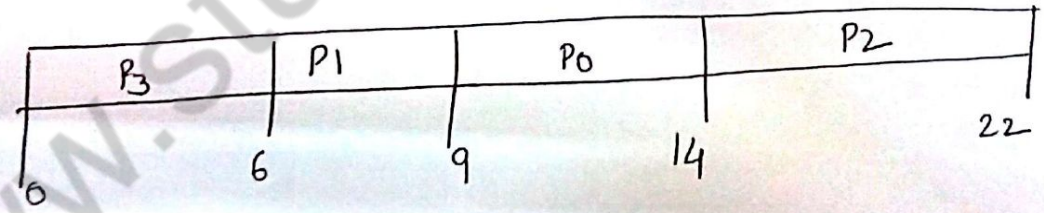
→ Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithm in batch systems.

→ Each process is assigned a priority. Process with highest priority is to be executed first and so on.

→ Processes with same priority are executed on FCFS basis.

→ Priority can be decided based on memory requirements, time requirements or any other resource requirement

| Process | Arrival Time | Execut$^n$ Time | Priority | Service time |
|---|---|---|---|---|
| $P_0$ | 0 | 5 | 1 | 9 |
| $P_1$ | 1 | 3 | 2 | 6 |
| $P_2$ | 2 | 8 | 1 | 14 |
| $P_3$ | 3 | 6 | 2 | 0 |




Waiting time :- $P_0$ – 9-0 = 9

$P_1$ – 6-1 = 5

$P_2$ – 14-2 = 12

$P_3$ – 6-0 = 6

A.W.T. = (9+5+12+0)/4 = 6.5

4. Shortest Remaining Time : (SRT).

→ Shortest Remaining time (SRT) is the preemptive version of the SJN algorithm.

→ The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.

→ Impossible to implement in interactive systems where required CPU time is not known.

→ It is often used in batch environments where short jobs need to give preference.

5. Round Robin Scheduling :-

→ Round robin is the preemptive process scheduling algorithm.

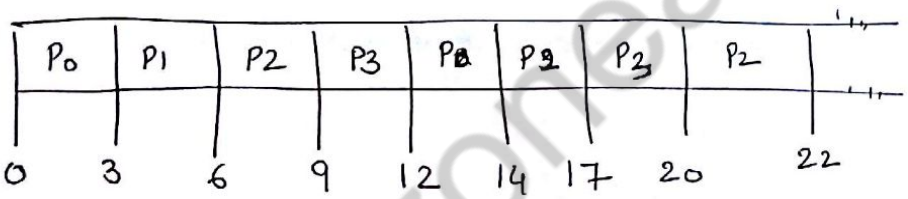
→ Each process is provided a fix time to execute, it is called a quantum.

→ Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

→ Context switching is used to save states of preempted processes.

Quantum = 3.

–




Waiting time :- $P_0$ – (0-0)+(12-3) = 9

$P_1$ – (3-1) = 2

$P_2$ – (6-2)+(14-9)+(20-17) = 12

$P_3$ – (9-3)+(17-12) = 11

Average W.T. = (9+2+12+11)/04 = 8.5.

## 6 Multiple-Level Queues Scheduling

Multiple level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

→ Multiple queues are maintained for processes with common characteristics

→ Each queue can have its own scheduling algorithms.

→ Priorities are assigned to each queue.

For eg. CPU bound jobs can be scheduled in one queue and all I/O bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

## Thread :-

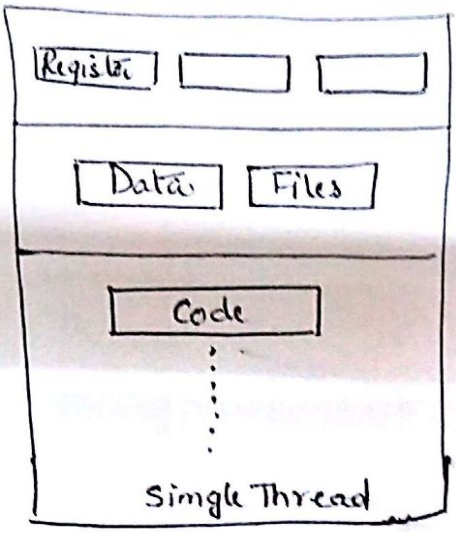
A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment & open files. When one thread alters a code segment memory item, all other threads see that.

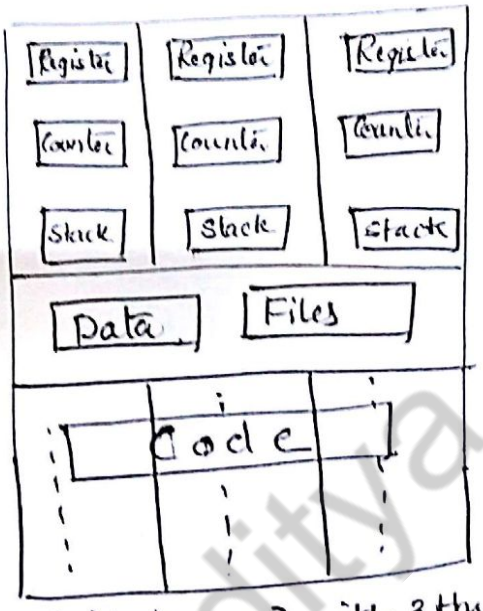
A thread is also called a light weight process. Thread provide a way to improve application performance through parallelism. Thread represents a s/w approach to improving performance of O.S. by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing n/w servers & web server. They also provide a suitable foundation for parallel execution of application on shared memory multiprocessors.

The following figure shows the working of a single-threaded & a multithreaded process.




Single Process P with Single thread




Single-process P with 3 threads.

| | PROCESS | THREAD |
|---|---|---|
| 1. | Process is heavyweight & resource intensive. | - Thread is light wt, taking lesser resources than a process. |
| 2. | Process switching needs interact" with OS. | - Thread switching does not need to interact with OS. |
| 3. | In multiprocessing environments each process executes the same code but has its own memory & file resources. | - All threads can share same set of open files, child processes. |
| 4. | If one process is blocked, then another process can execute until the first process is unblocked | - While one thread is blocked & waiting, a second thread in the same task can run. |
| 5. | Multiple process without using threads use more resources | - Multiple threaded processes use fewer resources. |
| 6. | In multiple process each process operates independently of the others. | - One thread can read, write or change another thread's data. |

## Advantages of threads:

→ Threads minimize the context switching time.
→ Use of threads provide concurrency within a process.
→ Efficient comm"
→ It is more economical to create & context switch threads.
→ Threads allow utilization of multiprocessor architectures to a greater scale & efficiency.

## Types of threads :-

Threads are implemented in following 2 ways:-

→ User level threads :- User managed threads
→ Kernel level threads :- OS manage threads acting on kernel, an O.S. core.

# Deadlocks

**System Model:-** A system consists of finite number of resources to be distributed among a number of competing processes.

→ Resources types :- $R_1, R_2 \cdots R_m$

→ Each resource type $R_i$ has 1 or more instance.

→ Each process utilizes a resource as follows:-

- → Request
- → Use
- → Release.

## The Deadlock Problem

→ A deadlock consist of a set of blocked processes, each holding a resource and waiting to acquire a resource held by another process in the set.

Eg.
- A system has 2 disk drives.
- $P_1$ and $P_2$ each hold one disk drive & each needs the one

Eg
- Semaphores A and B, initialized to 1.

| $P_0$ | $P_1$ |
|---|---|
| wait(A); | wait(B); |
| wait(B); | wait(A); |

# Deadlocks

**System Model:-** A system consists of finite number of resources to be distributed among a number of competing processes.

→ Resources types :- $R_1, R_2 \cdots R_m$

→ Each resource type $R_i$ has 1 or more instance.

→ Each process utilizes a resource as follows:-

- → Request
- → Use
- → Release.

## The Deadlock Problem

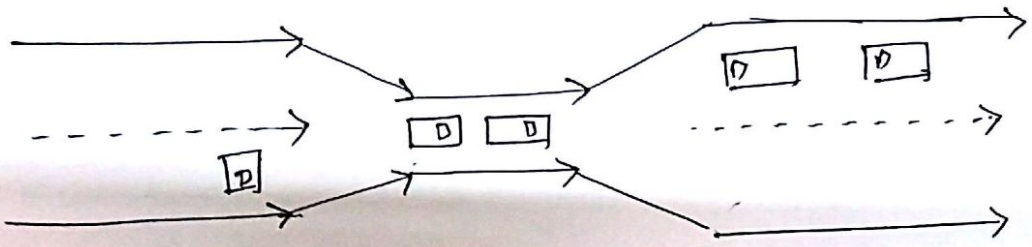
→ A deadlock consist of a set of blocked processes, each holding a resource and waiting to acquire a resource held by another process in the set.

Eg.
- A system has 2 disk drives.
- $P_1$ and $P_2$ each hold one disk drive & each needs the one

Eg
- Semaphores A and B, initialized to 1.

| $P_0$ | $P_1$ |
|---|---|
| wait(A); | wait(B); |
| wait(B); | wait(A); |

## Bridge Crossing Example



→ Traffic only in one direction
→ The resource is one one lane bridge
→ If a deadlock occurs, it can be resolved if one car backs up (preempt resources & rollback)
→ Several cars may have to be backed up if a deadlock occurs.
→ Starvation is possible.

## Deadlock Characterization

Deadlock can arise if 4 conditions hold simultaneously

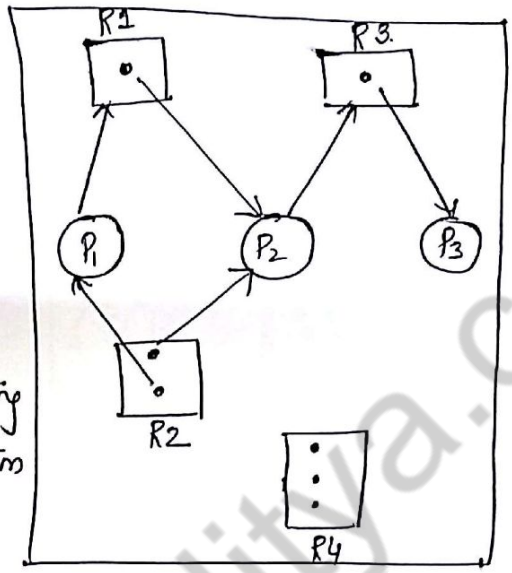
1. Mutual Exclusion:- only one process at a time can use the resource.
2. Hold & wait :- a process holding atleast one resource is waiting to acquire additional resources held by other processes.
3. No preemption :- a resource can be released only voluntarily by the process holding it after that process has completed its task.
4. Circular Wait :- there exists a set $\{P_0, P_1, \ldots, P_0\}$ 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 $P_2 \ldots P_{n-1}$ is waiting for a resource that is held by $P_n$ & $P_n$ is waiting for a resource that is held by $P_0$.

# Resource Allocation Graph

A set of vertices V and a set of edges E.

