OPERATING SYSTEM COMPLETE NOTES

TABLE OF CONTENTS

* Note : Page numbers are according to the sequence of this notebook.

OPERATING SYSTEM COMPLETE NOTES

1. Operating System Overview

- An operating system (OS) is system software that acts as an interface between the user and the computer hardware.
- It manages all the resources of a computer and provides services to the users and application programs.

1.1 Functions of Operating System

1. Process management
2. Memory management
3. File management
4. Device management
5. Security and protection
6. User interface
7. Error detection
8. Resource allocation
9. Accounting

1.2 Types of Operating Systems

1. Batch Operating System
2. Multiprogramming Operating System
3. Time Sharing Operating System
4. Real Time Operating System
5. Distributed Operating System
6. Network Operating System
7. Embedded Operating System

1.3 Goals of Operating System

- Convenience - makes the system easy to use.
- Efficiency - uses system resources in an efficient manner.
- Ability to evolve - OS should be constructed in such a way that allows effective development, testing and introduction of new system functions without interfering with service.
- Resource management - allocate resources fairly among users.

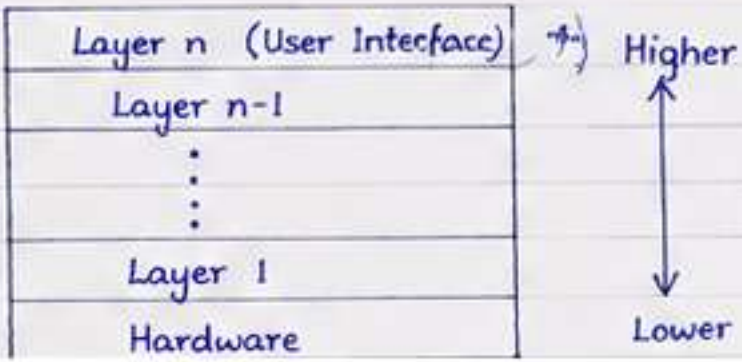
1.4 OS Structure

Operating system can be organized in different ways.

1. **Simple Structure**
   - All OS functions are written as one large program.
   - Not modular.

(Monolithic)

2. **Layered Structure**
   - OS is divided into a number of layers (levels).
   - Each layer uses services of the lower layer.



10. Control over system performance

(1)

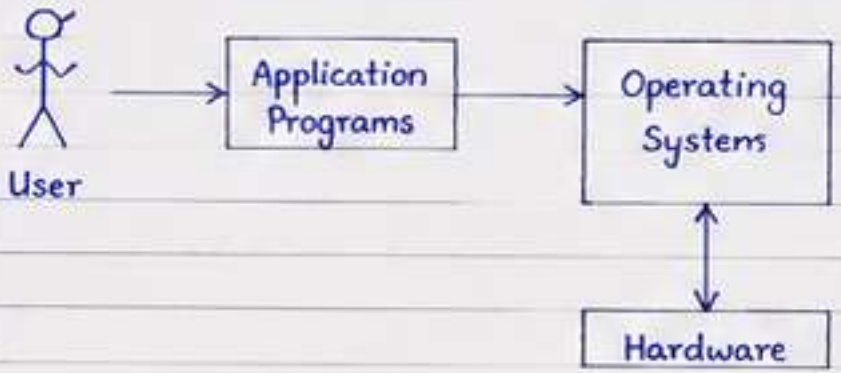
1.5 OS Services

An OS provides the following services to users and programs:

1. Program Execution - The OS loads a program into memory and executes it.
2. I/O Operations - It provides a uniform interface to access I/O devices.
3. File System Manipulation - It allows users to create, delete, read, write and modify files.
4. Communications - It provides mechanism for inter-process communication.
5. Error Detection - It detects and responds to errors.
6. Resource Allocation - It allocates resources among multiple users and programs.

1.6 OS as an Extended Machine

The OS hides the complexity of hardware and provides an abstract machine that is easier to use.



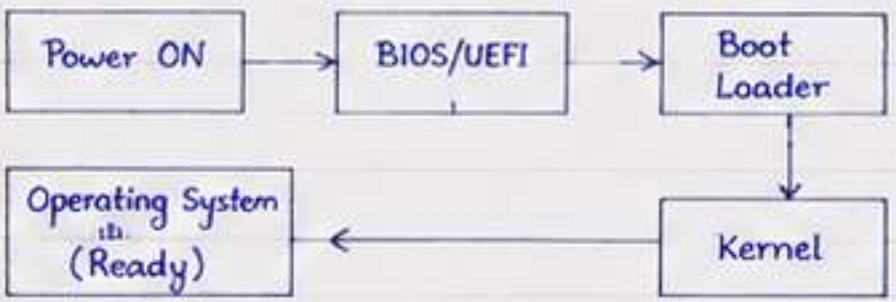
The OS acts as an intermediary between the user/application and the hardware.

> **Note :**
> Without OS, the user must interact directly with hardware which is difficult and inefficient.

1.7 OS Booting Process

When a computer is switched ON, the following steps occur:

1. Power is supplied to the system.
2. Firmware (BIOS/UEFI) runs a Power On Self Test (POST).
3. It locates the boot loader in storage device.
4. Boot loader loads the operating system kernel into memory.
5. Kernel initializes system resources.
6. Finally, OS is ready and user can start working.



1.8 System Calls

System calls provide an interface between a running program and the operating system. Through system calls, a program can request services from the OS.

Categories of System Calls

1. Process control
2. File management
3. Device management
4. Information maintenance
5. Communication
6. Protection

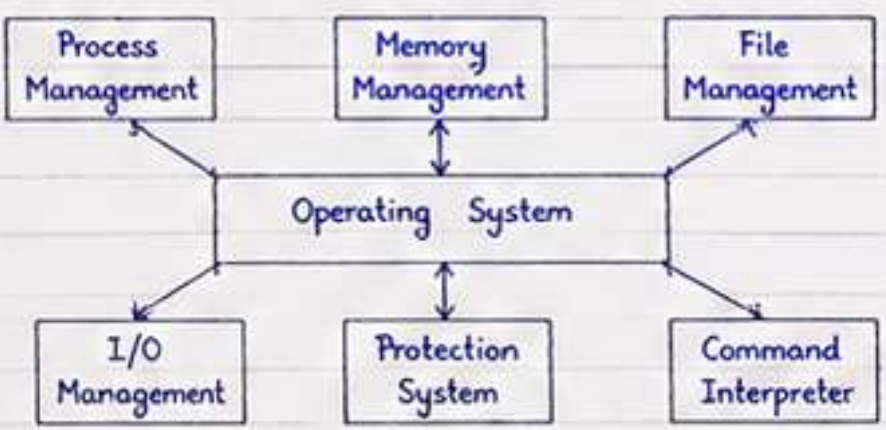
Example of System Calls

- create() - create a new process
- exit() - terminate a process
- open() - open a file
- read() - read data from a file
- write() - write data into a file
- ioctl() - control device
- fork() - create a child process

1.9 OS Components

An operating system consists of the following main components.

- Process Management - Creates, schedules, and terminates processes.
- Memory Management - Keeps track of memory usage and allocates/deallocates memory.
- File Management - Organizes files and directories on storage devices.
- I/O Management - Controls I/O devices and their operations.
- Protection System - Ensures protection and security of data and resources.
- Command Interpreter - Interprets and executes user commands.



1.10 Types of OS

Operating systems can be classified on the basis of their functionality and usage.

