Operating System

APNA COLLEGE

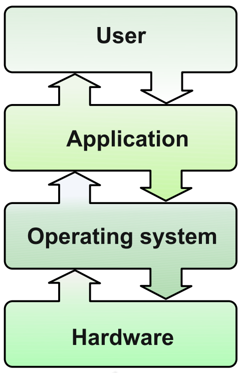
INDEX

⭐ **Important topics**

Introduction to Operating System

1. **What is an operating system?**

   An Operating System is a **system software** that acts as an **intermediary** between **user applications** and **computer hardware**.
   Examples are Windows, Linux, macOS, Android etc.



2. **Why do we need an operating system?**
   - **Complex and Bulky Applications:** Each application would need to include its own code to interact directly with hardware, making **development difficult** and programs **larger than necessary**.
   - **Uncontrolled Resource Usage:** Without an OS, one application could **monopolise CPU**, **memory**, or **I/O devices**, leading to **inefficient** and **unfair** resource utilisation.
   - **Lack of Memory Protection:** Applications could **access** or **overwrite** each other's memory, causing **data corruption** and **system crashes**.

3. **What are application software and system software?**

   Application software: It performs **specific tasks** for the user.
   System software: System software **controls** the computer system and provides a **platform** to run application software.

4. **What are the goals of an operating system?**

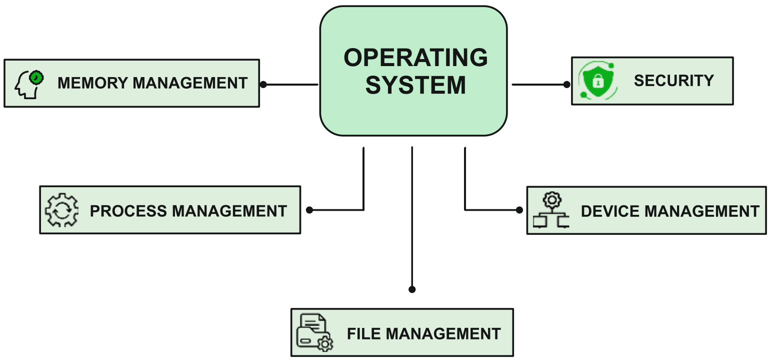
   The following are the goals of an operating system:
   - **convenience** (ease of use)
   - **efficiency** (proper resource allocation)
   - **energy conservation**
   - **minimal** user interference

5. **What are the main functions of an operating system?**

   i. **Process Management:** Handles **creation**, **scheduling**, and **termination** of processes.

   ii. **Resource Allocation:** Manages and distributes **hardware** and **software** resources **fairly** and **efficiently**.

   iii. **Memory Management:** Manages **allocation** and **deallocation** of main memory (RAM) to processes.



   iv. **File System Management: Organises**, **stores**, **retrieves**, and **manages** data on storage devices.

   v. **Device Management: Controls** and **coordinates** input/output devices through drivers.

vi. **Security and Protection:** It protects **data** and **system resources** from unauthorised access.

vii. **Process Scheduling:** Determines which processes run when, **optimising CPU usage**.

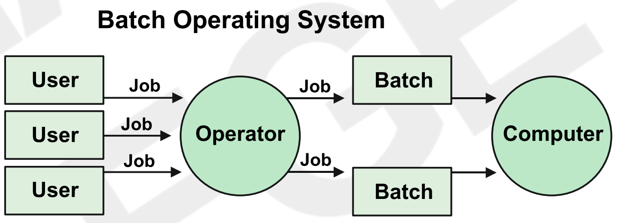
6. History and Evolution

The OS has evolved from **simple batch processing** to **advanced multi-tasking** systems **supporting multiple** users and devices.
Example: Imagine how **telephones evolved**—from **rotary phones** to **smartphones**. Similarly, the **OS evolved**:

i. **Batch systems: No** direct **user interaction**. **Punch cards** are submitted to the **operator**.
ii. **Time-sharing: Multiple users** could use the **computer** at the **same time**.
iii. **GUI-based systems** like **Windows** and **macOS**.
iv. **Modern day:** Advanced OS like **Android**, iOS, cloud-based, and **embedded** OS in **smart-watches** and smart TVs.

7. Types of Operating System

i. **Batch-processing:** This type of OS executes a **batch** of **jobs without manual intervention**. **Jobs** with **similar needs** are **grouped** and **processed together**.

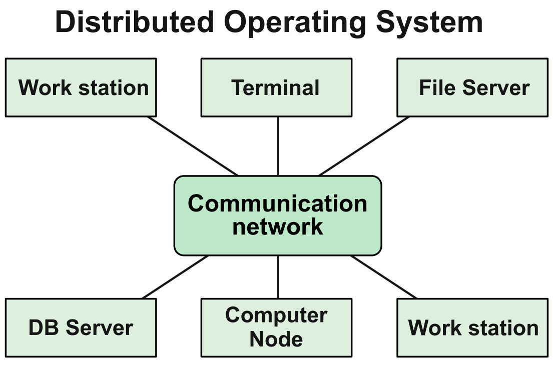


ii. **Multiprogramming:** This OS keeps **multiple programs** in **memory** (RAM) and **executes** them by **switching between them** to utilise the **CPU efficiently**.

iii. **Multitasking:** This OS allows a **single user** to run **multiple programs** at the **same time**, giving the illusion that they're running simultaneously.

iv. **Real-time:** An RTOS responds to **inputs** or **events immediately** with **minimal delay**, often used in critical systems like satellites.

v. **Distributed:** This OS manages a **group** of **computers** and presents them as a **single system** to the **user**, coordinating **processing** and **data sharing across machines**.



8. What is a Kernel?

A kernel is that part of the OS which **interacts directly** with the **hardware** and **performs** the **most crucial tasks**. It is known as the **heart** of the OS.

9. User mode vs Kernel mode.

Kernel mode is a **privileged mode** where the OS has **full access to all hardware and system resources.**
User mode is a **restricted mode** where applications run with **limited access** to system resources to **ensure safety** and **stability**.

10. Different structures of an Operating system?

Different types of structure of the OS:

i. **Monolithic Kernel:** The **entire OS** runs as **one large program** in **kernel mode**. Example: UNIX or Linux

ii. **Microkernel:** Only **essential services** run in **kernel mode**; others run in user mode. Example: Mach, MINIX

iii. **Layered Architecture:** OS is **divided** into **layers**, each built on top of the other. Example: THE

iv. **Modular Architecture:** OS is built using **independent modules** that can be added or removed easily. Example: Linux

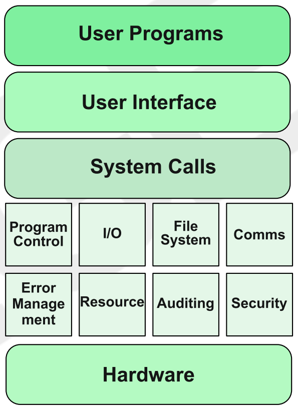
11. What is a system call?

A system call is a mechanism using which a **user program** can **request** a **service** from the **kernel** for which it **does not** have the **permission** to **perform**.

User programs typically do **not** have **permission** to perform operations like **accessing I/O devices** and **communicating** with **other programs**.