→ V is partitioned into 2 types:

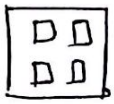
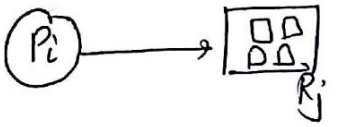
- P = {P1, P2, P3 ... Pn}, the set consisting of all the processes in the system.
- R = {R1 R2 .. Rm} the set consisting of all the resources in the system

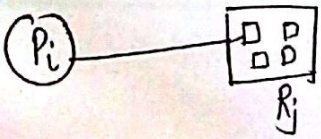



→ Request edge - directed graph.

$P_i \rightarrow R_j$

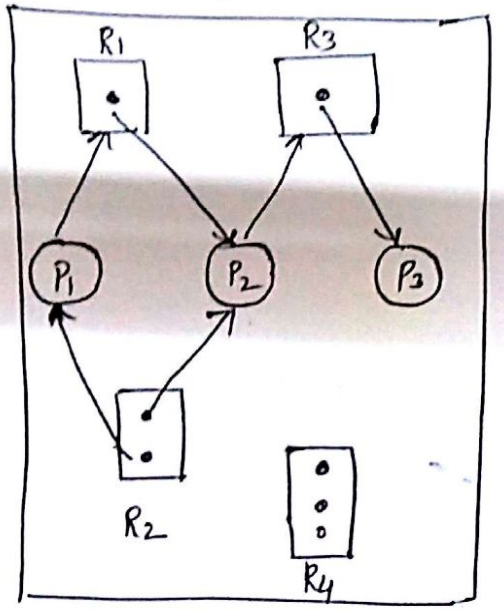
→ Assignment edge - directed edge $R_j \rightarrow P_i$.

- Process — ○
- Resource type with instances 
- $P_i$ request instance of $R_j$ 

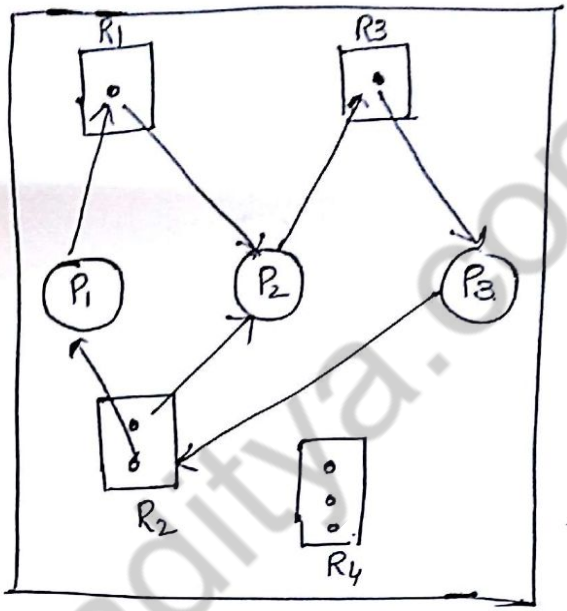
- $P_i$ is holding an instance of $R_j$ 

## Resource Allocation Graph with a Deadlock

Before $P_3$ requested an instance of $R_2$

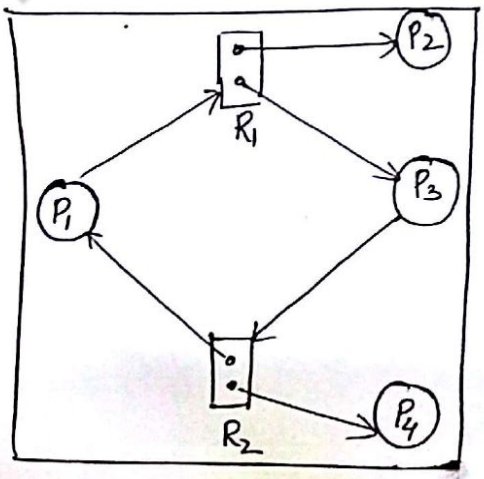



After $P_3$ requested an instance of $R_2$




## Graph with a Cycle but not a deadlock

Process $P_4$ may release its instance of resource type $R_2$. That resource can then be allocated to $P_3$, thereby breaking the cycle.




→ If a resource allocation graph contains no cycles ⇒ no deadlock.

→ If a resource allocation graph contains a cycle and if only one instance exists per resource type ⇒ deadlock.

→ If a resource allocation graph contains a cycle and if several instances exists per resource type ⇒ possibility of deadlock.

## Methods of handling Deadlocks

→ Prevention :- Ensure that the system will never enter a deadlock state.

→ Avoidance :- Ensure that the system will never enter an unsafe state.

→ Detection :- Allow the system to enter a deadlock state and then recover.

→ Do nothing :- Ignore the problem and let the user or system administrator respond to the problem used by most O.S, including Windows and UNIX.

## Deadlock Prevention :-

To prevent deadlock, we can restrain the ways that a request can be made :-

→ Mutual Exclusion :- The mutual-exclusion condition must hold for non-sharable resources.

→ Hold and Wait :- we must guarantee that whenever a process requests a resource, it does not hold any other resources.

- Require a process to request and be allocated all its resources before it begins execution, or allow a process to request resources only when the process has none.
- Result :- Low resource utilization, starvation possible

→ 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.
- Preempted resources are added to the list of resources for which the process is waiting.
- A process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

→ Circular Wait :- impose a total ordering of all resources types, and require that each process requests resources in an increasing order of enumeration.

For eg.

F(tape drive) = 1
F(disk drive) = 5
F(printer) = 12

## Deadlock Avoidance

Requires that the system has some additional 'a priori' information available:-

→ Simplest and most useful model requires that each processes 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.

→ A resource allocation state is defined by the number of available & allocated resources, & the maximum demands of the processes.

### Safe State:-

→ When a process requests an available resource, the system must decide if immediate allocation leaves the system in a safe state.

→ A system is in a safe state only if there exists a safe sequence.

→ A sequence of processes $< P_1, P_2, \ldots, P_n >$ is a safe sequence for the current allocation state if, for each $P_i$, the resource requests that $P_i$ can still make, can be satisfied by currently available resources plus resources held by all $P_j$, with $j < i$.

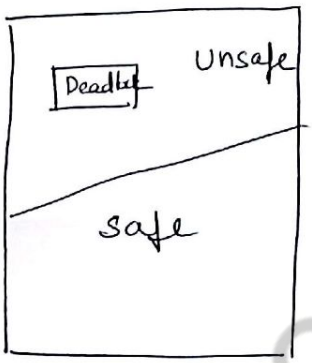
That is:-

- If the $P_i$ resource needs are not immediately available, then $P_i$ can wait until all $P_j$ have finished
- When $P_j$ is finished, $P_i$ can obtain needed resources, execute, return allocated resources & terminate
- When $P_i$ terminates, $P_{i+1}$ can obtain its needed resources & so on.

→ If a system is in safe state : no deadlock.

→ If a system is in unsafe state ⇒ possibility of deadlock.

→ Avoidance ⇒ ensures that a system will never enter an unsafe state.




## Avoidance Algorithms

→ For a single instance of a resource type, use a resource allocation graph.

→ For multiple instances of a resource type, use the banker's algorithm.

## Resource Allocation Graph scheme :-

① → Introduce a new kind of edge called a claim edge.

② → Claim edge $P_i \dashrightarrow R_j$ indicates that process $P_j$ may request resource $R_j$ which is represented by a dashed line.

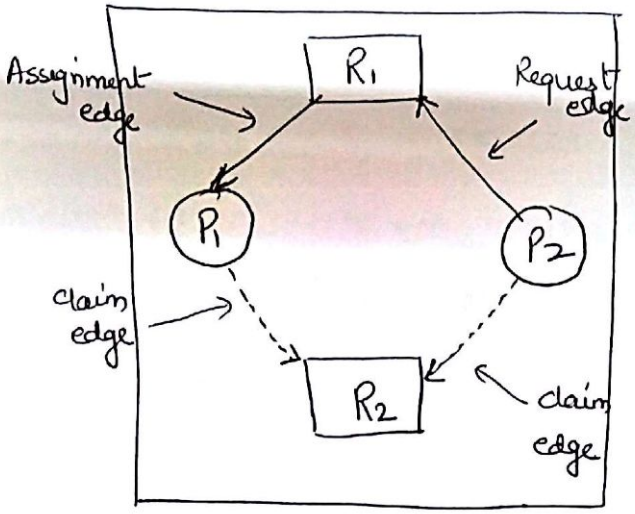
③ → A claim edge converts to an ~~assignment~~ request edge when the resource is allocated to the process.

⑤ → When a resource is released by a process, an assignment edge reconverts to a claim edge.

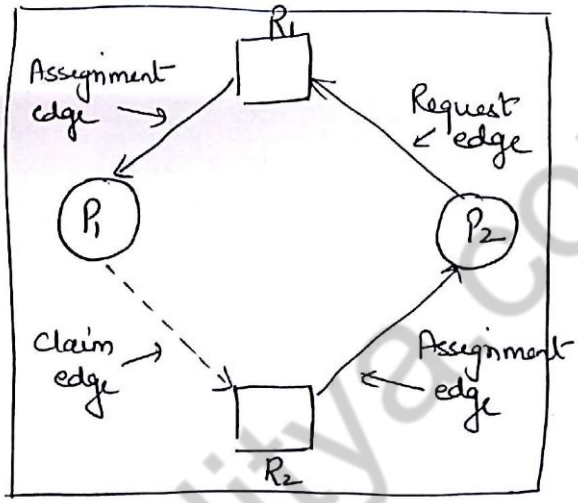
⑥ → Resources must be claimed a priori in the system.

④ → A request edge converts to an assignment edge when the resource is allocated to the process.

## Resource allocation graph with claim edges




## Unsafe state in Resource allocation graph.




## Resource Allocation Graph Algorithm

→ Suppose that process $P_i$ requests resource $R_j$

→ The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocat^n graph.

## Banker's Algorithm

→ Used when there exists multiple instances of a resource type.

→ Each process must a priori claim maximum use.

→ When a process requests a resource, it may have to wait.

→ When a process gets all its resources, it must return them in a finite amount of time.

## Data Structures for the Banker's algorithm

Let n = number of processes, and m = number of resource types.

→ Available :- Vector of length m. If available[j]=k, there are k instances of resource type $R_j$ available.

→ Max :- n×m matrix. If Max[i,j]=k, then processes $P_i$ may request at most k instance of resource type $R_j$.

→ Allocation :- n×m matrix. If Allocation[i,j]=k then $P_i$ is currently allocated k instances of $R_j$.

→ Need :- n×m matrix. If Need[i,j]=k, then $P_i$ may need k more instances of $R_j$ to complete its task.

$$Need[i,j] = Max[i,j] - Allocation[i,j].$$

## Deadlock Detection

## Deadlock Detection

→ For deadlock detection, the system must provide
- An algorithm that examines the state of the system to detect whether a deadlock has occcured.
- An algorithm to recover from the deadlock.