1. Batch Operating System

- Jobs having similar needs are batched together and executed.
- It doesn't interact with the user directly.
- Example - IBM OS/360

2. Multiprogramming Operating System

- Several programs are kept in memory at the same time.
- CPU switches among the programs, improving CPU utilization.
- Example - UNIX

3. Time Sharing Operating System

- Multiple users can access the system simultaneously.
- CPU time is divided into small time slots.
- Example - UNIX, Windows, Linux

1.11 Types of OS (Contd.)

4. Real Time Operating System

- Provides quick response within a strict time limit.
- Used in systems where time is critical.
- Example - VxWorks, QNX

5. Distributed Operating System

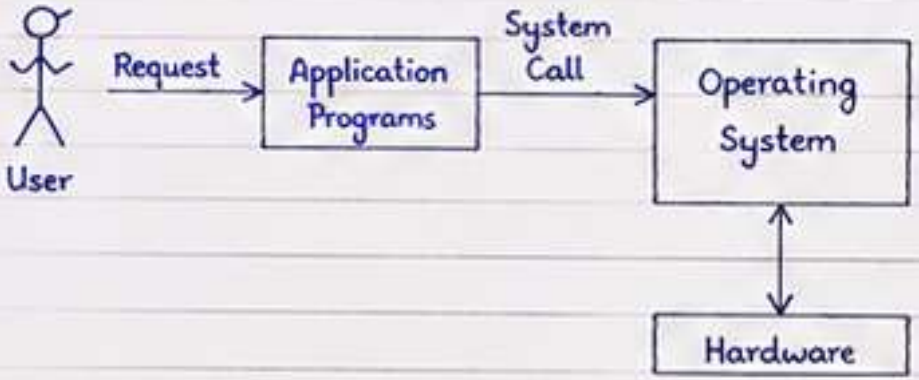
- Manages a group of independent computers and makes them appear as a single system.
- Resources are shared among multiple systems.
- Example - Amoeba, Plan 9

6. Network Operating System

- Allows sharing of files, printers and other resources over a network.
- Provides services to users connected over the network.
- Example - Windows Server, Novell NetWare

1.12 OS Working

The working of an OS can be explained as follows.



Steps :

1. User requests a service through application program.
2. Application program makes a system call to the OS.
3. OS communicates with hardware.
4. Hardware performs the task and returns the result.
5. OS returns the result to the application and then to the user.

1.13 User Mode vs Kernel Mode

A CPU has two modes of operation.

User Mode	Kernel Mode
1. Limited access to system resources.	1. Full access to all system resources.
2. Can't execute privileged instructions.	2. Can execute all instructions.
3. Used by application programs.	3. Used by operating system.
4. Errors in user mode do not affect the entire system.	4. Errors in kernel mode can crash the entire system.

Mode transition :

When a system call or interrupt occurs, the CPU switches from user mode to kernel mode to execute OS services. After completion, it switches back to user mode.

3. PROCESS CONCEPT

A process is a program in execution.
It is an active entity with a program counter, register set, stack, data section and other resources.

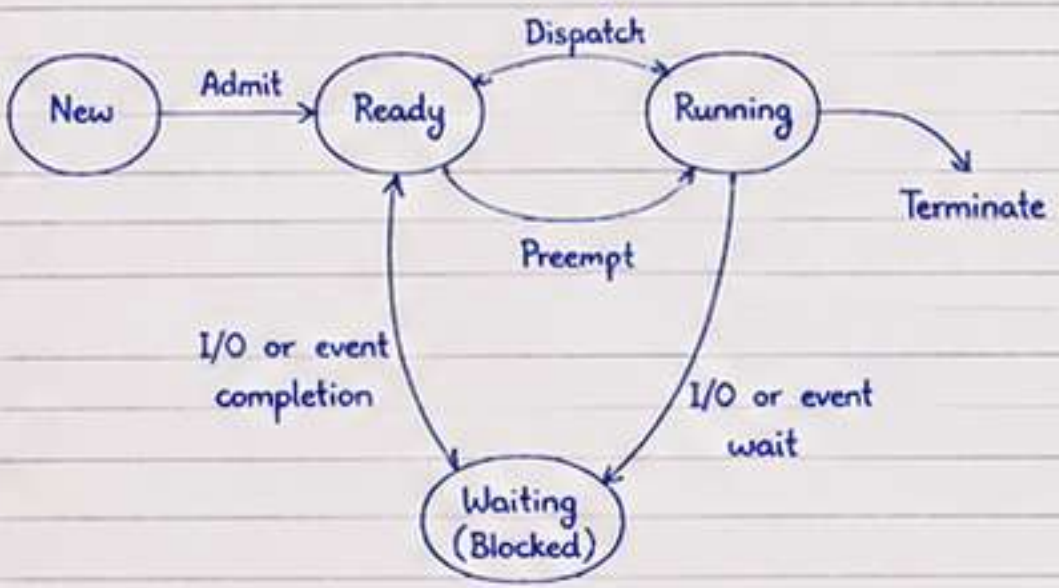
3.1 Process

- A program is a passive collection of instructions stored on disk.
- When a program is loaded into memory and executed, it becomes a process.
- Each process gets CPU time, memory, I/O resources etc.
- Multiple processes may exist in the system simultaneously.

Example:
When we open a browser, the program (passive) becomes a process (active) and starts using CPU and memory.

3.2 Process States

A process changes its state during its lifetime. The five basic states are:



Explanation:

- New : Process is being created.
- Ready : In memory, waiting for CPU.
- Running : Currently executing on CPU.
- Waiting : Waiting for an event or I/O.
- Terminate : Process has finished execution.

3.3 Process Control Block (PCB)

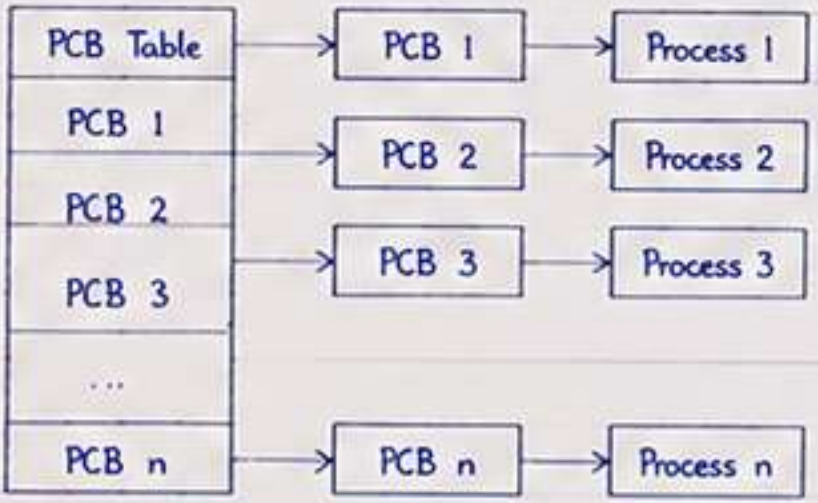
PCB is a data structure maintained by OS for every process. It stores all information about a process.

Process Control Block
Process ID (PID)
Process State
Program Counter
CPU Registers
CPU Scheduling Information
Memory Management Information.
Accounting Information
I/O Status Information
Priority
Pointers (Parent, Child, Next PCB)
...

PCB acts as an identity card of a process. When a process changes state, its PCB is updated accordingly.

3.4 PCB in Memory

All PCBs are kept in the OS kernel area. The OS maintains a PCB table or linked list of PCBs.

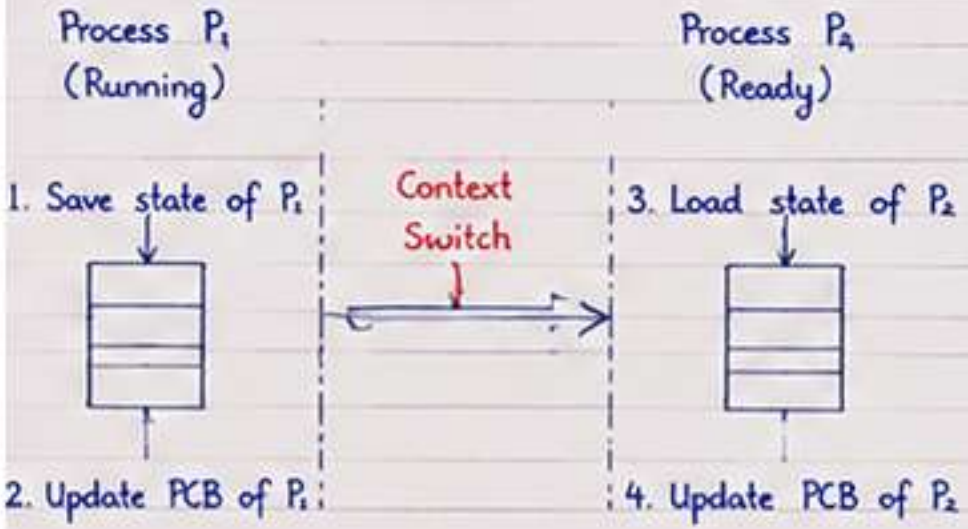


Benefits :

- OS can easily find and manage all processes.
- Helps during context switching.
- Keeps track of process information.