System calls are the primary mechanism by which a user-level program can intentionally request services from the kernel, triggering a switch to kernel mode.
Example – fork, exec, getpid, getppid, wait, exit.

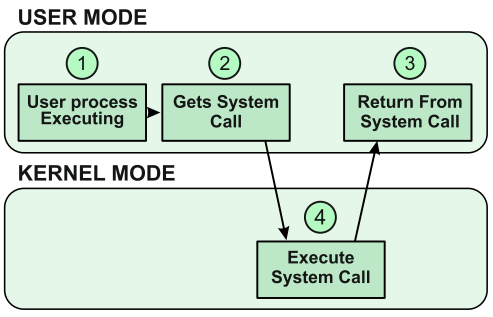
Introduction to System Call



Types of System Calls: (Just remember types; their examples are not important).

i. **Process Control:** System calls that **manage process creation**, **execution**, and **termination**.
- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

WORKING OF A SYSTEM CALL



ii. **File Management:** System calls that **handle file operations** like create, read, write, and delete.
- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

iii. **Device Management:** System calls that **control device access** and **I/O operations.**
- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

iv. **Information maintenance:** System calls that **retrieve** or **set system data** and **process information**.
- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

v. **Communication Management:** System calls that **establish** and **manage communication** between **processes**, either on the **same** or **different machines**.
- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

Type of System Call	Description	Linux Example	Windows
Process Control	Create, execute, and terminate processes	fork(), execve(), exit()	CreateProcess(), ExitProcess()
File Management	Open, read, write, and close files	open(), read(), write()	CreateFile(), ReadFile()
Device Management	Interact with and control hardware devices	ioctl(), read(), write()	SetConsoleMode() ReadConsole() WriteConsole()
Information Maintenance	Get/set system and process-related information	getpid(), uname()	GetProcessId(), GetSystemInfo()
Communication	Enable inter-process communication (IPC)	pipe(), shmget(), socket()	CreatePipe(), CreateNamedPipe()

MCQs:

1. **What is the primary function of an Operating System?**
   A. Store user data
   B. Act as an interface between user and hardware
   C. Perform calculations
   D. Provide internet access

2. **Which of the following is a type of real-time system?**
   A. Windows 10
   B. MS-DOS
   C. Airbag deployment system
   D. Android

3. **In which mode does the OS have full access to hardware?**
   A. User Mode
   B. Safe Mode
   C. Kernel Mode
   D. BIOS Mode

4. **A batch processing OS is best suited for:**
   A. Gaming
   B. Real-time response
   C. Processing payrolls
   D. Video calling

5. **What is multiprogramming?**
   A. Running programs in sequence
   B. Running many programs with only one in memory
   C. Keeping multiple programs in memory to utilize CPU efficiently
   D. Running only one program at a time

6. **A single process OS can?**
   A. Run multiple applications
   B. Only run one program at a time
   C. Schedule tasks concurrently
   D. Support real-time applications

7. **Multitasking allows a user to?**
   A. Switch between OSes
   B. Run multiple programs at once
   C. Use the OS without memory
   D. Access admin features

8. **Which OS structure runs everything in a single large kernel?**
   A. Microkernel
   B. Modular
   C. Monolithic
   D. Layered

9. **Distributed systems are designed to?**
   A. Operate on a single device
   B. Manage databases
   C. Coordinate multiple machines as one system
   D. Replace cloud computing

10. **What is a system call?**
    A. A call made to another computer
    B. An API for games
    C. A way for programs to request services from the OS
    D. A method to access kernel services from user programs

11. **What does a microkernel do differently than a monolithic kernel?**
    A. Runs everything in one block
    B. Provides all services in kernel mode
    C. Runs minimal services in kernel mode
    D. Doesn't use memory

12. **In which mode do applications typically run?**
    A. Admin Mode
    B. Kernel Mode
    C. Safe Mode
    D. User Mode

13. **Which OS type is best for controlling industrial robots?**
    A. Batch OS
    B. Time-sharing OS
    C. Real-Time OS
    D. Multiprogramming OS

14. **An operating system that switches rapidly between tasks is?**
    A. Batch processing
    B. Single user
    C. Time-sharing
    D. Real-time

15. **APIs in OS are used to?**
    A. Store data
    B. Develop UI
    C. Access hardware directly
    D. Allow programs to use OS services

ANSWER KEY

1.	(B)	2.	(C)	3.	(C)	4.	(C)	5.	(C)
6.	(B)	7.	(B)	8.	(C)	9.	(C)	10.	(C)
11.	(C)	12.	(D)	13.	(C)	14.	(C)	15.	(D)

Process Management

1. **What is a program?**

   A program is a **set** of written **instructions** (code) that **tells** a **computer what** to **do**. It is **passive** and **stored** on **disk** until it is executed.

2. **What is a process?**

   A process is an **instance** of a **program** that is **being executed**. It includes the **program code** and its **current activity** (like variables, program counter, registers). It is **stored** in **main memory**.

   Example: In a **movie theater,** the **movie** is like the **program** (static instructions) whereas **screening of the movie** is the **process** (active execution of the movie). We can have **multiple theaters showing the same movie** (multiple processes running the same program), but each has its **own audience, projector, and timing** — just like a process has its own memory, registers, and state.

   A **process comprises** the following:
   - **text section** containing the **program code.**
   - **current activity** represented by the values of the **program counter** and **other registers**
   - **program stack**
   - **data section** containing **global variables**
   - **heap**

3. **Write the difference between a program and a process.**

Aspect	Program	Process
Nature	**Passive** (just code/instructions)	**Active** (executing instance of a program)
Stored In	**Stored** on **disk** (e.g., .exe, .py file)	**Present** in **main memory** (RAM) during execution
Multiplicity	A program is a set of instructions that can be executed multiple times.	**Multiple processes** can run the **same program**

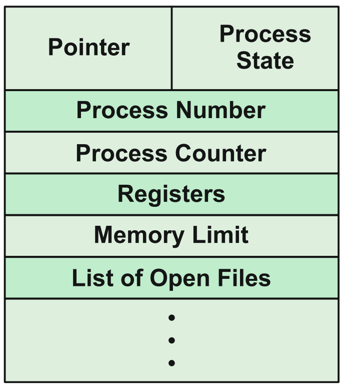
4. **What is a Process control block (PCB)?**

   The PCB **stores all information** about a **process**: process ID, state, program counter, CPU registers, memory limits, etc.

   PCB is stored in protected memory areas inaccessible to user-mode programs. This is because it holds **important information** about the process.

   Example: PCB is like a **hospital file** for a **patient**, it keeps track of patient (process) status, medication (resources), and history.

Process Control Block



5. **Why do we need a Process control block?**

   As the OS supports **multiprogramming**, it needs to **keep track of all active processes**. For this, the **PCB** is **essential**. **Each process** has its **own PCB**.

   Purpose of PCB:
   - **Track Process Execution:** Helps the OS **monitor** and **manage multiple processes** efficiently.
   - **Enable Context Switching: Stores** the **exact state** of a **process** so it can be **paused** and **resumed correctly**.
   - **Resource Management: Maintains details** about **memory**, **file handles**, and **I/O devices** used by the process.

6. **What is a process table?**

   The Process Table is an **array** of **PCBs**, which logically contains a PCB for **all** of the **current processes** in the system.