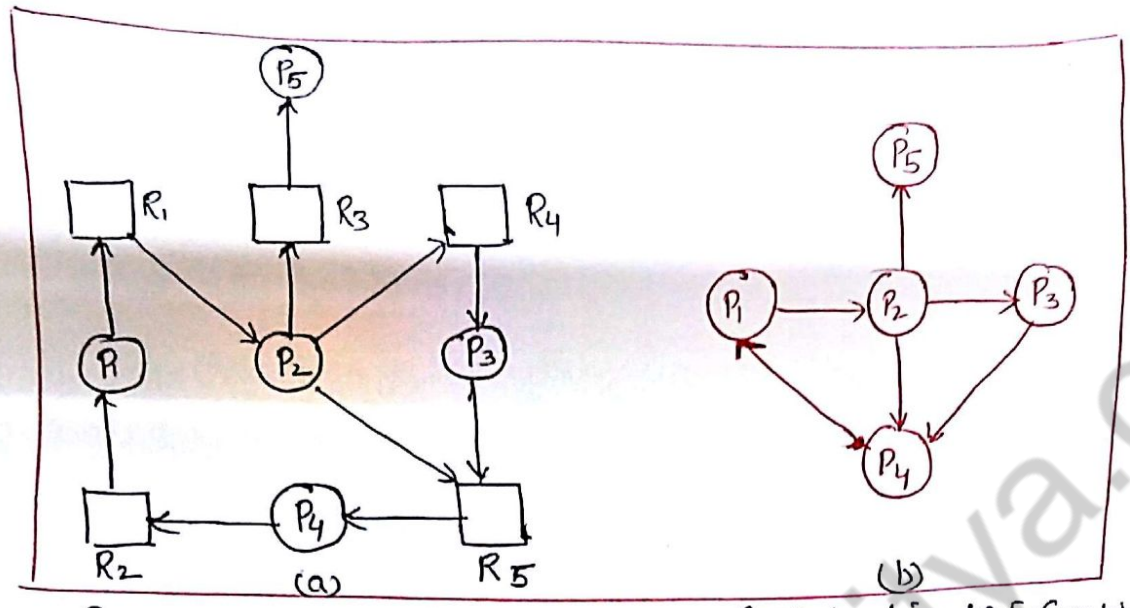
→ A detection-and-recovery scheme requires various kinds of overhead.
- Run-time costs of maintaining necessary information and executing the detection algorithm.
- Potential losses inherent in recovering from a deadlock.

### Single instances of each resource type

→ Requires the creation & maintenance of a wait for graph.
- Consists of a variant of the resource allocat$^n$ graph.
- The graph is obtained by removing the resource nodes from a resource allocation graph & collasping the appropriate edges.
- Consequently all nodes are processes.
- $P_i \rightarrow P_j$ if $P_i$ is waiting for $P_j$.

→ Periodically invoke an algorithm that searches for a cycle in the graph.
- If there is a cycle, there exists a deadlock.
- An algorithm to detect a cycle in a graph requires an order of $n^2$ operations, where n is the number of vertices in the graph.

# Resource allocation Graph & Wait for Graph




Resource allocation Graph

Corresponding W-F-Graph

## Multiple instances for a Resource Type

Required data structures

→ **Available** :- A vector of length m indicates the number of available resources of each type.

→ **Allocation** :- An nxm matrix defines the number of resources of each type currently allocated to each process.

→ **Request** :- A nxm matrix indicates the current request of each process. If Request[ij] = k, then process Pi is requesting k more instances of resource type Rj.

## Detection Algorithm Usage

→ When, and how often, to invoke the detection algo depends on:
- How often is a deadlock likely to occur?
- How many processes will be affected by deadlock when it happens?

→ If the detection algorithm is invoked arbitrary, there may be many cycles in the resource graph & so we would not be able to tell which one of the many deadlocked processes "caused" the deadlock.

→ If the detection algorithm is invoked for every resource request, such an action will be incur a considerable overhead in compulation time

→ If the detection algorithm is invoked for every resource request, such an action will incurs a considerable overehead in compulation.

→ A less expensive alternate is to invoke the algorithm when CPU utilization drops below 40% for eg.
- This is based on the observation that a deadlock eventually cripples system throughput and causes CPU utilization to drop.

## Recovery From Deadlock

Two approaches

- Process termination
- Resource preemption

## Process Termination

→ Abort all deadlocked process.

- This approach will break the deadlock, but at great expense.

→ Abort one process at a time untill the deadlock cycle is eliminated.

- This approach incurs considerable overhead since, after each process is aborted, a deadlock-detection algorithm must be re-invoked to determine whether any processes are still deadlocked.

→ Many factors may affect which process is chosen for termination.

- What is the priority of the process?
- How long has the process run so far & how much longer will the process need to run before completing its task.
- How many and what type of resources has the process used?
- How many resources does the process need in order to finish its tasks?
- How many processes will need to be terminated?
- Is the process interactive or batch?

## Resource Preemption

→ With this approach, we successfully preempt some resources from processes & give these resources to other processes until the deadlock cycle is broken.

→ When preemption is required to deal with deadlocks, then three issues need to be addressed :-

- Selecting a victim :- which resources and which processes are to be preempted.
- Rollback :- If we preempt a resource from a process, what should be done with that process?
- Starvation :- How do ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?

UNIT-4

## Memory Management

→ Program must be brought into memory & placed within a process for it to be run.

→ Input queue or job queue - collection of processes on the disk that are waiting to be brought into memory to run the program.

→ User programs go through several steps before being run.

### Binding of Instructions & Data to Memory

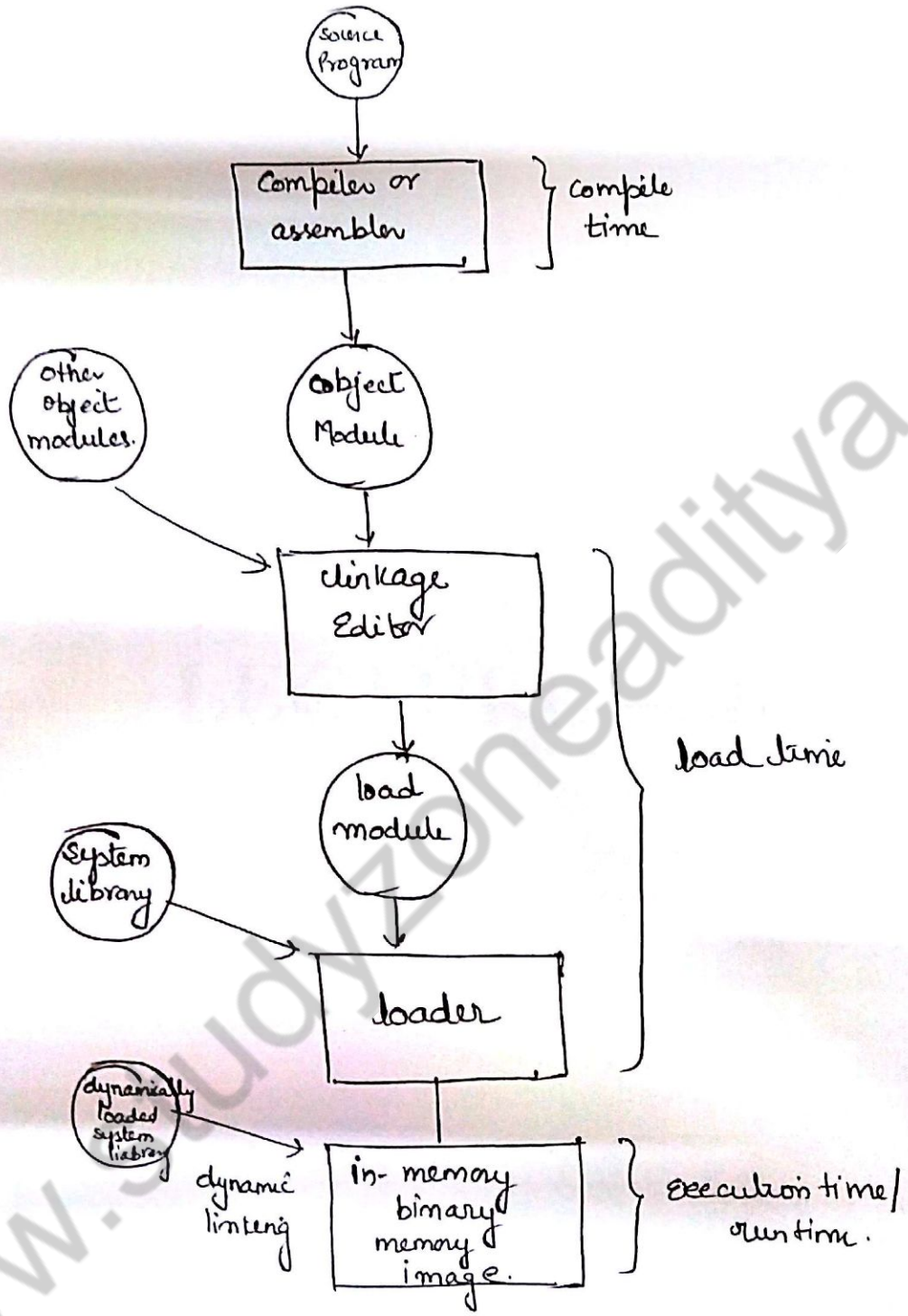
Address binding of instructions & data to memory addresses can happen in 3 different stages :-

→ Compile time :- If memory location known apriori, 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 h/w support for address maps. (eg base limit registers & limit registers).

# Multistep Processing of a User Program.



## Static vs Dynamic Loading

The choice b/w static & Dynamic loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical address.

If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided & rest of the work will be done at the time of execution.