3.5 Context Switching

When CPU switches from one process to another, the state of the current process is saved and the state of next process is restored. This is called context switching.



Context switching takes time and CPU cycles. So, excessive context switching reduces system performance.

3.6 CPU Scheduling (Basic Idea)

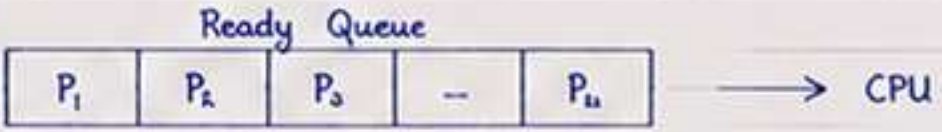
When multiple processes are in ready queue, OS scheduler selects one process and allocates CPU to it.

Objectives of CPU Scheduling:

1. High CPU utilization
2. High throughput
3. Low waiting time
4. Low turnaround time
5. Fairness

Scheduling Queue:

Ready Queue contains all processes that are in ready state.



Scheduler picks one process from the queue and assigns CPU.

4.1 Scheduling Criteria

The CPU scheduling algorithm is evaluated based on the following criteria :

1. CPU Utilization - Keep CPU busy 100%
   (or as much of the time as possible)
2. Throughput - Number of processes completed per unit time.
3. Turnaround Time - Total time from submission to completion.
4. Waiting Time - Total time a process spends in ready queue.
5. Response Time - Time from submission till first response.
6. Fairness - Make sure each process gets a fair share.
7. Priority - Higher priority processes should be executed first.

No scheduling algorithm is perfect. A good algorithm balances all the above criteria.

4.2 Types of Scheduling

Scheduling can be of the following types :

1. Non-preemptive Scheduling
2. Preemptive Scheduling

1. Non-preemptive Scheduling :

Once the CPU is allocated to a process, it cannot be taken away until the process completes its CPU burst or blocks for I/O.

2. Preemptive Scheduling :

The CPU can be taken from a running process and given to another process based on some criteria.

Non-preemptive	Preemptive
CPU not taken from process until it releases CPU.	CPU can be taken from process.
Simple to implement. Less overhead.	Complex to implement. More overhead.

4.3 Scheduling Algorithms

Some common CPU scheduling algorithms are :

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Priority Scheduling
4. Round Robin (RR)
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

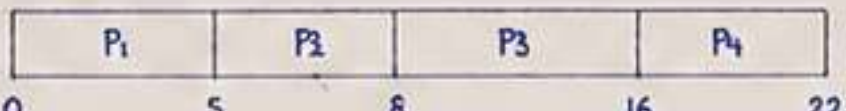
4.3.1 First Come First Serve (FCFS)

The process that requests CPU first is allocated CPU first. It is based on FIFO queue.
It is a non-preemptive algorithm.

Example :

Process	Arrival Time	Burst Time
P_1	0	5
P_2	1	3
P3	2	8
P_4	3	6

Gantt Chart :



4.3.2 Shortest Job First (SJF)

The process with the smallest CPU burst time is selected next.

It can be non-preemptive or preemptive (Shortest Remaining Time First).

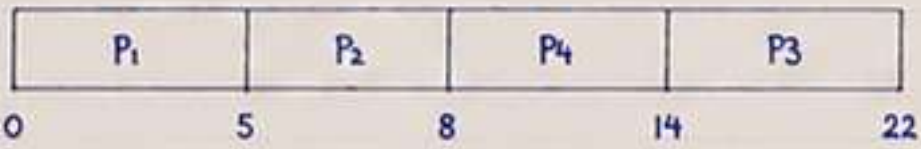
Here we consider non-preemptive SJF.

Using the same example :

Process	Arrival Time	Burst Time
P_1	0	5
P_2	1	3
P_3	2	8
P_4	3	6

Order of execution : $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_3$

Gantt Chart :



4.3.3 Priority Scheduling

Each process is assigned a priority.

Process with highest priority is executed first.

(Smaller number = Higher priority)

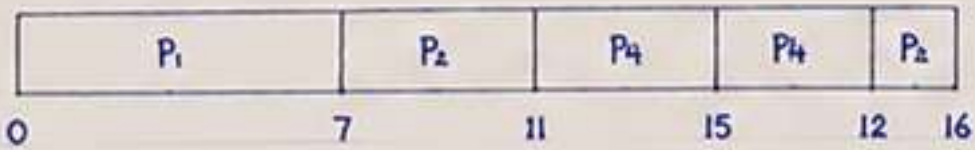
It can be preemptive or non-preemptive.

Example :

Process	Arrival Time	Burst Time	Priority
P_1	0	7	2
P_2	1	4	1
P3	2	1	3
P4	3	4	2

Execution Order : $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_3$

Gantt Chart :



4.3.4 Round Robin (RR)

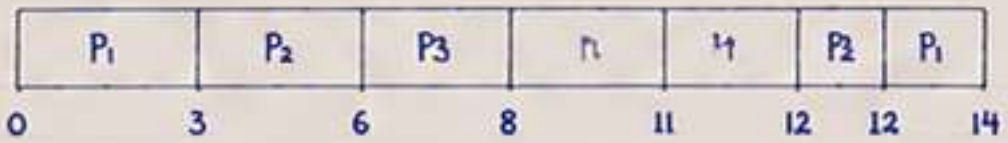
Each process is given a small time unit called Time Quantum (TQ).

If a process does not finish in TQ, it is preempted and added to the end of queue.

Example : Time Quantum = 3

Process	Arrival Time	Burst Time
P_1	0	5
P_2	1	4
P3	2	2

Gantt Chart :



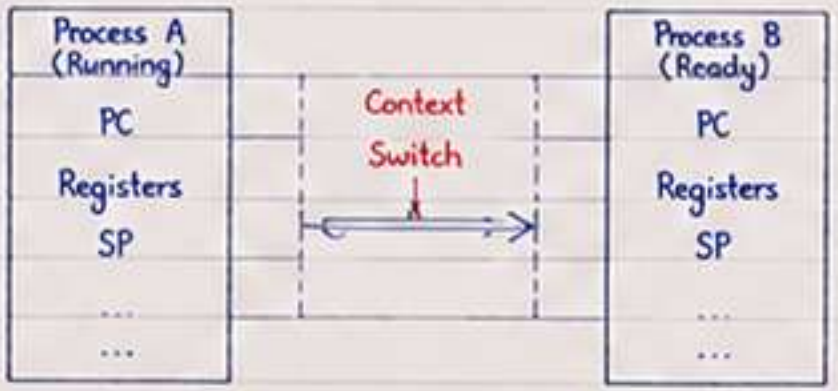
Round Robin is widely used in time sharing systems.

5. CONTEXT SWITCHING (Contd.)

Context switching is essential for multitasking. It allows the OS to switch the CPU from one process to another.

Steps in Context Switching :

1. Save the state of current process in PCB.
2. Update PCB state to Waiting/Ready.
3. Select another process from ready queue.
4. Load state of selected process from its PCB.
5. Update PCB state to Running.



Context switching is time consuming and causes overhead.

5.1 THREADS

A thread is a basic unit of CPU utilization within a process.

- All threads of a process share the same code, data, and resources.
- Threads have their own program counter, register set and stack.