7. **What is context switching?**

   Context switching is the process of **saving the state** of a **currently** running **process** and **restoring the state** of **another process** that is ready to run. It **enhances** the system's ability to perform **multitasking** effectively.

   How it works:
   - The **CPU saves the context** (registers, program counter, etc.) of the current process into its **PCB**.
   - The **kernel then loads the context** of the next scheduled process from its PCB into the CPU, allowing it to resume execution.

   Need for context switch:
   - **Multitasking Support**
     - Allows the CPU to **switch between multiple processes** or threads so they appear to run **simultaneously**.
     - Essential for running **many applications** at the **same time** (e.g., browser, music player, and text editor).

   - **Fair CPU Sharing**
     - Ensures **fairness** by giving each process a chance to use the CPU.
     - **Prevents** one **process** from **monopolizing** the CPU (important for time-sharing systems).

   - **Better System Responsiveness**
     - Enables **quick switching** to high-priority or interactive tasks (like responding to user input).
     - Keeps the system **responsive** even when **background processes** are **running**.

   - **Handling Interrupts & I/O**
     - When a process is **waiting for I/O**, the OS can switch to another process to avoid CPU idling.
     - Keeps the system **efficient** by using **CPU time** productively.

Nature of Operation:

- It is considered **an overhead**, as no useful work is done for the user during the switch.

Performance Impact:

- The **speed of context switching** depends on several factors, such as:
    - o The number of CPU registers to save/restore
    - o Memory speed
    - o Hardware support for context management

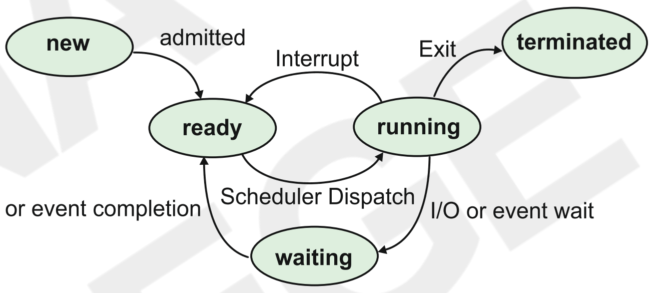
8. Describe the various process states in an OS?

A process can be in one of the following states at a time:

- **New:** The process is being **created** and the OS is setting up resources like memory, PID etc.

- **Ready:** The process is **waiting to be assigned to the CPU**.

- **Running:** The process is **currently being executed** on the CPU.



- **Waiting:** The process is **waiting for some I/O** operation to complete (like file read/write or user input). It cannot proceed until the event finishes.

- **Terminated** (or Exit): The process has **finished execution** or has been **killed** due to an error or user action. Its resources are now released.

- **Suspended:** The process is **temporarily paused**, often by the OS or a user. It can be moved back to the ready or waiting state later.

9. What is a thread?

A **thread** is the smallest unit of execution within a **process**.
It is often called a **"lightweight process"** because it runs independently but shares many things with its parent process.

Example: Imagine a **team working on a group project** (the process).
The entire **team** is like a **process** where each **team member** is a **thread**. They **share** the same tools, whiteboard, and workspace (code, data, files). But each one has their **own tasks, and progress** (stack, program counter, registers).

A thread has different states similar to processes — like Ready, Running, and Waiting, etc.

10. Why do we need Threads?

- **Concurrency: Multiple tasks run** at the **same time** (e.g., typing, auto-save, and formatting in MS Word).
- **Faster Performance:** Threads use **fewer resources** and **switch faster** than full processes.
- **No IPC Needed:** Threads **share** the **same memory**, so they can communicate easily.
- **Multitasking** improves **responsiveness** in applications.

11. What do Threads share?

- Code section
- Data section
- Open files, signals, and other OS resources

12. What is unique to each Thread?

- **Thread ID:** A **unique identifier** assigned to each thread by the OS.
- **Program Counter:** Keeps **track** of the **instruction** being executed.
- **Register Set:** Holds **intermediate** data and **temporary** values.
- **Own Stack:** Stores function calls, return addresses, local vars.

13. Write the difference between a thread and a process.

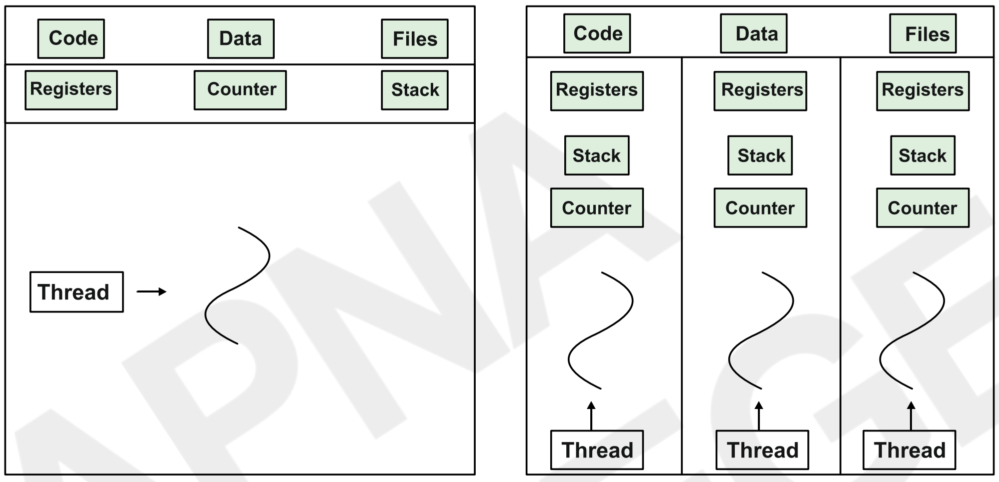
Aspect	Process	Thread
Definition	An **independent** unit of execution with its own memory	A lightweight unit of execution within a process
Memory	Has its **own address space**	**Shares** address space with other threads in the same process
Communication	Inter-process communication (IPC) is **complex & slower**	Communication is **easier and faster** (shared memory)
Context switch	Context switch is slower	Context switch is faster
Creation Time	Creation is slower and involves more OS overhead.	Creation is faster, less overhead
Crash Impact	One process crash **usually doesn't** affect others	One thread crash **can affect the entire process**
Control Block	Uses **Process Control Block (PCB)**	Uses **Thread Control Block (TCB)**

14. Difference between User-Level Thread (ULT) and Kernel-Level Thread (KLT).

Aspect	User-Level Thread (ULT)	Kernel-Level Thread (KLT)
Managed By	User-level **thread library**	**Operating System** (Kernel)
Kernel Awareness	**Kernel** is **unaware** of user threads	**Kernel** is **aware** and **manages** threads
Context Switch	**Fast** (no kernel involvement)	**Slow** (involves system calls to the kernel)
Blocking	If one thread blocks, it can cause the entire process to block.	Only the blocking thread is paused
Scheduling	Done by the user-level library	Done by the kernel
Implementation	Simple to implement	More complex implementation

15. What is multithreading?

Multithreading is a technique that allows a **single process** to run **multiple threads** concurrently, improving **performance** and **responsiveness**. Multithreading is effective on both single-core and multi-core CPUs. On a single-core CPU, it provides concurrency by allowing the CPU to switch between threads, which is useful for keeping applications responsive (e.g., performing I/O in one thread while another continues to run). On a multi-core CPU, it enables true parallelism, where threads can execute simultaneously on different cores.