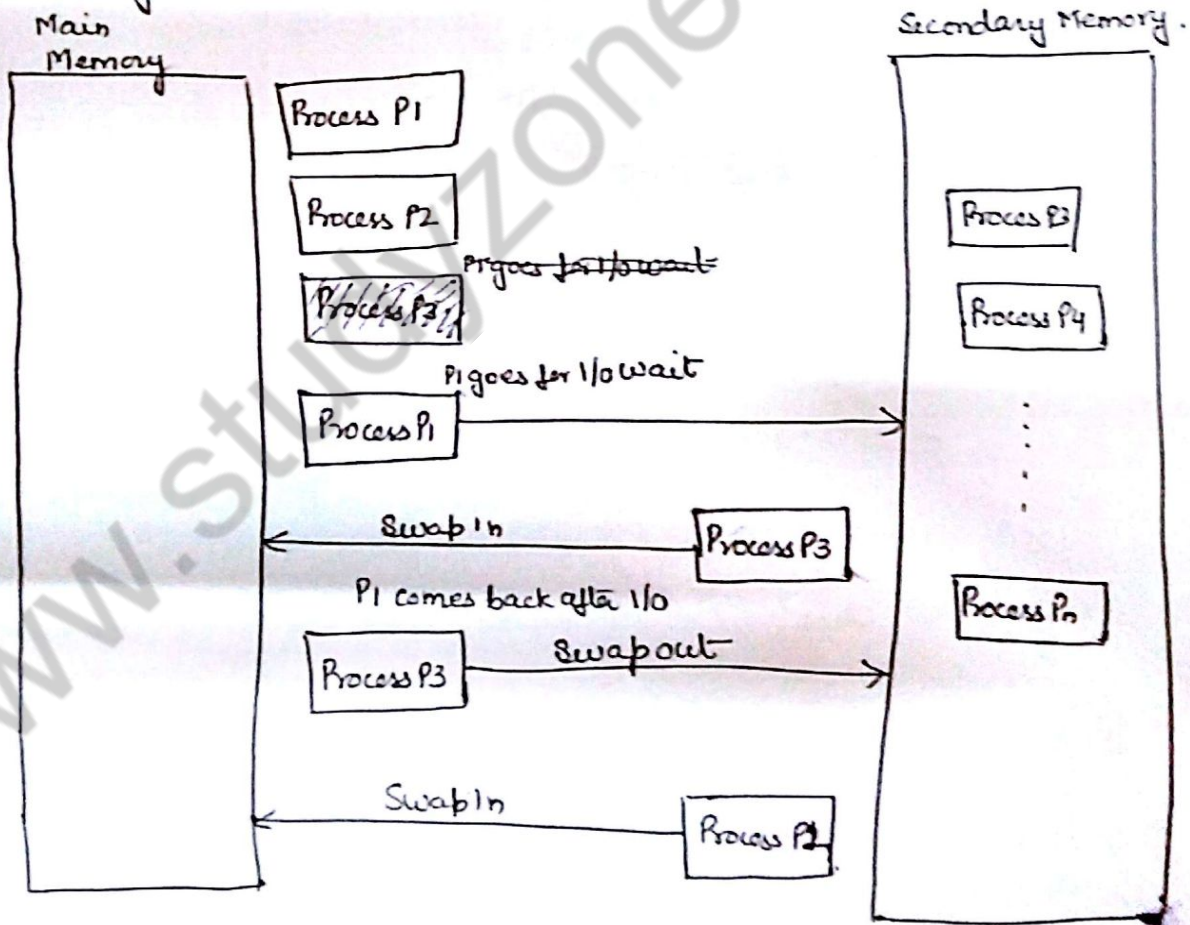
At the time of loading, with static loading, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using dynamic loading, dynamic routine of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

## Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel & that's the reason "Swapping is also known as a technique for memory compaction".



## Single Partition Allocation

In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing O.S. code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical address. Each logical address must be less than the limit register.

## Multiple partition

In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

# Fragmentation

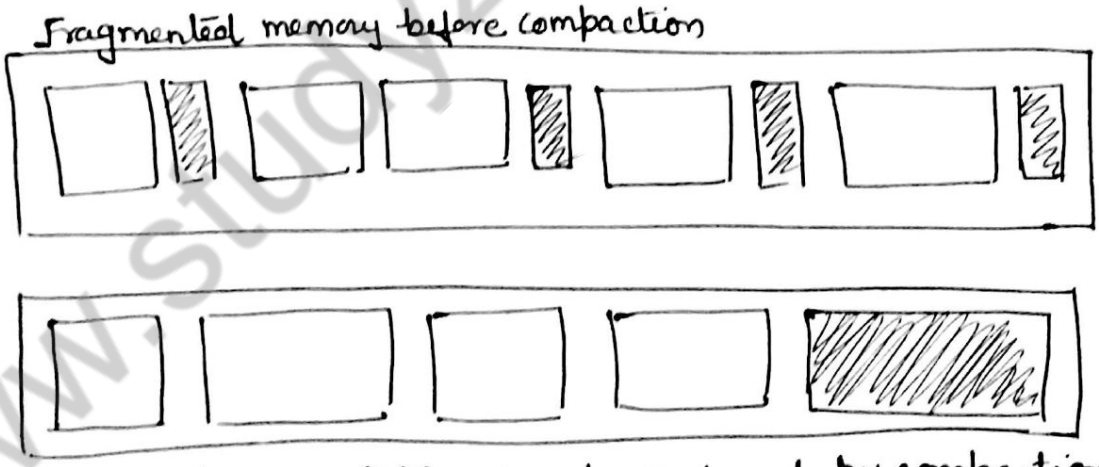
As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks remains unused.

This problem is known as Fragmentation.

Fragmentation is of 2 types.

External Fragmentation :- Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

Internal Fragmentation :- Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.




External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

Paging:

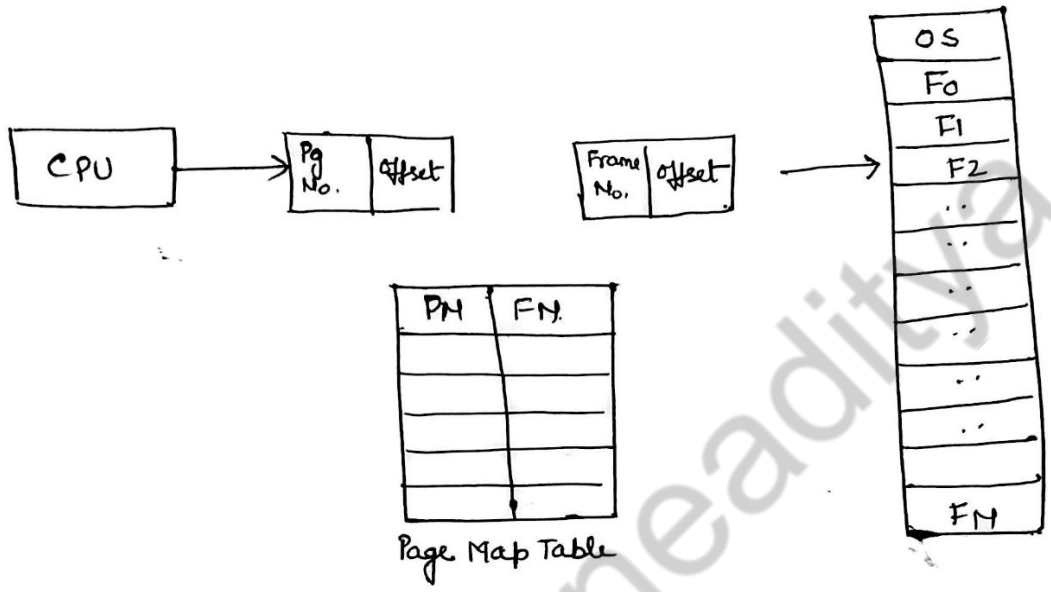
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of same size called pages (size is power of 2, b/w 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed size blocks of (physical) memory called frames. and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

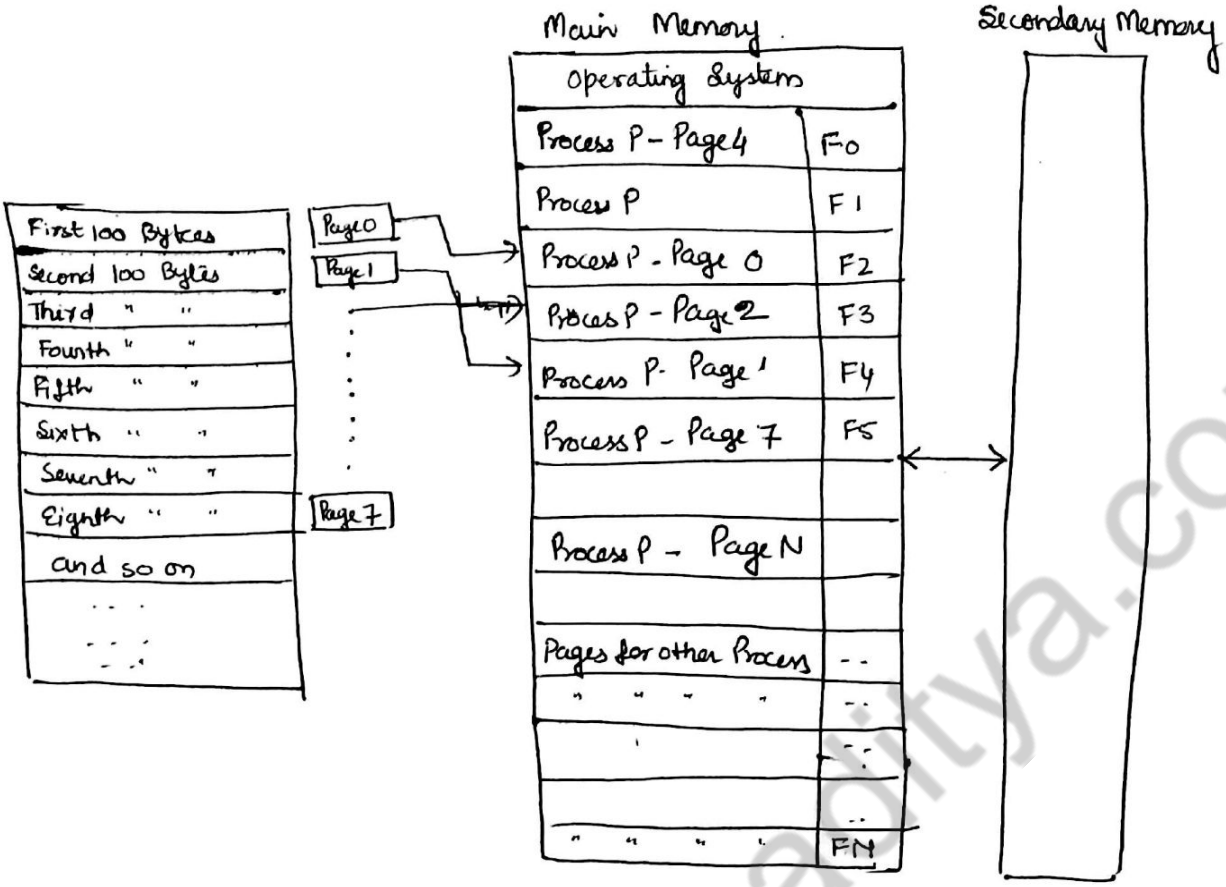
## Page Map table

A data structure called page map table is used to keep track of the relation between a page map table is used to keep track of the relation between a page of a process to a frame in physical memory.




When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is executed, its corresponding pages are loaded into the memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given pt. in time, then the page concept will come into picture. When a computer runs out of RAM, the OS will move idle, or unwanted page of memory to free up RAM for other processes & brings them back when needed by the program.




## Paging Technique

### Address Translation

Page address is called logical address and represented by page number and the offset.

> Logical Address = Page Number + page offset

Frame address is called physical address and represented by a frame number and the offset.

> Physical Address = Frame Number + Page offset

## Advantages and Disadvantages of Paging

→ Paging reduces external fragmentation, but still suffer from internal fragmentation

→ Paging is simple to implement and assumed as an efficient memory management technique.