Types of Threads :

1. User Level Threads
   - Managed by user level thread library.
   - Efficient to create and manage.

2. Kernel Level Threads
   - Managed by the OS kernel.
   - More overhead but true concurrency.

Feature	User Level Threads	Kernel Level Threads
Managed By	User Library	OS Kernel
Creation Time	Less	More
Context Switch	Fast	Slow
Parallelism	No	Yes
Blocking	Affects entire process	Does not affect other threads

5.2 MULTIPROCESSING

Multiprocessing systems have more than one CPU.
CPU's can be of two types :

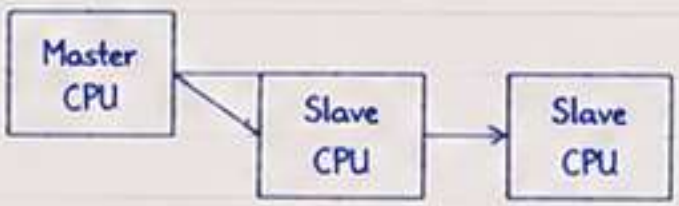
1. Asymmetric Multiprocessing (AMP)

- One processor acts as master and others as slaves.
- Master schedules and assigns work.

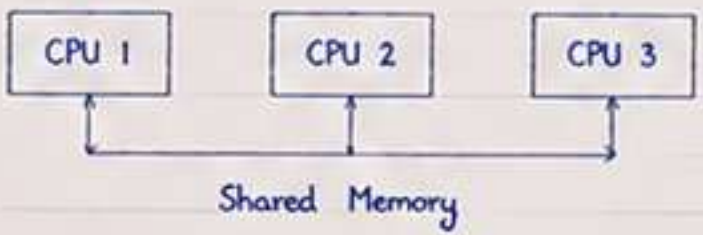
2. Symmetric Multiprocessing (SMP)

- All processors are equals.
- Each processor can schedule and execute processes.

AMP System



SMP System



5.3 PROCESS COMMUNICATION

Processes may need to communicate and exchange information.

Two models of IPC (Inter Process Communication):

1. Shared Memory

- A region of memory is shared among processes.
- Processes can read/write to this region.
- Faster but needs synchronization.

2. Message Passing

- Processes communicate by sending and receiving messages.
- Uses send() and receive() operations.
- Slower but easier to implement.

5.4 PROCESS SYNCHRONIZATION

When multiple processes access shared data concurrently, data inconsistency may occur. Synchronization is used to coordinate processes and maintain data consistency.

Critical Section :

Section of code where shared resources are accessed.

Only one process should be in critical section at a time.

Requirements for critical section solution :

1. Mutual Exclusion - Only one process in CS.
2. Progress - If no process is in CS, selection cannot be postponed indefinietly.
3. Bounded Waiting - A limit should exist on how many times other processes are allowed to enter CS after a process has requested.

5.5 SEMAPHORE

A semaphore is an integer variable used to control access to shared resources.

Two types of semaphore :

1. Binary Semaphore (0 or 1)
   - Also called mutex lock.
   - Used for mutual exclusion.
   - 0 → resource busy, 1 → resource free.

2. Counting Semaphore (0 to n)
   - Used to control access to multiple instances of a resource.
   - Value indicates number of available instances.

Semaphore Operations :

Operation	Meaning
wait() / P() (decrement)	If value > 0, decrement else block.
signal() / V() (increment)	Increment the value and wake up waiting process.

6. PROCESS CREATION AND TERMINATION

6.1 Process Creation

When a new process is created, the OS performs the following steps :

- Assign a unique process ID.
- Allocate memory for the process.
- Initialize the PCB.
- Set up the necessary resources.
- Place the new process in the ready queue.

6.2 Process Termination

A process terminates when :

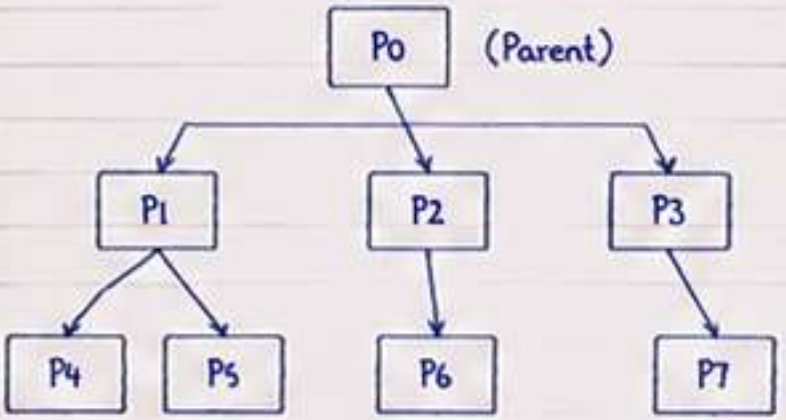
- It finishes execution normally.
- It encounters an error.
- It is aborted by another process.
- It is terminated by the OS.

Steps in process termination :

- Release all resources held by the process.
- Deallocate memory.
- Remove PCB from the system.
- Wake up parent process if waiting.

6.3 Process Tree

Processes are often arranged in a tree structure. A process can create child processes, which in turn can create their own children.



P0 is the parent of P1, P2 and P3.
P1 is the parent of P4 and P5 and so on.

6.4 Orphan and Zombie Process

- **Orphan Process :**
  A child process whose parent has terminated before it.
- **Zombie Process :**
  A process that has completed execution but whose parent has not collected its exit status.

6.5 Inter Process Communication (IPC)

IPC allows processes to communicate and synchronize their actions.

Methods of IPC :

1. Shared Memory
2. Message Passing
3. Pipes
4. Signals
5. Sockets

Comparison of IPC Methods :

Method	Speed	Synchronization	Use Case
Shared Memory	High	Required	Large data exchange
Message Passing	Medium	Not required	Small data exchange
Pipes	Medium	Required	Between related processes
Signals	High	Not required	Event notification
Sockets	Medium	Required	Between systems

6.6 Pipes

A pipe is a unidirectional communication channel that allows data to be passed from one process to another.

Types of Pipes :

1. **Anonymous Pipe**
   - Used between related processes.
2. **Named Pipe (FIFO)**
   Used between unrelated processes.

Example of pipe between two processes :

Data flows in only one direction.

Advantages :

- Simple to implement.
- Efficient for small data transfer.

Disadvantages :

- Limited to parent-child processes (anonymous pipe).
- Unidirectional.

6.7 Signals

Signals are used to notify a process about the occurrence of an event.

Common Signals :

Signal	Purpose
SIGINT	Interrupt from keyboard (Ctrl + C)
SIGTERM	Termination request
SIGKILL	Kill signal (cannot be caught)
SIGSTOP	Stop the process
SIGCONT	Continue the stopped process

Signal Handling Steps :

1. Signal is generated.
2. Signal is delivered to the process.
3. Signal is handled (default or user defined).

Note :

Signals provide a simple way of communication between OS and a process.

6.8 Summary:

- A process is a program in execution.
- Processes go through different states.
- PCB stores all information about a process.
- CPU scheduling selects a process for execution.
- IPC allows processes to communicate.
- Proper management of processes improves system performance.

Remember :

Efficient process management is the key to a good operating system.

7. DEADLOCK

A deadlock is a situation where a set of processes are blocked forever, each waiting for a resource held by another process in the set.

7.1 Necessary Conditions for Deedlock

The four necessary conditions (Coffman conditions) for deadlock are :

1. Mutual Exclusion – At least one resource must be non-shareable.
2. Hold and Wait – A process holding at least one resource is waiting to acquire additional resources held by other processes.
3. No Preemption – Resources cannot be forcibly taken; they must be released voluntarily.
4. Circular Wait – A set of waiting processes $\{P_0, P_1, \ldots, P_n\}$ exists such that P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2, ..., and P_n is waiting for a resource held by P_0.

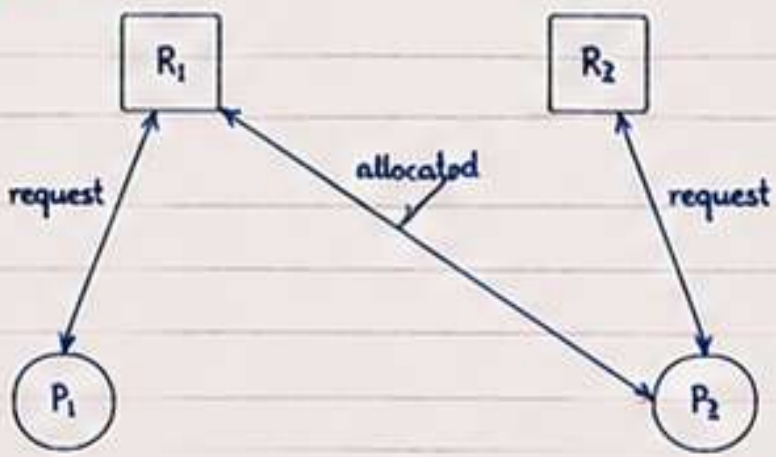
If any one of the above condition is not satisfied, deadlock cannot occur.