Single Threaded Process

Multi Threaded Process

16. What are CPU bound and I\O bound processes?

CPU bound: A process that spends **most of its time using the CPU** for **computations** like complex calculations.
It performs **few I/O operations**, takes **long CPU bursts** and **keeps CPU busy**.

I/O bound: A process that spends **more time waiting for I/O operations** (e.g., reading/writing to disk or user input) than **using the CPU**.
It performs **frequent I/O operations**, takes **short CPU bursts** and **keeps CPU idle often**.

MCQs

1. **What is the primary difference between a process and a program?**
   A. A process is stored on disk, while a program runs in memory
   B. A program is an active entity, while a process is a passive entity
   C. A program is a passive entity, while a process is an active entity
   D. Both are active entities

2. **Which of the following best describes a process in an Operating System?**
   A. A system call made by the OS
   B. A static sequence of instructions
   C. An executing instance of a program
   D. A hardware-level operation

3. **Which of the following is not a part of the Process Control Block (PCB)?**
   A. CPU registers
   B. Program counter
   C. Stack pointer
   D. Instruction Set Architecture (ISA)

4. **When a context switch occurs, what happens to the PCB of the currently running process?**
   A. It is deleted
   B. It is updated and stored
   C. It is moved to user space
   D. It is flushed from memory

5. **What is the correct sequence of process states in its lifecycle?**
   A. Ready → Running → Terminated → Waiting
   B. New → Ready → Running → Waiting → Terminated
   C. Running → New → Waiting → Terminated
   D. Ready → Waiting → Running → Terminated

6. **A process is in the waiting state when:**
   A. It is waiting to be assigned to the CPU
   B. It is executing
   C. It is waiting for I/O operation to complete
   D. It is being terminated

7. **Which of the following is true about threads?**
   A. Threads have separate code and data segments
   B. Each thread has its own PCB
   C. Threads share code, data, and open files
   D. Threads have less overhead than processes

8. **Which component is unique to each thread and not shared?**
   A. Code section
   B. Data segment
   C. Open files
   D. Stack

9. **Which of the following statements is true about threads vs processes?**
   A. Context switching between threads is more expensive than between processes
   B. Threads do not support parallelism
   C. Threads are lighter and share the same memory space
   D. Processes are preferred over threads for concurrent tasks

10. **What is one advantage of user-level threads over kernel-level threads?**
    A. Better CPU utilization
    B. Kernel handles scheduling
    C. Faster context switching
    D. Direct access to hardware

11. **In kernel-level threads, the thread management is done by:**
    A. Application developer
    B. Operating System
    C. Compiler
    D. Virtual machine

12. Which multithreading model maps many user threads to one kernel thread?

A. One-to-One

B. Many-to-Many

C. Many-to-One

D. Two-Level Model

13. Which multithreading model allows the OS to create a sufficient number of kernel threads based on demand from user threads?

A. Many-to-One

B. One-to-One

C. Many-to-Many

D. Two-to-One

14. Context switching is:

A. Creating a new process

B. Saving and loading process state during a switch

C. Deleting an existing thread

D. Running two processes simultaneously

15. A CPU-bound process:

A. Spends most of its time doing I/O operations

B. Requires minimal CPU and more I/O

C. Frequently yields the CPU

D. Spends most of its time performing computations

ANSWER KEY									
1.	(C)	2.	(C)	3.	(D)	4.	(B)	5.	(B)
6.	(C)	7.	(C)	8.	(D)	9.	(C)	10.	(C)
11.	(B)	12.	(C)	13.	(C)	14.	(B)	15.	(D)

CPU Scheduling

1. **What is CPU scheduling?**

   It is the process used by the OS to decide **which process in the ready queue should be executed next by the CPU**.
   Since there's usually **one CPU** and **many processes**, the OS uses a scheduler to decide **which process runs next**.

2. **What are the Goals of CPU scheduling?**

   - Maximize CPU utilization and throughput
   - Minimize turnaround time, waiting time and response time
   - Ensure fairness by preventing starvation and giving each process a chance.

3. **Different types of CPU scheduling.**

   There are two types of CPU scheduling:

   i. **Non-Preemptive:** Once a process is given the CPU, it **runs to completion** (or until it wants to perform I/O operation), without being interrupted.
   Example: Simple systems like batch processors.

   Advantages:
   - Simpler and more efficient
   - Less context switching

   Disadvantages:
   - Poor responsiveness
   - One long task can block others

   ii. **Preemptive:** The CPU can be **taken away** from a **currently** running **task** if a **higher-priority** or more appropriate task arrives.
   Example: A short task arrives while a long task is running.

   Advantages:
   - Better response time
   - Fair CPU distribution

   Disadvantages:
   - More overhead (context switches)
   - Harder to implement

4. **Describe the different types of queues used in CPU scheduling.**

   There are primarily three main types of queues.

   i. **Job Queue (Secondary memory)**
   - Contains all processes **submitted to the system**.
   - Some jobs may wait in the job queue before being admitted into RAM by the long-term scheduler.

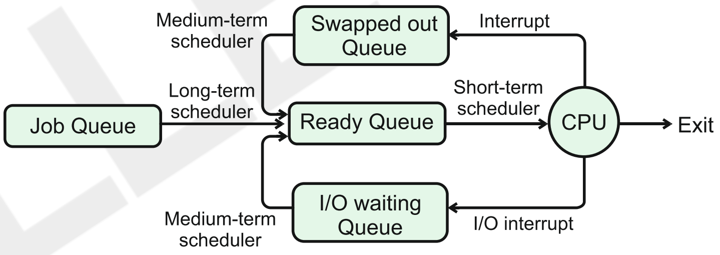
   ii. **Ready Queue (Main memory)**
   - Holds processes **loaded in memory** and **ready to run**.
   - Waiting for CPU allocation.

   iii. **Waiting (Blocked) Queue**
   - Contains processes that are **waiting for I/O** (disk, keyboard, etc.).
   - Will return to the ready queue once I/O is done.

   iv. **Swapped-Out Queue**
   - Processes that are **temporarily moved to disk** (due to memory limitations).
   - Can be swapped back in when memory is available.

5. **What are different types of Schedulers?**

   i. **Long-Term Scheduler (Job Scheduler)**
   - Selects processes from the **job queue** and **moves** to the **main memory** (ready queue).
   - Controls the **degree of multiprogramming** (number of processes in memory).
   - Balances **CPU-bound vs I/O-bound** processes.
   - Runs **infrequently**.



   ii. **Short-Term Scheduler (CPU Scheduler)**
   - Picks a **process** from the **ready queue** for CPU execution.
   - Runs **frequently** (every few milliseconds).

   iii. **Medium-Term Scheduler**
   - Handles **swapping:** suspends and resumes processes.
   - Improves **memory** and **CPU utilization** by managing load.

6. **What is a dispatcher?**

   It is responsible for **giving control of the CPU** to the process selected by the **short-term scheduler**. It plays a key role in **process switching and multitasking**.

   Performs:
   - **Context switching** (saving and restoring process states).
   - Switching from **kernel mode** to **user mode.**
   - Jumping to the starting instruction of the selected process.