→ Due to equal size of the pages and frames, swapping becomes very easy.

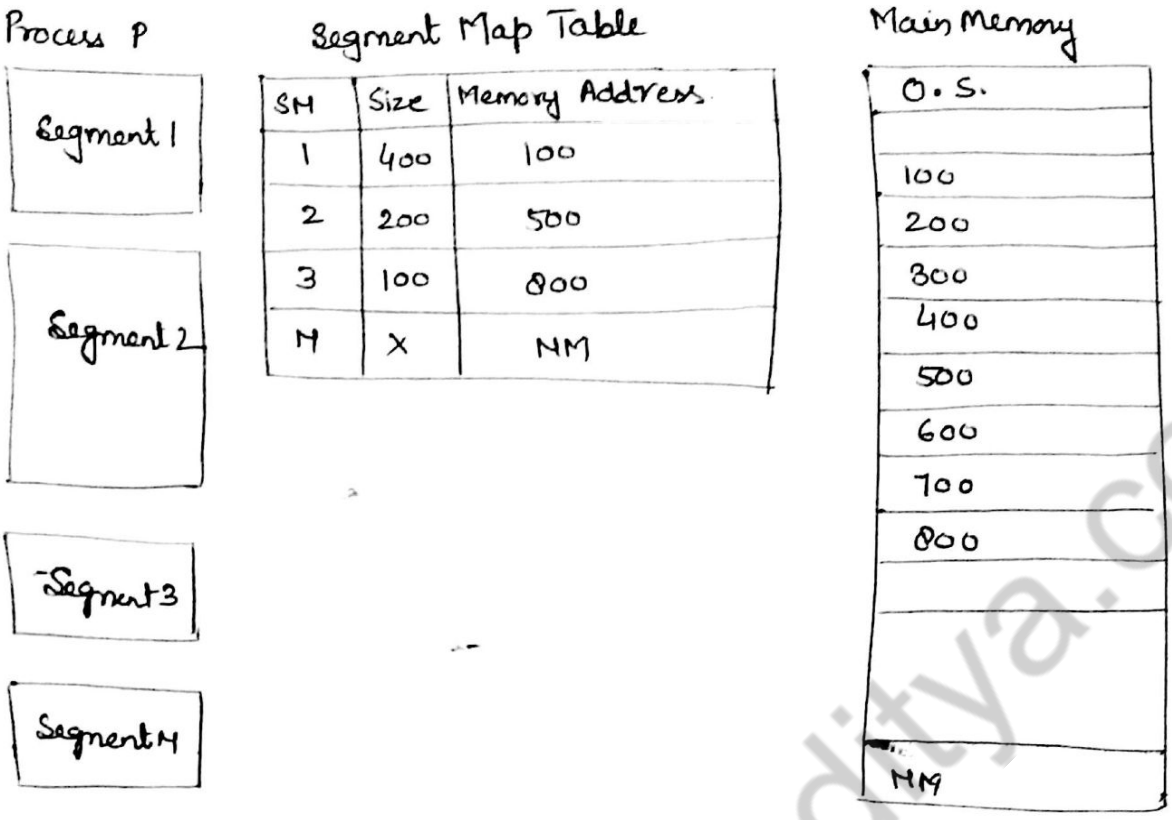
→ Page table requires extra money space, so may ~~no~~ not be good for a system having small RAM.

## Segmentation

Segmentation is a memory mgmt technique in which each job is divided into several segments of different sizes, one for each job ~~is divided into se~~ module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contagious memory though every segment is loaded into a contagious ~~memory~~ block of available memory.

Segmentation memory mgmt works very similar to paging but here segments are of variable length whereas in paging pages are of fixed size.




A program segment contains the program's main memory, utility functions, data structures, and so on. The O.S. maintains a segment map table for every process and a list of free memory block along with segment numbers, their size & corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment & an offset.

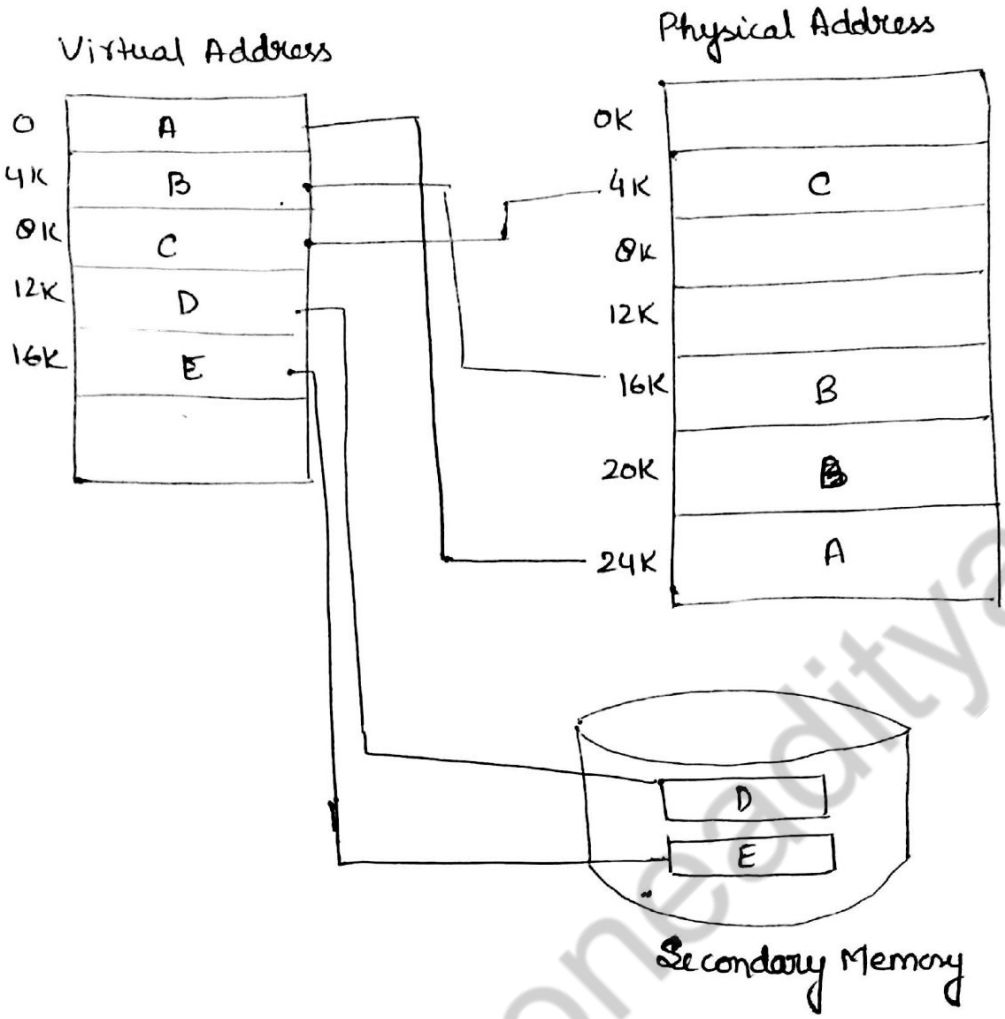
## Virtual Memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it's a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory ~~and it is a section of a hard disk~~ serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

There are several situations where entire program is not required to be loaded in the M.M.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the h/w. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below :-




Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

# UNIT 5

## Mass-Storage Structure

1. Disk Structure :-

A. Magnetic disks provide bulk of secondary storage of modern computers.

→ Drives rotate at 60 to 200 times per second.

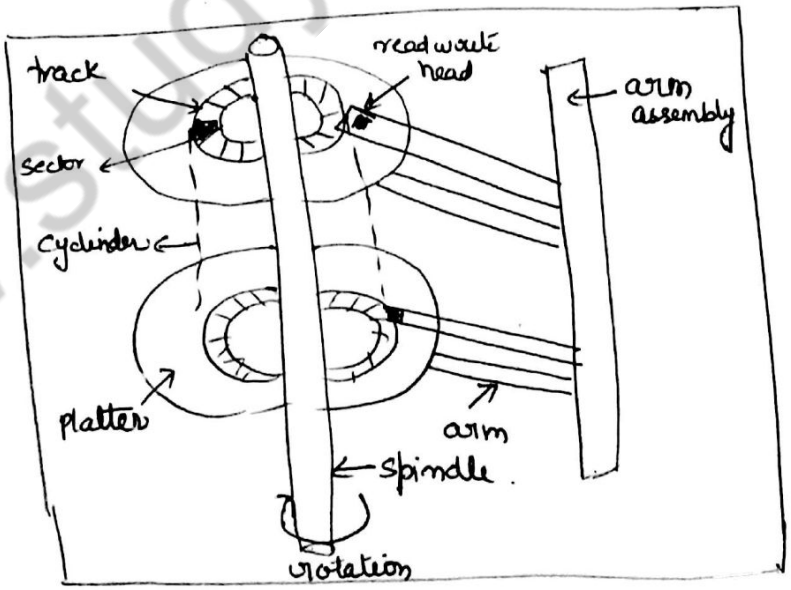
→ Transfer rate is a rate at which data flow between drive & computer

→ Positioning time (random-access-time) is time to move disk arm to desired cylinder (seek time) & time for desired sector to rotate under the disk head (rotational latency)

→ Head crash results from disk head making contact with the disk surface. That's bad.

B. Disk can be removable.

C. Drive attached to computer via I/O bus.



## Disk Scheduling

A. The operating system is responsible for using h/w efficiently - for the disk drives, this means having a fast access time and disk bandwidth.

B. Access time has 2 major components :-

Seeking time :- is the time for the disk are to move the heads to the cylinder.

Rotational Latency :- is the additional time waiting for the disk to rotate the desired sector to the disk head.

C. Minimize seek time.

D. Seek time ≅ seek distance

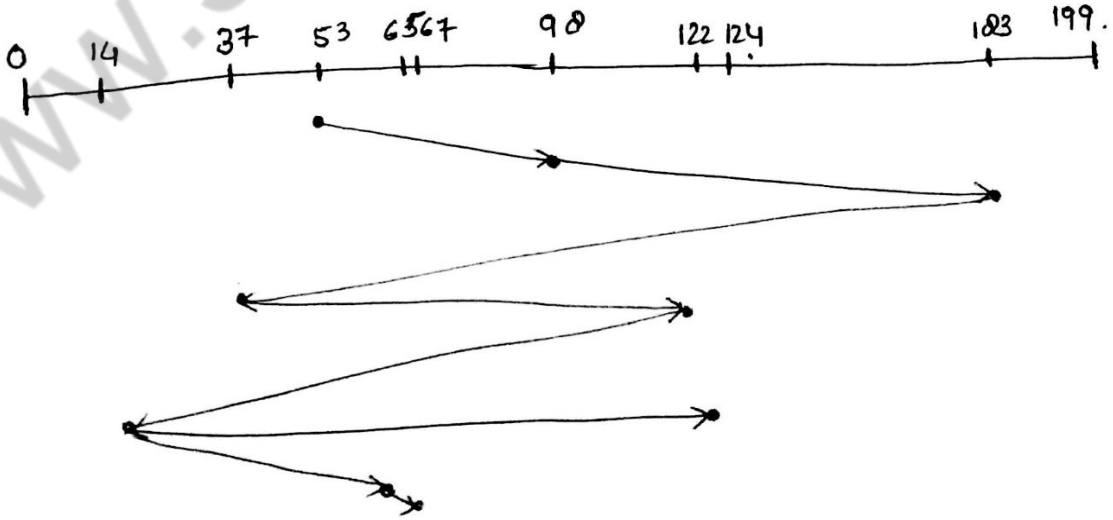
E. Disk bandwidth is the total number of bytes transferred, divided by the total time b/w the first request and the completion of last transfer.