7.2 Resource Allocation Graph (RAG)

RAG is a directed graph used to describe the state of resource allocation.

- Circle → Process
- Rectangle → Resource type
- Request edge ($P_i \rightarrow R_j$) : P_i is requesting an instance of R_j.
- Assignment edge ($R_j \rightarrow P_i$) : An instance of P_j is allocated to P_i.

Example :



If the graph contains a cycle, then a deadlock exists (for single instance of each resource).

7.3 Handling Deadlock

There are four approaches to handle deadlock:

1. **Deadlock Prevention**
   Ensure that at least one of the four necessary conditions never holds.

2. **Deadlock Avoidance**
   Allocate resources only if the system remains in a safe state.

3. **Deadlock Detection and Recovery**
   Allow deadlocks to occur, then detect them and recover from them.

4. **Ignore the Problem**
   Used when deadlocks are rare and the cost of prevention is high.

☆ Deadlock prevention and avoidance are done before deadlock occurs, while detection and recovery are done after it occurs.

7.4 Deadlock Prevention (Breaking a Condition)

We can prevent deadlock by ensuring at least one of the four necessary conditions does not hold.

Condition	How to Prevent
Mutual Exclusion	Make resources sharable where possible.
Hold and Wait	Require process to request all resources at once.
No Preemption	If a process holding resources requests another, preempt the held resources.
Circular Wait ...	Impose a total ordering on resource types and require processes to request in order.

Limitation :

Prevention may lead to low resource utilization and poor system performance.

7.5 Deadlock Avoidance

Deadlock avoidance ensures that the system never enters an unsafe state.

The most well-known algorithm for single instance per resource type is Banker's Algorithm.

Safe State :

A state is safe if there exists a sequence of processes $<P_1, P_2, \ldots, P_n>$ such that each process can obtain its maximum required resources.

Banker's Algorithm (Steps) :

1. Check if Request$_i$ $\leq$ Need$_i$.
2. Check if Request$_i$ $\leq$ Available.
3. Pretend to allocate resources to P_i and update Available and Need.
4. If the system is in safe state, grant the request; otherwise, P_i must wait.

7.6 Deadlock Detection and Recovery

7.6.1 Detection

Allow deadlock to occur, then detect it using algorithms like :

- Wait-for Graph (for single instance resources)
- Detection Algorithm (for multiple instances)

7.6.2 Recovery :

Once deadlock is detected, recover by :

- Process Termination – Abort one or more processes.
- Resource Preemption – Preempt resources from processes.

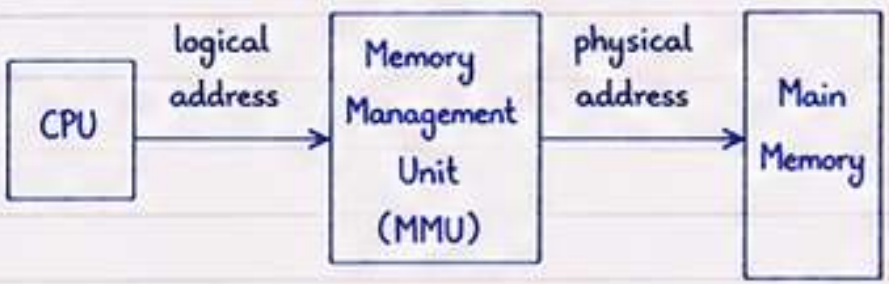
> **Note :**
> Victim selection in recovery should minimize cost and ensure fairness.

8. MEMORY MANAGEMENT

Memory management keeps track of which parts of memory are being used and by whom. It allocates and deallocates memory as per requirement.

8.1 Logical vs Physical Address Space

- Logical Address : Generated by CPU.
- Physical Address : Actually present in main memory.



8.2 Memory Allocation

Types :

- Contiguous Allocation
- Non-contiguous Allocation
  - Paging
  - Segmentation

8.2.1 Contiguous Allocation

In contiguous allocation, each process is allocated a single continuous block of memory.

Advantages :

- Simple to implement
- Less overhead

Disadvantages :

- External fragmentation
- Difficult to allocate memory to a process if a large contiguous block is not available.

8.2.2 Non-contiguous Allocation

A process is allocated memory in non-contiguous blocks.

1. Paging :

- Memory is divided into fixed size blocks called frames.
- Process is divided into same size blocks called pages.
- Pages of a process can be placed in any free frames.

2. Segmentation :

- A process is divided into variable size segments based on logical units (e.g., code, data, stack).
- Each segment has a base and limit.

Segment Table Format :

Segment No.	Base	Limit
0	b_0	l_0
1	b_1	l_1
2	b_2	l_2
...	...	...
n	b_n	l_n

Memory Management Unit (MMU)

MMU is a hardware device that maps logical address to physical address and provides memory protection.

8.3 Page Table

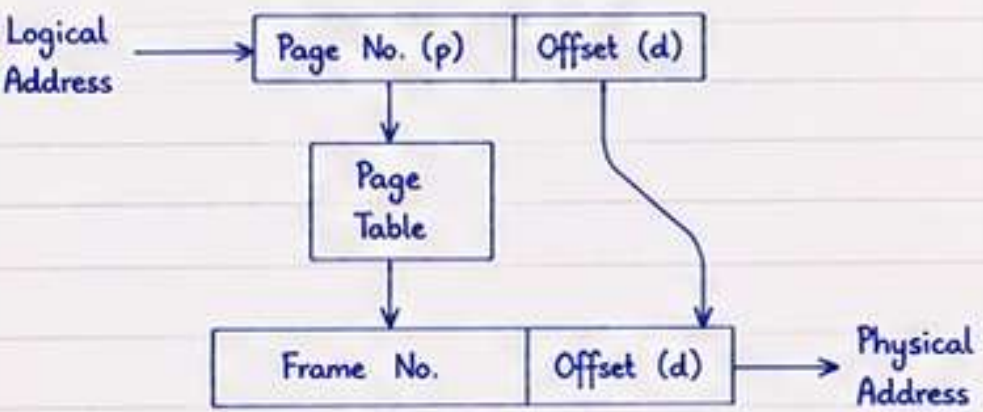
In paging, a page table is used to keep track of all pages of a process.

Each entry in page table contains the frame number where the page is located.

Example :

Page No.	Frame No.
0	5
1	2
2	9
3	1
...	...
n	m

8.4 Address Translation in Paging



8.5 Page Replacement Algorithms

When a page fault occurs and no frame is free, OS must replace one of the existing pages.

Common Algorithms :

1. FIFO (First In First Out)
2. Optimal
3. LRU (Least Recently Used)
4. LFU (Least Frequently Used)

Example : FIFO (Page Reference String)

Reference String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3

Frames = 3

7	7	7	2	2	3	3	4	2	2	0
-	0	0	0	0	0	0	0	3	3	3
-	-	1	1	1	1	1	2	2	2	0

Page Faults = 9

8.6 Thrashing

If a process does not have enough frames, it will suffer a lot of page faults. This leads to thrashing.

- CPU utilization decreases.
- System performance degrades.

Thrashing :

A condition where the system spends most of the time in paging activity and very little in executing useful work.

8.7 Summary :

- Memory management handles allocation and deallocation of memory.
- Paging and segmentation are non-contiguous allocation techniques.
- Page replacement algorithms help in improving memory utilization.
- Thrashing should be avoided for good performance.

9. FILE MANAGEMENT

File management is the part of OS that deals with creating, deleting, organizing, storing and accessing files.

9.1 File Concept

A file is a named collection of related information stored on secondary storage.