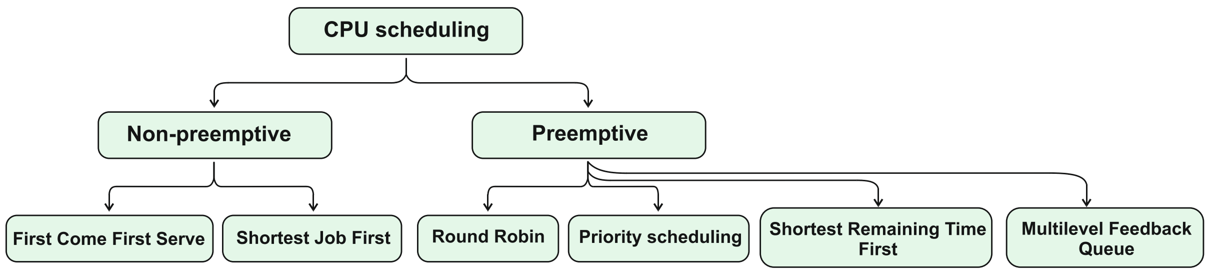
7. **Describe the important terminologies in CPU scheduling.**

- **Arrival time (AT):** The time at which a process **enters the ready queue**.
- **Burst time (BT):** The time required by a process to **complete its execution** on the CPU.
- **Completion time (CT):** The time at which a process **finishes its execution completely**.
- **Turnaround time (TAT):** Total time taken from **arrival to completion** of a process.
  - Turnaround time = Completion time - Arrival time
- **Waiting time (WT):** Total time a process spends **waiting for CPU** to be assigned.
  - Waiting time = Turnaround time - Burst time
- **Response time (RT):** The time from when a process arrives to when it **starts executing** for the first time.
- **Throughput:** The number of processes **completed per unit of time** by CPU. Higher throughput is equivalent to better system performance.
- **CPU utilization: Percentage** of time the **CPU** is **actively working**.
- **Dispatch latency:** The time taken by the **dispatcher** to **stop one process** and **start another**.
- **Gantt chart:**
  - It is a **visual timeline** that shows the **order and duration** in which processes are executed by the CPU.
  - It helps to **understand the execution flow**, calculate turnaround time, waiting time, and response time.

8. **What are the needs of CPU Scheduling Algorithms?**

- Only **one process** can use the **CPU at a time**, so **scheduling decides** which **process runs next**.
- Helps in **efficient CPU utilization** by **reducing idle** time.
- Aims to **improve performance** metrics like waiting time, turnaround time, and response time.
- Ensures **fairness among processes** and **avoids starvation**.
- Supports **multitasking** and handles different types of workloads.
- Allows **priority handling** where important tasks are given preference.

9. **Describe different types of CPU scheduling algorithms with their advantages and disadvantages.**



i. **First-Come, First-Served (FCFS):** Processes are scheduled in the **order they arrive** in the **ready queue.**

Advantages:

- Simple, easy to implement and fair (in terms of arrival time).

Disadvantages:

- **Convoy effect:** Short processes get stuck behind long ones.
- High average waiting time.

ii. **Shortest Job First (SJF):** Schedules the process with the **smallest burst time** first.

Advantages:

- Optimal for **minimizing** average **waiting time**.

Disadvantages:

- **Requires** knowledge of **burst times**.
- Causes **Starvation:** Long processes may be delayed indefinitely.

iii. **Shortest Remaining Time First (SRTF):** Pre-emptive version of SJF; runs the process with the **least remaining time**.

Advantages:

- Best for minimum average turnaround time.

Disadvantages:

- Frequent **context switching.**
- **Starvation** of longer processes.

iv. **Round Robin (RR):** Each process gets a **fixed time quantum** in a cyclic order.

Advantages:

- Good for **time-sharing** systems and **all processes** get **equal CPU time**.

Disadvantages:

- High context switching overhead if time quantum is small.
- Poor performance if quantum is too small or too large.

v. **Priority Scheduling:** Each process is assigned a **priority**, and the CPU is given to the highest-priority process.

Advantages:

- Handles important tasks first.

Disadvantages:

- **Starvation** of low-priority processes and may require **aging** to prevent starvation.

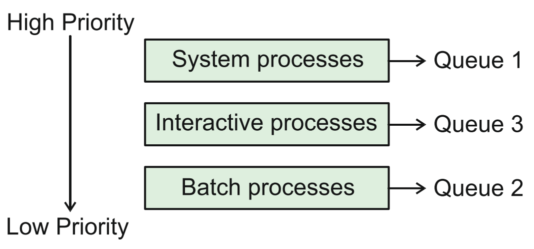
vi. **Multilevel Queue Scheduling: Multiple queues** for **different process types** (e.g., system, user), each with its own scheduling algorithm.

Advantages:

- Separates different types of processes and is customizable for system needs.

Disadvantages:

- Multilevel queue is rigid because processes are not allowed to move between queues.
- Possible **starvation** of lower-priority queues.



vii. **Multilevel Feedback Queue (MLFQ):** Similar to multilevel queue, but **processes can move between queues** based on behavior and aging.

Advantages:

- Highly **flexible** and **reduces starvation** through feedback and aging.

Disadvantages:

- **Complex** to implement and tune.
- Needs **careful parameter** design (quantum, levels).

10. **Find out completion time, turnaround time, waiting time and response time for each process for each of the algorithms.**

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	2
P4	3	1

First-come, first-serve

Gantt chart: P1(0-8) -> P2(8-12) -> P3(12-14) -> P4(14-15)

Process	AT	BT	CT	TAT	WT	RT
P1	0	8	8	8	0	0
P2	1	4	12	11	7	0
P3	2	2	14	12	10	0
P4	3	1	15	12	11	1

Avg CT: 12.25, Avg TAT: 10.75, Avg WT: 7, Avg RT: 8.5.

Shortest Job First

NOTE: non-preemptive SJF and works only on processes available at the time of scheduling.

Gantt chart: P1(0-8) -> P4(8-9) -> P3(9-11) -> P2(11-15)

Process	AT	BT	CT	TAT	WT	RT
P1	0	8	8	8	0	0
P2	1	4	15	14	10	0
P3	2	2	11	9	7	0
P4	3	1	9	6	5	1

Avg CT: 10.75, Avg TAT: 9.25, Avg WT: 5.5, Avg RT: 7.

Shortest Remaining Time First
Gantt chart: P1(0-1) -> P2(1-2) -> P3(2-4) -> P4(4-5) -> P2(5-8) -> P1(8-15)

Process	AT	BT	CT	TAT	WT	RT
P1	0	8	15	15	7	0
P2	1	4	8	7	3	0
P3	2	2	4	2	0	0
P4	3	1	5	2	1	1

Avg CT: 8, Avg TAT: 6.5, Avg WT: 2.75, Avg RT: 1.75.

Round Robin
Gantt chart: P1(0-2) -> P2(2-4) -> P3(4-6) -> P4(6-7) -> P1(7-9) -> P2(9-11) -> P1(11-15)

Process	AT	BT	CT	TAT	WT	RT
P1	0	8	15	15	7	0
P2	1	4	11	10	6	1
P3	2	2	9	7	5	2
P4	3	1	7	4	3	3

Avg CT: 10.5, Avg TAT: 9.75, Avg WT: 5.25, Avg RT: 3.