⇒ Several algorithms exists to schedule the serving of disk I/O requests.

1. FCFS scheduling (First Come First Served)

queue = 98, 183, 37, 122, 14, 124, 65, 67.

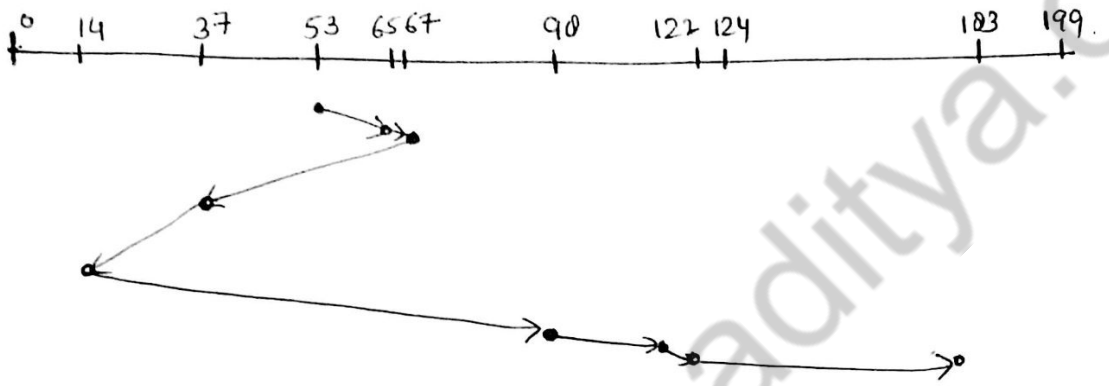
head starts at 53.



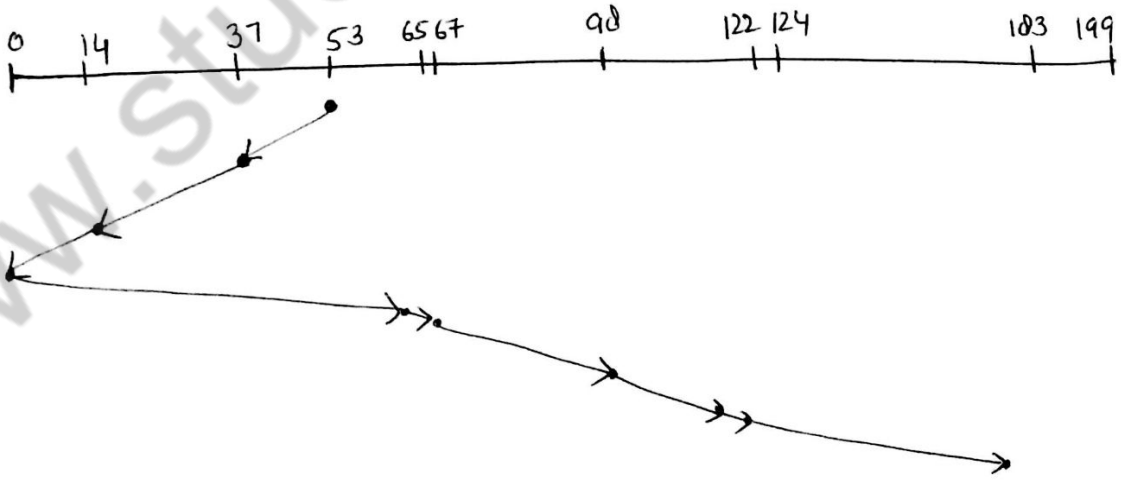
2. SSTF Scheduling (Shortest Seek time First).

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

queue = 98, 183, 37, 122, 14, 124, 65, 65.
head starts at 53.



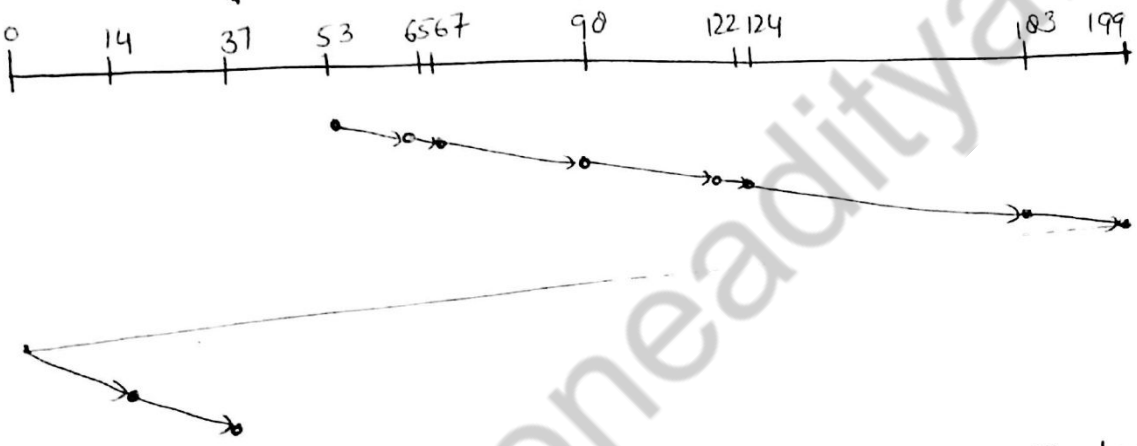

3. SCAN Scheduling:- The disk arm starts at one end of the disk, moves towards the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. Sometimes called the elevator algorithm



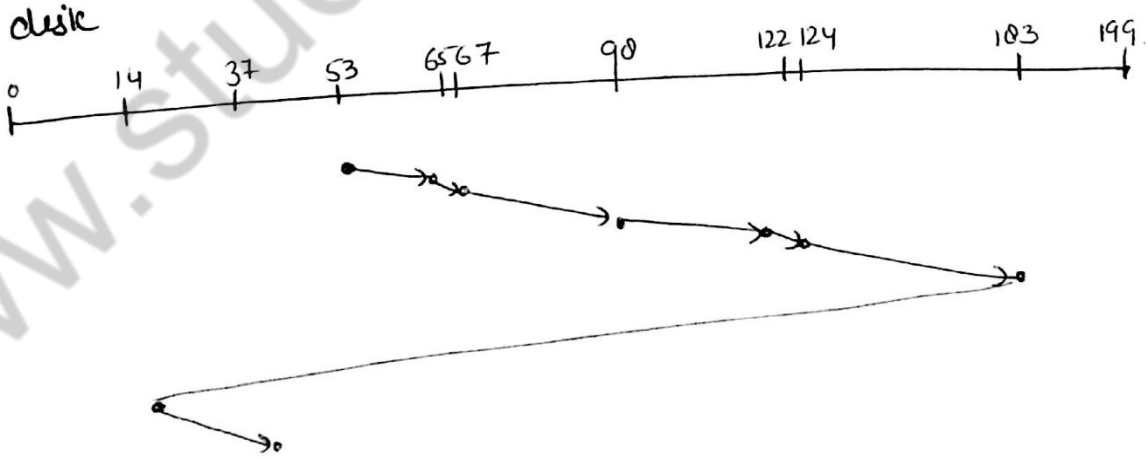
4. CSCAN Scheduling :- Provides a more uniform wait time than SCAN.

→ The head move from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the begining of the disk, without servicing any requests on the return trip.

→ Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.




5. C LOOK Scheduling :- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk



## Criteria for selecting a Disk Scheduling Algorithm

→ SSTF is common and has a natural appeal.
→ SCAN and C-SCAN performs better for systems that place a heavy load on the disk.
→ Performance depends on the number and types of requests.
→ Requests for disk service can be influenced by the file-allocation method.
→ The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
→ Either SSTF or LOOK is a reasonable choice for the default algorithm.

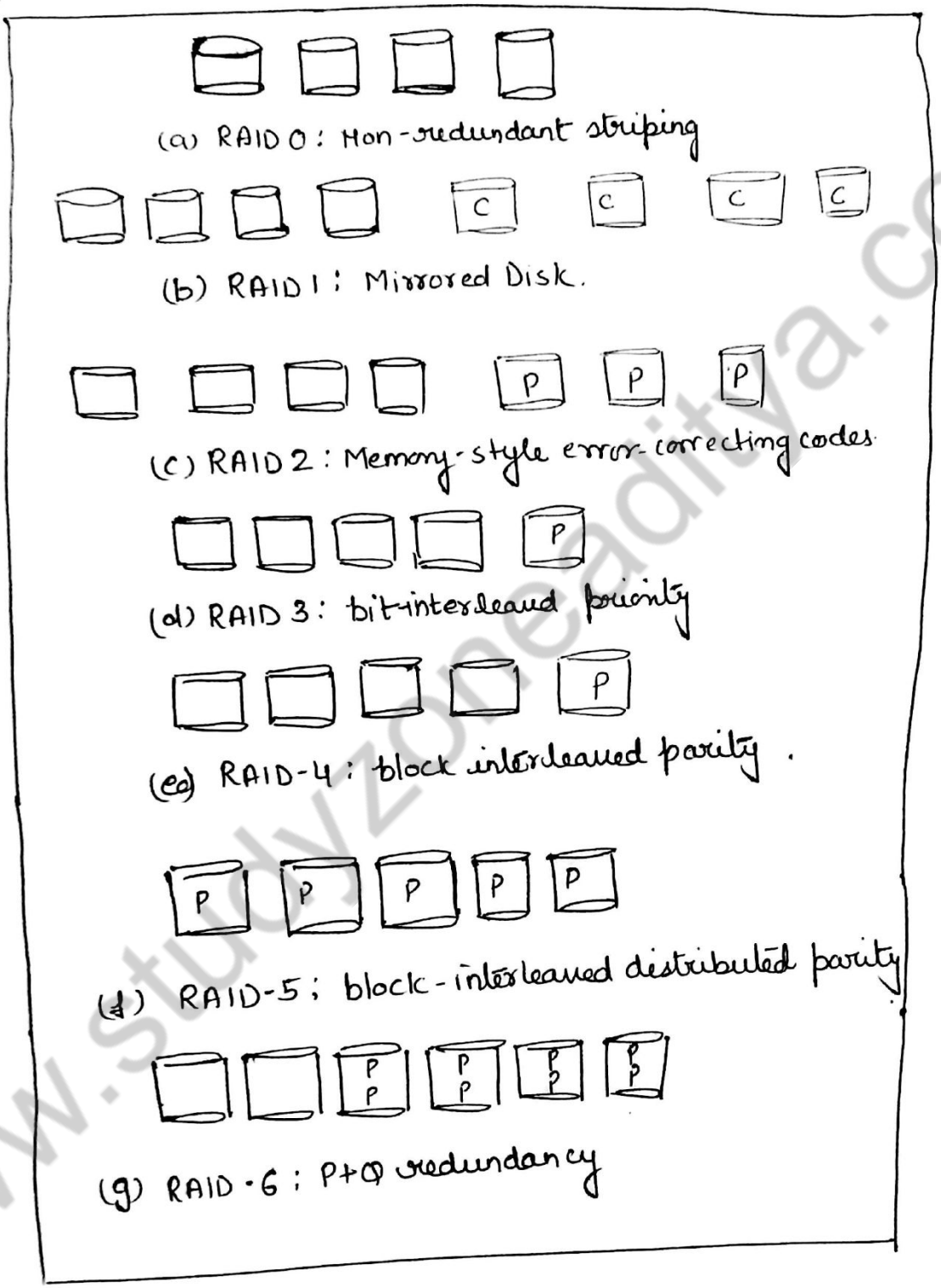
## RAID Structure (Redundant Array Of Inexpensive Disk):