Types of Files :

- Text Files – Contain characters
- Binary Files – Contain binary data
- Executable Files – Ready to execute
- System Files – Used by OS
- Library Files – Collection of routines

9.2 File Attributes

Attribute	Description
Name	Human readable file name
Identifier	Unique tag (number)
Type	Text, binary, etc.
Location	Pointer to storage location
Size	Current size of file
Protection	Access control information
Time, date, user ID	Information about creation, modification and last access

9.3 File Operations

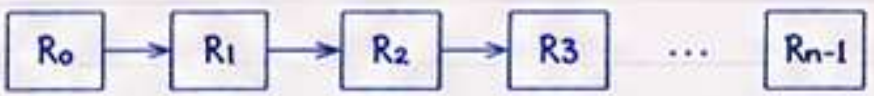
Common file operations are :

1. Create - To create a new file
2. Open - To open an existing file
3. Read - To read data from file
4. Write - To write data into file
5. Append - To add data at the end of file
6. Close - To close an opened file
7. Delete - To remove a file

9.4 File Access Methods

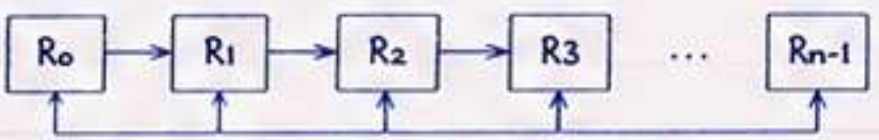
1. Sequential Access

Records are accessed in sequential order from the beginning to the end.



2. Direct Access (Random Access)

Any record can be accessed directly without reading previous records.



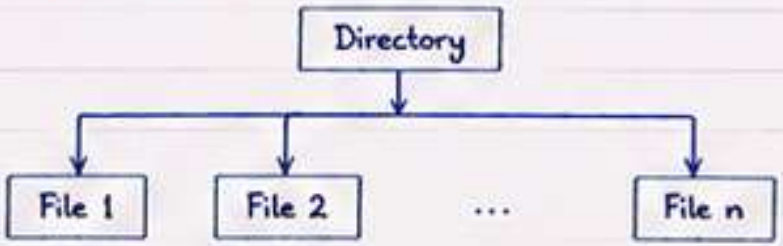
Direct access is faster than sequential access but needs more complex hardware.

9.5 Directory Structure

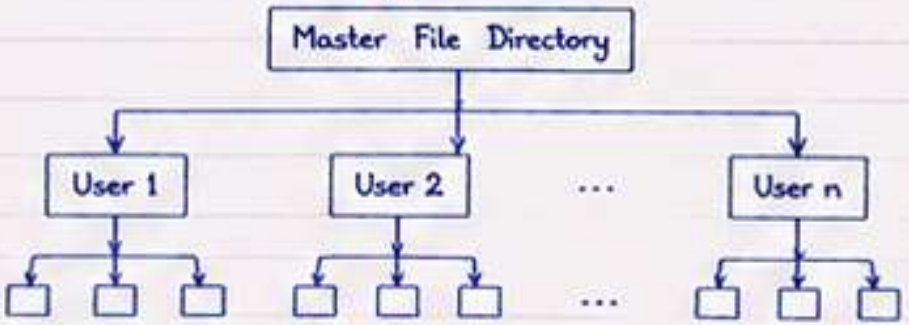
Directory is a special file that contains information about all files.

Types of Directory Structure :

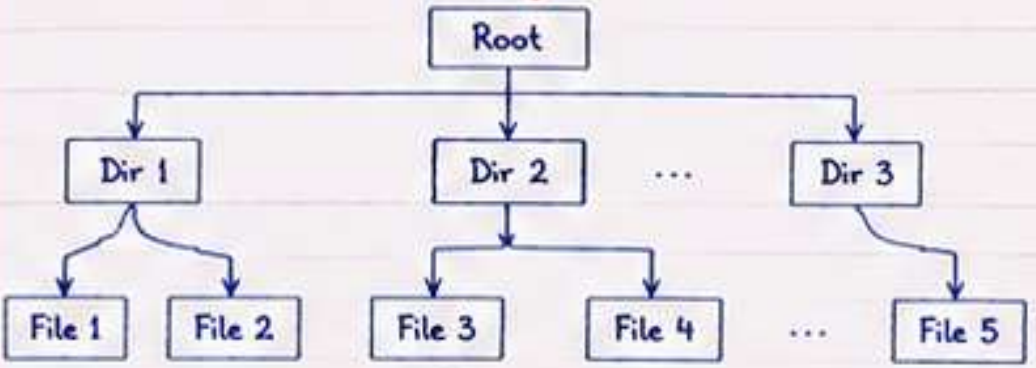
1. Single Level Directory



2. Two Level Directory



3. Tree Structured Directory



9.6 File Protection

Protection is needed to control access to files.

Access Types :

- Read – View file contents
- Write – Modify file contents
- Execute – Execute the file
- Delete – Remove the file

Methods of Protection :

- Access Control Matrix
- Access Control List
- Capability List

Access Control Matrix Example :

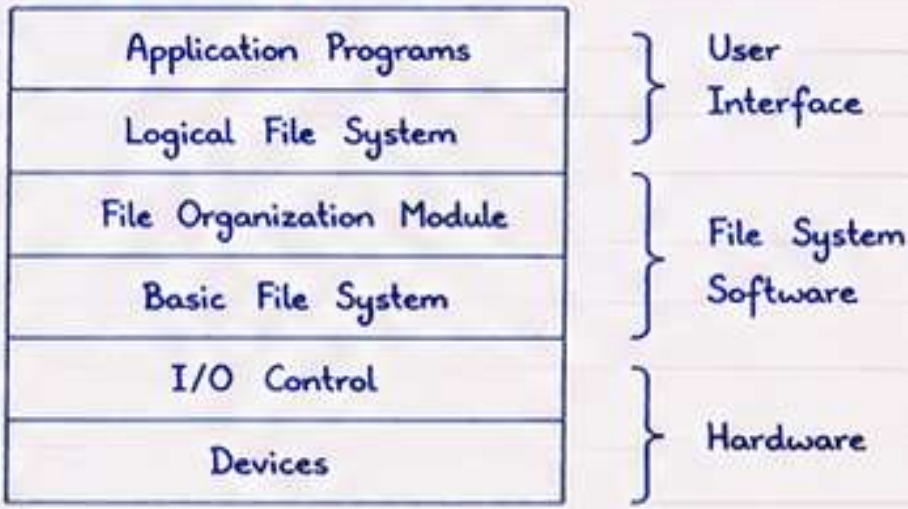
User / File	File 1	File 2	File 3
User 1	R W	R	–
User 2	R	R W	R
User 3	–	R	R W
User 4	R	–	R

R = Read W = Write – = No access

9.7 File System Implementation

A file system resides on secondary storage and manages files efficiently.

File System Layers :



Types of File Systems :

- FAT (File Allocation Table)
- NTFS (New Technology File System)
- ext4 (Linux File System)
- UFS (Unix File System)

9.8 Summary

- File management allows OS to organize and manage data on storage devices.
- Files have attributes and can be accessed using sequential or direct methods.
- Directory structures help in organizing files in a hierarchical manner.
- File protection is essential for data security.
- Efficient file system implementation improves overall system performance.

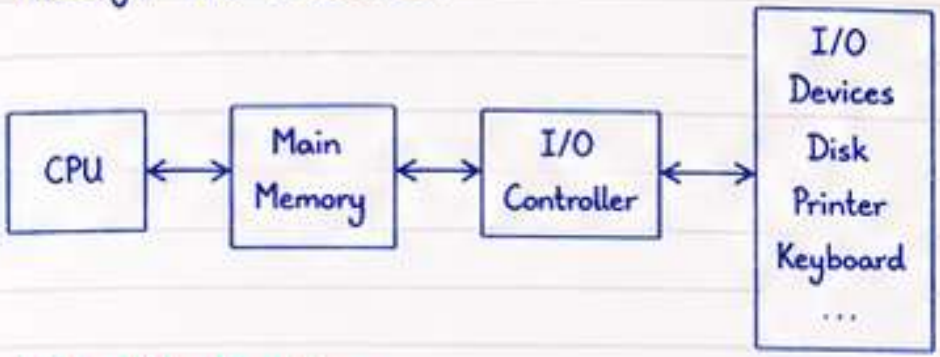
Good file management ensures data security, easy access and better system performance.