MCQs

1. **Which of the following determines the next process to be executed on the CPU?**
   A. Long-term scheduler
   B. Short-term scheduler
   C. Dispatcher
   D. Medium-term scheduler

2. **Which of the following is true about CPU-bound processes?**
   A. Spend most of the time waiting for I/O
   B. Always have higher priority
   C. Perform most of their operations using the CPU
   D. Require minimal CPU usage

3. **Which component gives control of the CPU to the process selected by the scheduler?**
   A. Long-term scheduler
   B. Dispatcher
   C. CPU controller
   D. Ready queue

4. **Which scheduling algorithm may lead to the convoy effect?**
   A. First-Come, First-Served
   B. Shortest Job First
   C. Round Robin
   D. Priority Scheduling

5. **Which algorithm gives the minimum average waiting time if all burst times are known?**
   A. FCFS
   B. Shortest Job First
   C. Round Robin
   D. Priority Scheduling

6. **In Round Robin, if the time quantum is too small, it leads to:**
   A. Convoy effect
   B. Process starvation
   C. High context switching overhead
   D. Better performance

7. **Which algorithm is preemptive and selects the process with the least remaining burst time?**
   A. FCFS
   B. SJF
   C. Shortest Remaining Time First (SRTF)
   D. Round Robin

8. **Which scheduling algorithm may cause starvation for low-priority processes?**
   A. FCFS
   B. Round Robin
   C. Priority Scheduling
   D. SJF

9. **The scheduler that decides which job to admit into the system is called:**
   A. Long-term scheduler
   B. Short-term scheduler
   C. Medium-term scheduler
   D. Dispatcher

10. **Which queue stores processes that are waiting to be assigned to the CPU?**
    A. Job Queue
    B. Ready Queue
    C. Waiting Queue
    D. Device Queue

11. **Processes waiting for I/O are placed in the:**
    A. Ready Queue
    B. Waiting Queue
    C. Swapped Queue
    D. CPU Queue

12. **Which scheduler may suspend and swap out processes to manage memory?**
    A. Long-term
    B. Medium-term
    C. Short-term
    D. I/O scheduler

13. **In Multilevel Feedback Queue, processes can:**
   A. Only run in one fixed queue
   B. Not change priority
   C. Move between queues based on behavior
   D. Only run once

14. **Which scheduling algorithm is best for time-sharing systems?**
   A. FCFS
   B. Round Robin
   C. SJF
   D. Priority Scheduling

15. **A major disadvantage of SJF is:**
   A. High overhead
   B. Starvation of longer processes
   C. Low CPU utilization
   D. Low throughput

16. **A Gantt chart is used to show:**
   A. Memory allocation
   B. I/O operations
   C. Process execution order over time
   D. Cache usage

17. **Which of the following is NOT a CPU scheduling criterion?**
   A. Turnaround time
   B. CPU utilization
   C. Throughput
   D. Disk space

18. **Aging in scheduling is used to:**
   A. Refresh old processes
   B. Prevent starvation
   C. Increase time quantum
   D. Remove finished jobs

ANSWER KEY									
1.	(B)	2.	(C)	3.	(B)	4.	(A)	5.	(B)
6.	(C)	7.	(C)	8.	(C)	9.	(A)	10.	(B)
11.	(B)	12.	(B)	13.	(C)	14.	(B)	15.	(B)
16.	(C)	17.	(D)	18.	(B)				

Process Synchronization

1. **What are Cooperating process and independent processes?**

   Cooperating process: A process that **shares data, resources, or communicates** with one or more other processes.
   Example: **Producer-consumer** problem, where one process produces data and another consumes it.

   Independent process: A process that **does not share data or resources** with any other process. **No synchronization** is needed.
   Example: A music player app running while you type in a text editor. They do **not interact,** so they're independent.

2. **What is process synchronization?**

   Synchronization is the **coordination** of **processes** so they **don't interfere** with each other when **accessing shared resources** (e.g., memory, variables, files). **Without it,** two or more processes could **change shared data simultaneously**, leading to **inconsistent** or **incorrect** results.

3. **When does race condition occur?**

   Race condition occurs when **multiple processes access** and **manipulate** the **same data concurrently**. In this condition, the **final outcome depends** on the **order** of **execution**, which is **unpredictable**. There are software as well as hardware solutions to this problem.

4. **What is a critical section?**

   It is a **part** of the **code** where **shared resources** are **accessed**. Only **one process** should **execute** its **critical section** at a time to **avoid race conditions.**

5. **What are different types of process synchronization?**

   We can divide it into two main types:

   - **Mutual Exclusion Synchronization:** It ensures that **only one process/thread** can access a **critical section** (shared resource) at a time.
     - **Techniques:** Peterson's/Bakery, Mutex, Semaphores, Monitors, Spinlocks, Hardware Instructions.
   - **Condition Synchronization: Waiting** for an **event** or a certain **condition** in **shared data** and **waking** up a **waiting process** when the **condition** becomes **true**.
     - **Techniques:** Conditional variables, Semaphores.

☆ **6. What are the conditions to design a process synchronization algorithm?**

The algorithm must satisfy the following **three essential conditions**:

- **Mutual Exclusion:** Only **one process** should be allowed to enter its **critical section** at a time. It **prevents race conditions** and **ensures data consistency**.
- **Progress:** The **decision** of which **process enters** the **critical section** should **not depend on processes** that are **not interested** and the decision should **eventually be made**, not postponed forever.
- **Bounded Wait:** After a process has made a request to enter its critical section, there must be a **limit on the number of times** other processes are allowed to enter their critical sections **before this process gets its turn**.
- **No busy waiting (Optional):** Processes should **not waste CPU cycles** continuously checking if they can enter the critical section (unlike spinlocks).

7. What are the common solutions to the critical section problem?

i. **Peterson's Solution:** A **software-based** solution for two-process synchronization. It ensures **mutual exclusion**, **progress,** and **bounded waiting**. Peterson's solution works **only for two processes** and assumes atomic memory access.

How it works (for processes P0 and P1):

- Uses two variables:
  - **flag [2]:** To indicate if a process wants to enter the critical section.
  - **turn:** To decide whose turn it is.
- **Each process** sets its **flag** to **true** and gives the turn to the other.
- It waits if the other process also wants to enter and it's their turn.

ii. **Bakery Algorithm:** Generalizes Peterson's solution for multiple processes. It is inspired by "take-a-number" systems in bakeries.

How it works:

- Each process takes a number.
- The process with the smallest number gets access to the critical section.
- Ensures fairness and mutual exclusion.

☆ iii. **Semaphores:** Semaphores are synchronization tools (like counters) used to control access to shared resources.

Types:

- **Counting Semaphore:** It can have **any non-negative integer value**. It is used when multiple resources are available (e.g., 5 printers).
- **Binary Semaphore:** Only 0 or 1. Works like a **lock**: 0 means locked, 1 means available.

Operations:

- wait() (or P): Decrease value; if it's negative, the process is blocked.

```
wait(S):
   while S <= 0; // busy wait
   S = S - 1;
```

In practice, semaphores block the process instead of busy waiting — this code is for concept only.

- signal() (or V): Increase value; wakes up a blocked process if any.

```
signal(S):
            S = S + 1;
```

These are **atomic operations**, they cannot be interrupted.