→ RAID - multiple disk drives provides reliability via redundancy.
→ Several improvements in disk-use technique involve the use of multiple disks working cooperatively.
→ Disk striping uses a group of disks as one storage unit.
→ RAID schemes improve the reliability of the storage system by storing redundant data.
  • Mirroring or shadowing keeps duplicate of each disk.
  • Block interleaved parity uses much less redundancy.
→ RAID is arranged into six different levels.

(i) **RAID Level 0:-** Level 0 refers to disk arrays with striping at the level of blocks but without any redundancy (such as mirroring or parity bits).

(ii) RAID Level 1 :-

Level 1 refers to disk mirroring. Fig (b) shows a mirrored organization.




(a) RAID 0 : Non-redundant striping

(b) RAID 1 : Mirrored Disk.

(c) RAID 2 : Memory-style error-correcting codes

(d) RAID 3 : bit-interleaved parity

(e) RAID-4 : block interleaved parity.

(f) RAID-5 ; block-interleaved distributed parity

(g) RAID-6 ; P+Q redundancy

(iii) RAID Level 2:- is also known as memory-style error-correcting code (ECC) organization. Memory systems have long detected certain errors by using parity bits. Each byte in a memory system may have a parity bit associated with it that records whether the no. of bits set to 1 in the byte set to 1 is even parity (parity = 0) or odd parity (parity = 1). If one of the bits in the byte is damaged (either a 1 becomes 0 or a 0 becomes 1), the parity of the byte changes and thus will not match the stored parity.

(iv) RAID Level 3:- or bit interleaved parity org$^n$, improves on level 2 by taking into account the fact that, unlike memory systems, disk controllers can detect whether a sector has been read correctly, so a single parity bit can be used for error correction as well as for detection.

(v) RAID Level 4:- Blocked interleaved parity org$^n$, uses block-level striping as in RAID 0, and in addition keeps a parity block on a separate disk for corresponding blocks from N other disks.

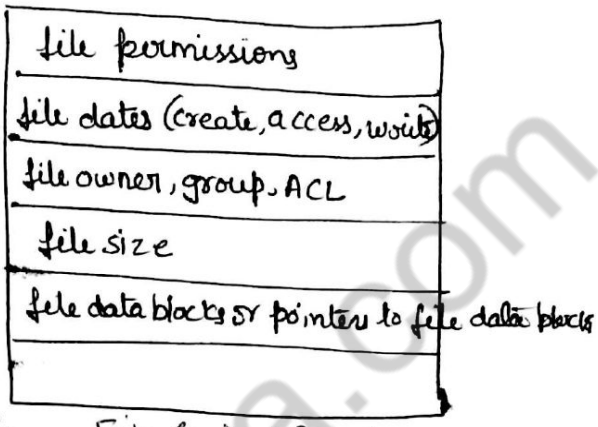
(v) RAID Level 5:- Block interleaved distributed parity differs from level 4 by spreading data and parity among all N+1 disks, rather than storing data in N disks and parity in one disk. For each block, one of the disks stored the parity & others stores data.

(vii) RAID Level 6: P+Q Redundancy scheme, is much like RAID Level 5 but stores extra redundant inf$^n$ to guard against multiple disk failures. Instead of parity, error-correcting codes such as the Reed-Solomon codes are used. In the scheme shown in Fig. (g) 2 bits of redundant data are stored for every four bits of data - compared with 1 parity bit in Level 5 and the system can tolerate two disk failures.

# File System

File Concept :- File organization

→ The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. Files are mapped by the operating system onto physical devices.



File Control Block

→ It contains data in type of numeric, character & binary etc.

→ It has file control block - storage structure consisting of inf$^n$ about a file.

→ It has 2 types of record structure.

Simple record structure : Lines, fixed length, Variable length

Complex Structures :- Formatted Document, Relocatable load file

## File Attributes :-

(i) Name : only information kept in human-readable form

(ii) Identifier : unique tag (number) identifies files within file system

(iii) Type : needed for systems that support different types.

(iv) Size : current file size.

(v) Location : pointer to file location on device.

(vi) Protection : controls who can do reading, writing, executing.

(vii) Time, date and user identification - data for protection, security & usage monitoring.

(viii) Inf$^n$ about files are kept in the directory structure, which is maintained on the disk.

## File Operations :-

(i) File is an abstract data type.
(ii) Create
(iii) Write
(iv) Read
(v) Reposition within file
(vi) Delete
(vii) Truncate
(viii) open(Fi) - search the directory structure on disk for entry Fi and move the content of entry to memory.
(ix) close(Fi) - move the content of entry Fi in memory to directory structure on disk.

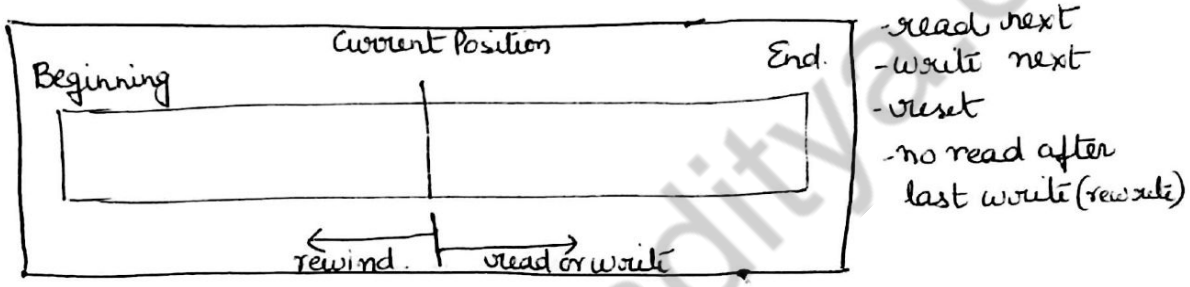
## File Types

→ Executable
→ object
→ source code
→ batch
→ text
→ word processor
→ library
→ print or view
→ archive
→ multimedia

## Access methods of a file

1. Sequential Access :-

The simplest access method is sequential access. Info in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.




2. Direct Access :-

Another method is direct access (or relative access). A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records.

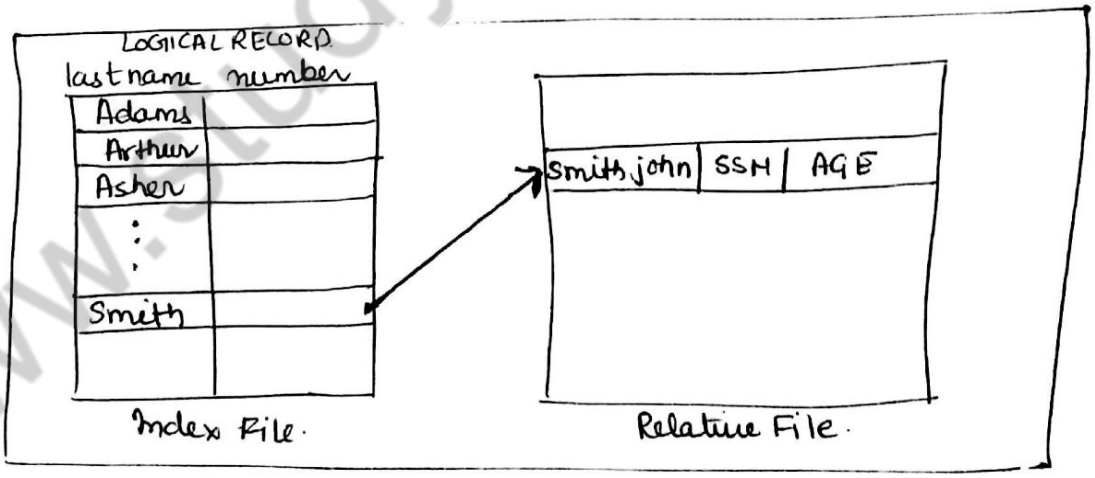
There are no restrictions on the order of reading or writing for a direct-access file.

## Simulation of Sequential Access on a Direct Access file

| Sequential Access | Implementation for direct access. |
|---|---|
| reset | cp = 0; |
| read next | read cp;<br>cp = cp+1 |
| write next | write cp;<br>cp = cp+1; |

### 3. Indexed Sequential Access Method (ISAM):-

Indexed sequential access method (ISAM) uses a small master index that points to disk blocks of a secondary index. The secondary index blocks point to the actual file blocks. The file is kept sorted on a defined key. To find a particular item, we first make a binary search of the master index, which provides the block number of the secondary index. This block is read in, and again a binary search is used to find the block containing the desired record.