10. I/O MANAGEMENT

I/O management is responsible for efficient communication between the CPU, main memory and I/O devices.

10.1 I/O Hardware

I/O devices are connected to the system through I/O controllers.



10.2 I/O Operations

I/O operations include :

- Request I/O – Issue I/O command
- Wait I/O – Wait for device ready
- Process I/O – Transfer data
- Release I/O – Free I/O device

10.3 I/O Methods

- Programmed I/O
- Interrupt-driven I/O
- Direct Memory Access (DMA)

10.4 I/O Data Transfer Techniques

1. Programmed (Polling) I/O

- CPU checks the status of I/O device repeatedly until it is ready.
- Simple but wastes CPU time.

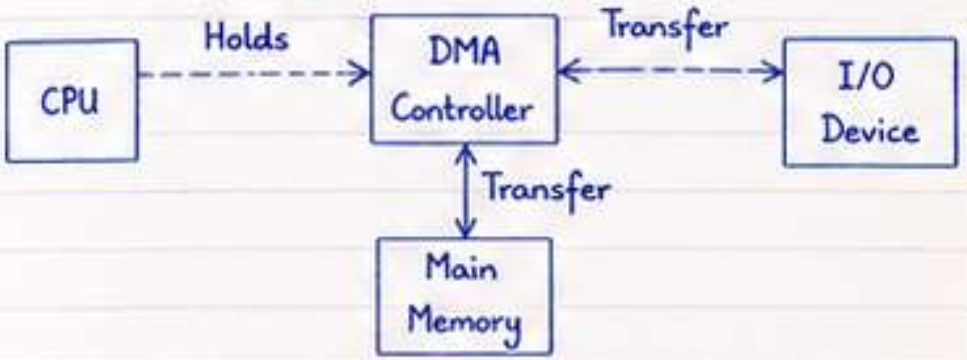
2. Interrupt-driven I/O

- I/O device sends an interrupt when it is ready.
- CPU can perform other tasks meanwhile.

3. Direct Memory Access (DMA)

- Data is transferred directly between I/O device and memory.
- CPU is not involved in data transfer.

10.5 DMA Transfer



DMA improves system efficiency by reducing CPU involvement in data transfer.

11. SECONDARY STORAGE MANAGEMENT

Secondary storage provides large non-volatile storage for the system.

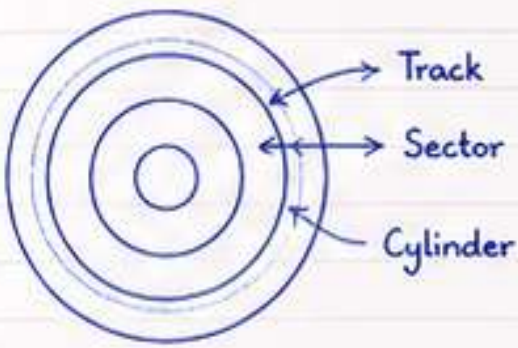
11.1 Characteristics

- Non-volatile
- Large storage capacity
- Slower than primary memory
- Low cost per bit

11.2 Secondary Storage Devices

- Magnetic Disks
- Solid State Drives (SSD)
- Optical Disks (CD, DVD)
- Magnetic Tapes

11.3 Disk Structure



- Track - Circular path on the disk surface.
- Sector - Smallest unit of data.
- Cylinder - Set of tracks at the same radius across all platters.

11.4 Disk Scheduling Algorithms

Disk scheduling decides the order in which disk I/O requests are serviced.

1. FCFS (First Come First Serve)

- Requests are served in the order they arrive.
- Simple but may lead to large seek time.

2. SSTF (Shortest Seek Time First)

- Next request with minimum seek time is serviced.
- Better performance than FCFS.

3. SCAN (Elevator Algorithm)

- The disk arm moves in one direction and services requests.
- When it reaches the end, it reverses direction.

4. C-SCAN (Circular SCAN)

- The arm moves in one direction only.
- After reaching the end, it jumps to the beginning without servicing requests.

12. PROTECTION AND SECURITY

Protection ensures that all system resources are used in a controlled and authorized manner.

12.1 Goals

- Ensure data integrity
- Prevent unauthorized access
- Ensure availability of resources

12.2 Protection Mechanisms

- Authentication – Verify the identity of users.
- Authorization – Allow access to allowed resources.
- Access Control – Restrict access to resources.
- Encryption – Protect data from unauthorized reading.

Security protects the system from external threats like viruses, hackers, and attacks.

12.3 Security Threats

- Viruses
- Worms
- Trojan Horses
- Denial of Service
- Unauthorized Access

13. DISTRIBUTED OPERATING SYSTEMS

A distributed OS manages a collection of independent computers and makes them appear as a single system to users.

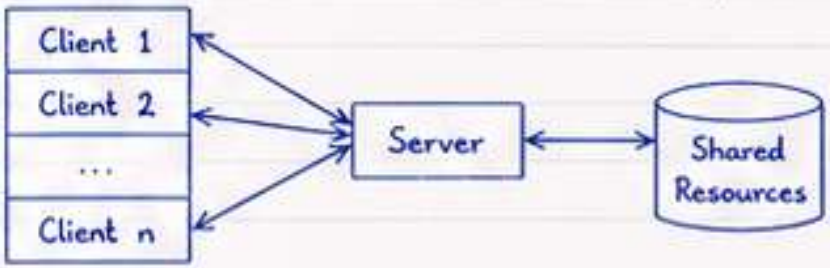
13.1 Features

- Resource sharing
- High reliability
- Scalability
- Load balancing

12.2 Types

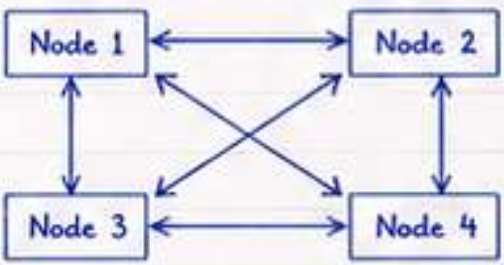
1. Client-Server System

- Servers provide services and clients request them.



2. Peer-to-Peer System

- All nodes act as both client and server.



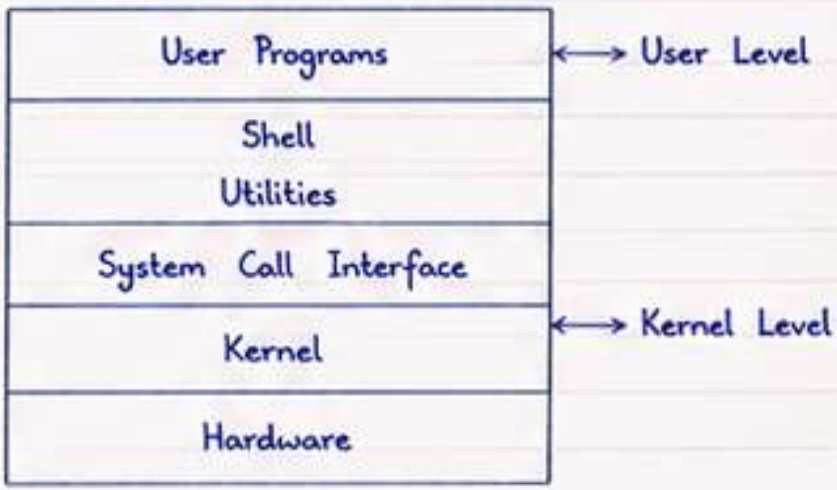
14. UNIX OPERATING SYSTEM

UNIX is a multiuser, multitasking and portable operating system developed in the 1970s.