8. What are hardware synchronization algorithms?

Hardware provides **low-level** mechanisms to **enforce mutual exclusion** directly using special machine instructions. These are **fast** and **do not rely on high-level constructs** like semaphores or monitors.

- **Disable interrupts:** A process **disables all interrupts** while in the critical section. It **prevents context switching**, so no other process can interrupt and enter the critical section.
  - **Simple** and **effective** for a **single-CPU system**.
  - **Not** suitable for **multi-core systems.**
  - **Disabling interrupts** for **too long affects** system **responsiveness**.
- **Test and Set:** A special hardware instruction that **tests** a memory location and **sets** it (like flipping a lock).
  - If the lock is already taken, the process keeps checking (spinning) until it's free.
  - It causes busy waiting, which leads to CPU time being wasted if lock is not acquired quickly.
- **Compare and Swap:** This instruction compares the content of a memory location to a given value and, if they match, **swaps** it with a new value.
  - Very useful in **lock-free programming**.
  - Can still result in busy-waiting if not combined with other mechanisms.
  - These atomic operations may still cause livelock or priority inversion if not combined with scheduling mechanisms.

9. What is a monitor?

A **Monitor is a high-level synchronization construct** used in OS and multithreaded programming that combines mutual exclusion and condition synchronization using condition variables.
It ensures that:

- **Only one thread/process** executes a monitor function at a time **(mutual exclusion)**,
- **Condition variables** are used to make threads **wait** or **signal** each other when certain conditions are met.

10. What is a condition variable?

A **condition variable** is used **inside a monitor** to let a thread or process:

- **Wait** (pause and release the monitor),
- And let other threads **signal** (notify) it when a certain condition is true.

☆ 11. What are common synchronization problems?

The following are the common synchronization problems along with their core issues and solutions using semaphores:

i. **Producer-Consumer:**

a. **Problem:**
- A producer puts data into a **bounded buffer**.
- A consumer takes data from it.
- Producer must **wait if the buffer is full**, and the consumer must **wait if the buffer is empty**.

b. **Semaphores Used:**
- mutex = 1 → to protect critical section (buffer access).
- empty = N → counts empty slots (N = buffer size).
- full = 0 → counts filled slots.

c. **Code snippet:** https://codeshare.io/GLOkKa

ii. **Dining Philosophers:**

a. **Problem:**
- 5 philosophers sit at a table and alternate between thinking and eating.
- A philosopher needs two forks (left and right) to eat and there are 5 forks present.
- If all pick up one fork, they get stuck = deadlock.

b. **Semaphores Used:**
- mutex → to modify states.
- semaphore[i] → one per philosopher to control access to forks.
- States: THINKING, HUNGRY, EATING.

c. **Code snippet:** https://codeshare.io/G73wDG

iii. **Reader-writer:**

a. **Problem:**
- Multiple readers can read at the same time.
- Only one writer can write, and no readers during writing.
- Two variants:
  - o Reader-priority (can starve writers)
  - o Writer-priority (avoids writer starvation)

b. **Semaphores Used:**
- mutex = 1 → to update read Count.
- write Lock = 1 → lock for writing.
- read Count = 0 → number of active readers.

c. **Code snippet:** https://codeshare.io/2WbxYG

Problem	**Goal**	**Semaphore Strategy**
Producer-Consumer	Balance buffer access	mutex, empty, full
Dining Philosophers	Avoid deadlock & starvation	mutex, individual semaphore[i], state testing
Reader-Writer	Allow multiple readers or 1 writer	mutex, write Lock, track read Count

MCQs

1. **What is the main purpose of process synchronization in an operating system?**
   A. To increase memory size
   B. To avoid deadlocks
   C. To ensure correct sequence of execution when processes share resources
   D. To reduce context switch time

2. **Which of the following describes a race condition?**
   A. Two processes access different resources simultaneously
   B. A situation where concurrent access to shared data leads to unpredictable results depending on timing.
   C. A deadlock due to two processes waiting for each other
   D. A starvation scenario where one process never gets CPU time

3. **In process synchronization, the Critical Section is:**
   A. The part of the OS code that handles system calls
   B. The segment where shared resources are accessed
   C. The waiting queue of the process
   D. The code segment responsible for I/O operations

4. **Which of the following is not a condition for a solution to the Critical Section Problem?**
   A. Mutual Exclusion
   B. Deadlock Prevention
   C. Progress
   D. Bounded Waiting

5. **Peterson's Solution works for how many processes?**
   A. Only one
   B. Two
   C. Any number of processes
   D. Exactly four

6. **The Bakery Algorithm is mainly designed to:**
   A. Eliminate starvation in CPU scheduling
   B. Implement non-preemptive scheduling
   C. Ensure fair access to the critical section
   D. Detect and recover from deadlock

7. **A binary semaphore can have how many possible values?**
   A. Any non-negative integer
   B. 0 and 1
   C. 1 and 2
   D. 0 and -1

8. **What distinguishes a counting semaphore from a binary semaphore?**
   A. Binary semaphore allows multiple processes; counting allows only one
   B. Counting semaphore uses only 0 and 1
   C. Counting semaphore allows multiple instances of a resource
   D. Binary semaphore is implemented using mutex

9. **Which of the following hardware synchronization methods involves disabling interrupts?**
   A. Compare and Swap
   B. Test and Set
   C. Spinlock
   D. Disable Interrupts

10. **The Test-and-Set instruction is used to:**
    A. Enable interrupt handling
    B. Avoid deadlocks
    C. Set a flag to true and return its old value
    D. Schedule CPU for I/O bound processes

11. In the Producer-Consumer problem, the shared buffer is used to:
- A. Store OS logs
- B. Communicate between kernel and hardware
- C. Synchronize output to printer
- D. Pass data between two processes using shared memory

12. What is the role of a mutex in synchronization?
- A. Used only for file protection
- B. Ensures exclusive access to shared resources
- C. Used to count waiting processes
- D. Terminates processes after execution

13. A spinlock is most suitable when:
- A. Processes wait for a very long time
- B. Lock is expected to be held briefly
- C. Context switch overhead is negligible
- D. Multithreading is not supported

14. A monitor in operating systems is:
- A. A physical device to watch processes
- B. A low-level hardware synchronization tool
- C. A high-level synchronization construct that provides mutual exclusion and condition synchronization
- D. Used only in single-threaded environments

15. What is the purpose of a condition variable inside a monitor?
- A. To lock critical sections
- B. To keep track of CPU-bound processes
- C. To block a process until a certain condition is true
- D. To switch the process to kernel mode

ANSWER KEY

1.	**(C)**	**2.**	**(B)**	**3.**	**(B)**	**4.**	**(B)**	**5.**	**(B)**
6.	**(C)**	**7.**	**(B)**	**8.**	**(C)**	**9.**	**(D)**	**10.**	**(C)**
11.	**(D)**	**12.**	**(B)**	**13.**	**(B)**	**14.**	**(C)**	**15.**	**(C)**

Deadlock

1. **What is a Deadlock?**

A deadlock is a situation in an OS where a set of processes become **permanently blocked** because **each process is waiting for a resource** that another process is holding.

Example: Imagine two processes: **P1** holds **Resource A** and is waiting for **Resource B**, at the same time **P2** holds **Resource B** and is waiting for **Resource A**. Since neither can proceed without the other releasing the resource, both are stuck → **Deadlock**.