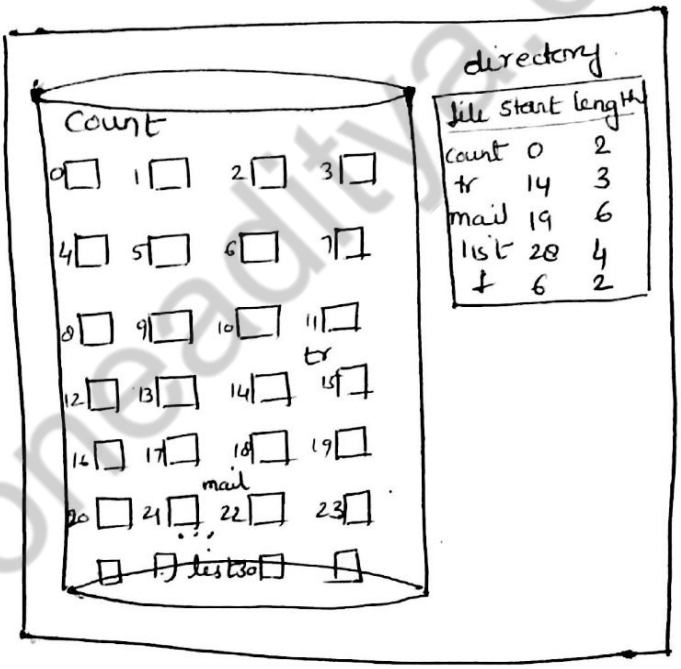
# Allocation Methods

An allocation method refers to how disk blocks are allocated for files :-

(i) Contiguous allocation
(ii) Linked allocation
(iii) Indexed allocation

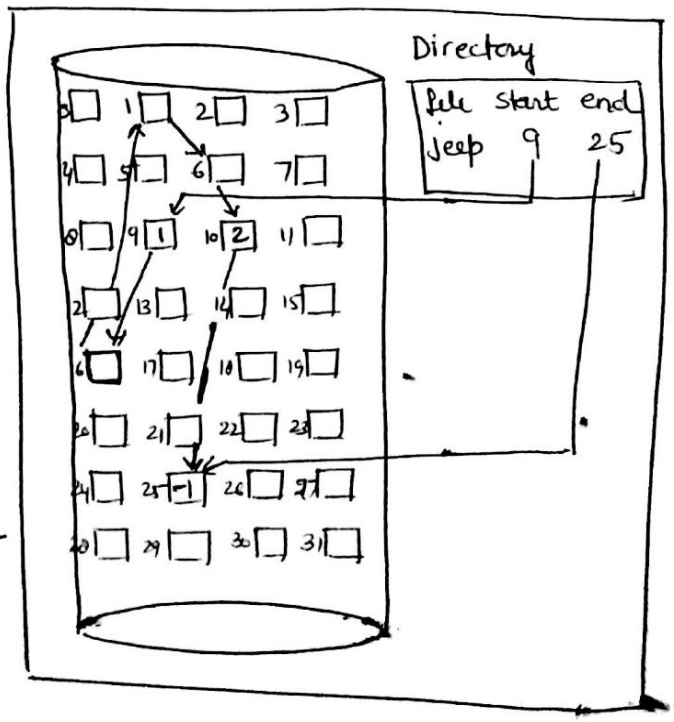
## 1. Contiguous allocation :-

- Each file occupies a set of contiguous blocks on the disk
- Simple - only starting location (block #) and length (no. of blocks) are required.
- Random access
- Wasteful of space (dynamic storage - allocation problem)
- Files cannot grow.



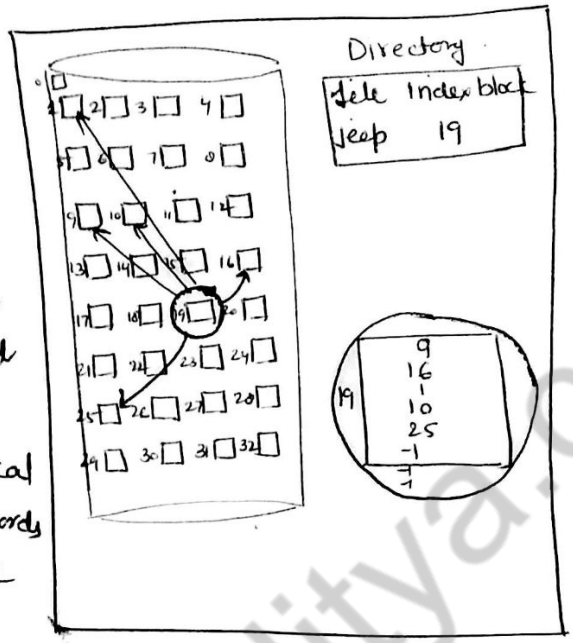

## 2. Linked Allocation :-

- Each file is a linked list of disk blocks : blocks may be scattered anywhere on the disk
- Simple - need only starting addr.
- Free-space mgmt system - no waste of space.
- No random access
- Mapping
  - File allocation table (FAT) - disk-space allocation used by MS-DOS and OS/2



3. Indexed Allocation

- Bring all pointers together into the indexed block.
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of max size of 256K words and block size of 512 words. We need only 1 block for index table.




## FILE SHARING

### Multiple Users :-

→ User IDs identify users, allowing permissions and protections to be per-user

→ Group IDs allow user to be in groups, permitting group access rights.

### Remote file system :-

→ Uses networking to allow file system access between systems.

- Manually via programs like FTP
- Automatically, seamlessly using distributed file system
- Semi automatically via the world wide web.

→ Client server model allows clients to mount remote file systems from servers.

- Server can serve multiple clients
- Client and user-on-client identification is insecure or complicated.
- NFS is standard UNIX client-server file sharing protocol.
- standard operating system file calls are translated into remote calls.

Failure Modes:-

→ Remote file systems add new failure modes, due to n/w failure, server failure.

→ Recovery from failure can involve state inf$^n$ about status of each remote request.

→ Stateless protocols such as NFS include all inf$^n$ in each request, allowing easy recovery but less security.

→ Client server model allows clients to mount remote file systems from servers.

- Server can serve multiple clients
- Client and user-on-client identification is insecure or complicated.
- NFS is standard UNIX client-server file sharing protocol.
- standard operating system file calls are translated into remote calls.

Failure Modes:-

→ Remote file systems add new failure modes, due to n/w failure, server failure.

→ Recovery from failure can involve state inf$^n$ about status of each remote request.

→ Stateless protocols such as NFS include all inf$^n$ in each request, allowing easy recovery but less security.

## File Protection & Access Control

Protection mechanisms provide controlled access by limiting the types of file access that can be made (controlled access). Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

→ Read : Read from the file
→ Write : Write or rewrite the file
→ Execute : Load the file into the memory & execute it
→ Append : Write new info at the end of the file.
→ Delete : Delete the file and free its space for possible reuse.
→ List : List the name & attributes of the files.

(i) The most common approach to implement protect° is to make access dependent on the identity of the user.

(ii) The most general scheme is to implement identity dependent access is to associate with each file & directory an access-control list (ACL) specifying user names and the types of access allowed for each user.

(iii) The main problem with access list is their length. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

(a) Constructing such a list may be a tedious task especially if we do not know in advance the list of users in the system.

(b) The directory entry, previously of fixed size, now needs to be of variable size, resulting in more complicated space mgmt.

(c) These problems can be resolved by use of a condensed version of the access list. To condense the length of the access control list, many systems recognize three classifications of users in connection ~~list~~ with each file.

→ Owner: The user who created the file is the owner.

→ Group: A set of users who are sharing the file and need similar access is a group, or work group.

→ Universe: All other users in the system constitute the universe.

Recovery :-

→ Consistency checking: compares data in directory structure with data blocks on disk, and tries to fix inconsistencie

→ Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)

→ Recover lost file or disk by restoring data from backup.

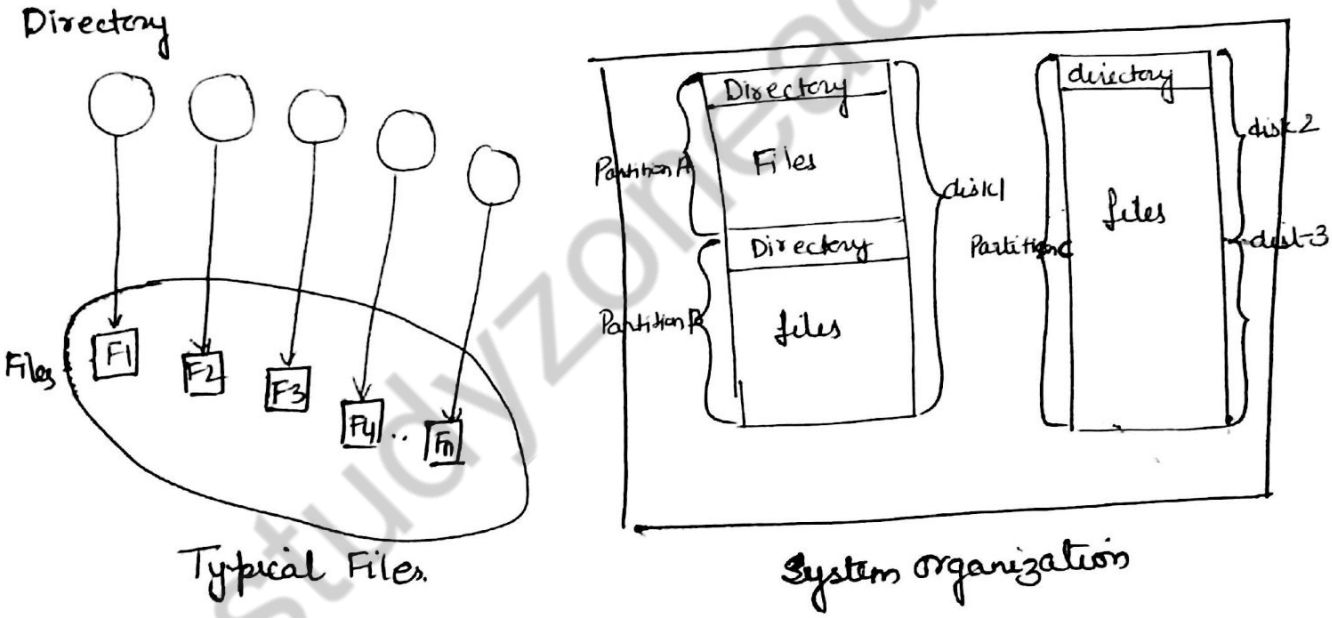
## Directory Structure :-

→ To manage all the data we need to organize them. This org^n is usually done in 2 parts.

→ First, disks are split into one or more partitions, also as minidisks or volumes in the PC.

→ Second, each partition contains inf^n about files within it. This inf^n is kept in entries in a device directory or volume table of contents.

→ A collection of nodes containing inf^n about all files.

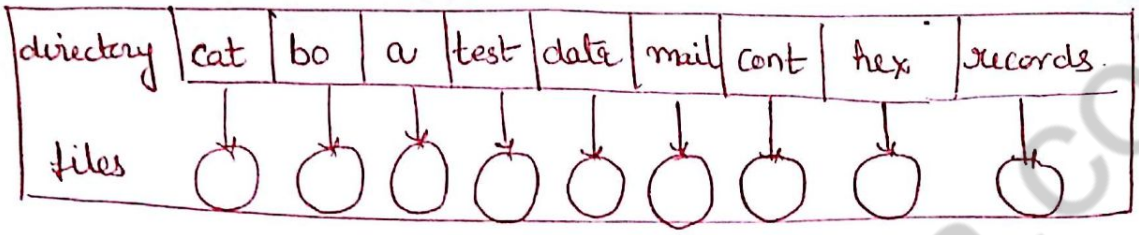



## Single Level Directory :-

→ The simplest directory structure is the single level directory. All files are contained in the same directory, which is easy to support & understand.

→ A single level directory has significant limitations, since all files are in the same directory, they must have unique names.

→ It is difficult to remember the names of all the files as the number of files increases.




## Two level Directory