14.1 Features

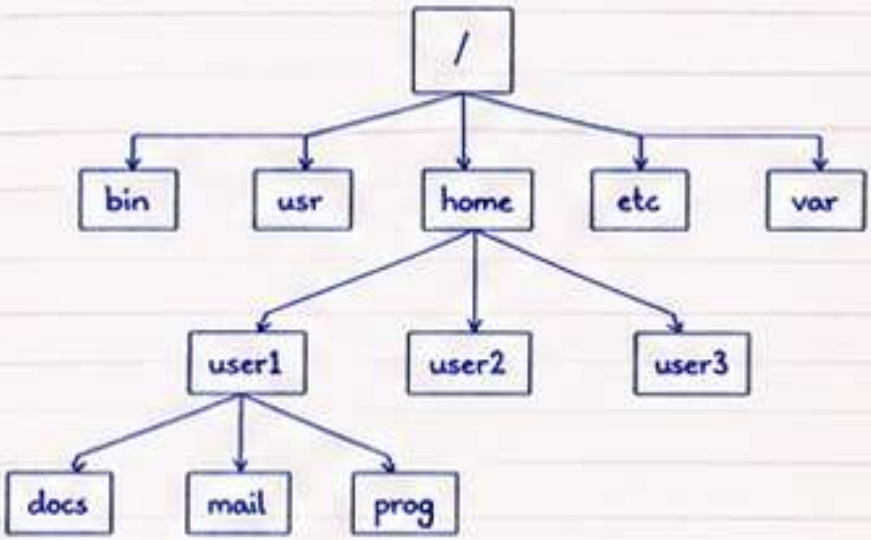
- Multiuser - Many users can work simultaneously.
- Multitasking - Many tasks run at the same time.
- Portability - Can run on different hardware platforms.
- Security - File permissions and owner concept.
- Hierarchical File System - Tree structured directory.

14.2 UNIX Architecture



14.3 UNIX File System Structure

UNIX uses a tree like directory structure starting from the root directory '/'.



14.4 File Permissions

Each file in UNIX has 3 sets of permissions :

- Owner (u)
- Group (g)
- Others (o)

Permissions :

r - Read w - Write x - Execute

Example : -rwxr-xr--

u g o

UNIX provides strong protection and security using file permissions.

14.5 UNIX Commands (Examples)

Command	Description
ls	List files and directories
cd	Change directory
pwd	Print working directory
cp	Copy files
mv	Move or rename files
rm	Remove files
cat	Display file contents
mkdir	Create a new directory
rmdir	Remove an empty directory

14.6 Summary

- UNIX is a powerful and widely used OS.
- It supports multiuser and multitasking.
- It uses a hierarchical file system.
- File permissions ensure security.
- UNIX commands make system management easy and efficient.

15. LINUX OPERATING SYSTEM

Linux is an open source, free and Unix like operating system developed by Linus Torvalds in 1991.

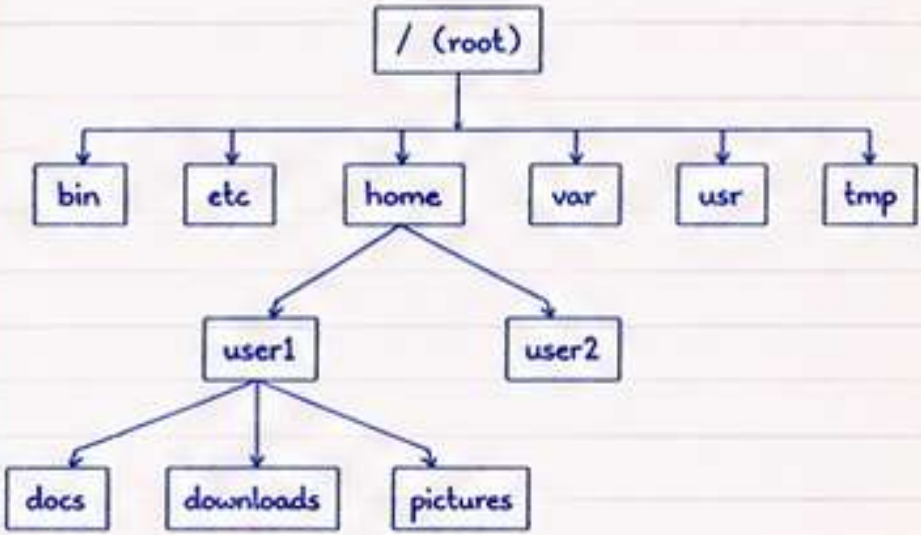
15.1 Features :

- Open Source - Source code is freely available.
- Multiuser and Multitasking
- Secure and Stable
- Supports multiple platforms
- Large community support

15.2 Linux Architecture

15.3 Linux File System Hierarchy

Linux follows a standard directory structure as shown below.



Common Directories :

/bin - Essential user command binaries
/etc - System configuration files
/home - Home directories of users
/var - Variable files (logs, spool, cache)
/usr - User programs and data
/tmp - Temporary files

15.4 Basic Linux Commands

Command	Description
ls	List directory contents
cd	Change directory
pwd	Print working directory
touch	Create an empty file
cp	Copy files or directories
mv	Move or rename files
rm	Remove files
cat	Display file contents
man	Display manual of commands

15.5 Summary :

- Linux is an open source operating system.
- It is secure, stable and widely used in servers.
- Linux supports many distributions like Ubuntu, Debian, Fedora, etc.
- It provides a powerful command line interface.

Linux is the backbone of many systems including servers, mobiles and embedded devices.

16. NETWORKING BASICS IN LINUX

Linux provides powerful networking tools for configuring and troubleshooting networks.

16.1 Important Networking Commands

Command	Description
ifconfig/ip	Display or configure network interfaces
ping	Test connectivity to a host
netstat/ss	Display network connections and statistics
hostname	Show or set system hostname
route	Display or modify routing table
traceroute	Trace the path to a host

16.2 Example

- ping google.com → Test connection to Google
- ifconfig → Show IP address and interface details
- netstat -tuln → Show open ports and listening services

Linux networking tools help in managing and monitoring network connections efficiently.

17. PROCESS MANAGEMENT IN LINUX

A process is an instance of a program in execution.

17.1 Process Commands

Command	Description
ps	Display running processes
top/htop	Monitor processes in real-time
kill	Terminate a process by PID
killall	Terminate processes by name
nice	Start a process with modified priority
renice	Change priority of a running process
bg, fg	Send process to background/ foreground

17.2 Example

- ps aux → Show all running processes
- kill 1234 → Terminate process with PID 1234
- top → Real-time process monitoring

Efficient process management ensures smooth system performance.

18. SHELL SCRIPTING BASICS

Shell scripting allows automation of tasks using shell commands.

18.1 Structure of a Shell Script

```
#!/bin/bash                          → Shebang (Interpreter)
# This is a comment                  → Comment
echo "Hello, World!"                 → Command
VAR=10                               → Variable
echo "Value of VAR is $VAR"          → Use of variable
# End of script                      → Comment
```

18.2 Running a Script

- Save the script with .sh extension
- Give execute permission

  chmod +x script.sh
- Run the script

  ./script.sh

Shell scripts help in automating repetitive tasks and increasing productivity.

19. PACKAGE MANAGEMENT

Package managers help in installiig, updating and removing software.

19.1 Debian / Ubuntu (apt)

Command	Description
sudo apt update	Update package list
sudo apt upgrade	Upgrade installed packages
sudo apt install <pkg>	Install a package
sudo apt remove <pkg>	Remove a package
sudo apt search <pkg>	Search for a package

19.2 Red Hat / Fedora (dnf/yum)

Command	Description
sudo dnf update	Update all packages
sudo dnf install <pkg>	Install a package
sudo dnf remove <pkg>	Remove a package
sudo dnf search <pkg>	Search for a package

Package managers make software management easy and consistent.

20. USER AND GROUP MANAGEMENT

20.1 User Management Commands

Command	Description
useradd <user>	Add a new user
passwd <user>	Set password for user
usermod <user>	Modify user account
userdel <user>	Delete a user
id <user>	Display user information

20.2 Group Management Commands

Command	Description
groupadd <group>	Add a new group
groupdel <group>	Delete a group
usermod -aG <group> <user>	Add user to group
groups <user>	Show groups of user

Proper user and group management ensures system security and access control.

21. SUMMARY

- Linux is a powerful, open source operating system.
- File system in Linux follows a hierarchical structure.
- Commands help in managing files, directories, processes and system.
- Shell scripting automates tasks.
- Networking tools help in managing connections.
- Package managers simplify software management.
- User and group management ensures security.