2. **What are necessary conditions for a deadlock to occur?**

All four conditions must hold simultaneously for a deadlock to occur:

- **Mutual Exclusion: At least one resource** must be **held** in a **non-shareable mode**.
- **Hold and Wait:** A **process** is **holding at least one resource** and **waiting** to acquire **additional resources.**
- **No Preemption: Resources cannot** be **forcibly taken** from a **process** holding them.
- **Circular Wait:** A set of processes are waiting for resources held by others in a circular chain.

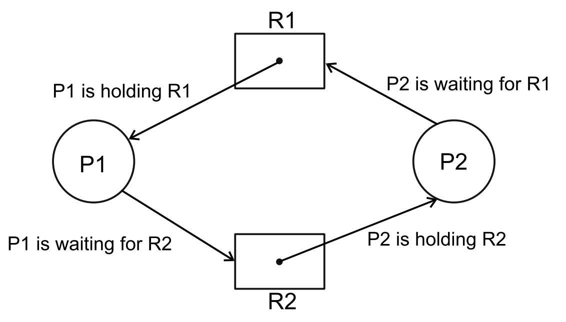
3. **What is a Resource Allocation Graph (RAG)?**

It is a **directed graph** used to **represent** the **state** of **resource allocation** in a system. It shows:

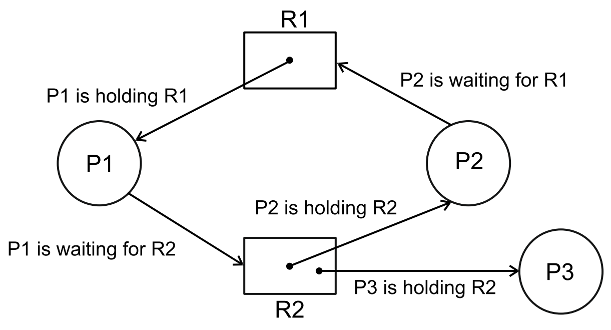
- **Processes requesting resources**
- **Resources assigned** to processes

Components of RAG

i. **Nodes (Vertices)**
There are two types of nodes:
   - **Process nodes (P_1, P_2, ..., Pn):** Represent the processes in the system.
   - **Resource nodes (R_1, R_2, ..., Rm):** Represent the resources in the system.
      - **Single Instance Type Resource:** It refers to a type of resource in the system that has only **one available instance**. In a system with single-instance resources, the presence of a cycle indicates a deadlock.
      - **Multi-Resource Instance Type Resource:** It refers to a type of resource that has **multiple instances available**. In multiple-instance resource systems, a cycle may or may not indicate a deadlock.



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK



MULTI INSTANCE WITHOUT DEADLOCK

ii. **Edges (Directed Arrows)**
There are two types:
- **Request Edge:** P → R (from process to resource) which means process P is requesting resource R.
- **Assignment Edge:** R → P (from resource to process) which means resource R is allocated to process P.

About Multi-Instance Resources:
- Resource nodes **R** have a label showing the **number of available instances** (e.g., R1(2)).

4. What are various ways to handle a deadlock?

Deadlock can be handled in four main ways in Operating Systems.

i. **Deadlock Prevention:** Ensure that at least one of the four necessary conditions for deadlock **never occurs**.
- **Eliminate Mutual Exclusion**: Use sharable resources (not always practical).
- **Eliminate Hold and Wait**: Require processes to request all resources at once.
- **Eliminate No Preemption**: Allow forcibly taking a resource from a process.
- **Eliminate Circular Wait**: Impose a global order on resource allocation.
- **Advantage:** Deadlock is guaranteed not to occur.
- **Disadvantage:** Can lead to **low resource utilization** and **reduced concurrency**.

ii. **Deadlock Avoidance:** Dynamically decide whether resource allocation is safe.
- Use **Banker's Algorithm** (for multiple instances of resources).
- Only grant resource requests **if it leads to a safe state**.
- **Advantage:** Deadlock is avoided without unnecessarily denying resources.
- **Disadvantage:** Requires knowledge of **maximum resource demand**, and involves **complex checks** at runtime.

iii. **Deadlock Detection and Recovery:** Let deadlock occur, then detect and recover from it.
- Run a **deadlock detection algorithm** periodically.
  - Use Resource Allocation Graph (RAG) for single instances.
  - Use Banker's algorithm for multiple instances.
- On detection:
  - Terminate processes involved.
  - Or preempt resources and roll back.
- **Advantage:** Higher resource utilization.
- **Disadvantage:** Deadlock **may impact system performance** until recovery.

iv. **Ignore the Problem (Ostrich Algorithm):** Do nothing and assume deadlock will not happen.
- Used in systems like Windows or Linux when:
  - Deadlocks are very rare.
  - Overhead of prevention/detection is not justified.
- **Advantage:** Simple and no overhead.
- **Disadvantage:** Deadlocks can cause serious issues if they do happen.

☆ 5. What is Banker's Algorithm?

It is a **resource allocation** and **deadlock avoidance algorithm** used in systems where resources have **multiple instances**.
Goal: To ensure the system never enters an unsafe state (i.e., a state that could potentially lead to deadlock).

Terminology & Data Structures:

- **n processes:** P0, P1, ..., Pn−1
- **m resource types:** R0, R1, ..., Rm−1
- **Available[m]:** Number of available instances of each resource type.
- **Max[n][m]:** Maximum demand of each process for each resource.
- **Allocation[n][m]:** Resources currently allocated to each process.
- **Need[n][m]:** Remaining resources each process may still request. Calculated as: Need = Max – Allocation

Banker's Algorithm steps:

i. **Input initial values**
   a. Input values for Available, Max and Allocation.
   b. Compute Need = Max - Allocation.

ii. **Safety check (Is the system in a safe state?)**
   a. Initialize
      - Work = Available
      - Finish[i] = false for all i (i.e., no process is finished yet)
   b. Find a process P_i such that:
      - Finish[i] == false
      - Need[i] ≤ Work (element-wise comparison)
      - If found, pretend to allocate its remaining resources:
         - Work = Work + Allocation[i]
         - Finish[i] = true
      - Repeat step 2
   c. If all processes can finish (Finish[i] == true for all i):
      - The system is in a safe state.
   d. If no such process is found:
      - The system is in an unsafe state (allocation must be denied).

iii. **Resource Request Algorithm (For runtime requests)**
   a. When a process Pi makes a request Request[i]:
      - Check: **Request[i] ≤ Need[i].** If not, error (process is asking for more than declared).
      - Check: **Request[i] ≤ Available**. If not, the process must wait.
      - **Pretend Allocation:**
         - Available -= Request[i]
         - Allocation[i] += Request[i]
         - Need[i] -= Request[i]
      - Run safety check (step 2)
         - If the system is safe: allow allocation.
         - If unsafe: roll back the changes and make the process wait.

Advantage:

- Ensures system safety.
- Prevents deadlock proactively.

Disadvantage:

- Requires advance knowledge of **maximum needs**.
- Can be **computationally expensive** for large systems.
- Not practical in all real-world OS implementations.

Code of banker's algorithm: https://codeshare.io/G6BqpG

MCQs

1. **Which of the following best defines a deadlock in an operating system?**
   A. A condition where a process uses excessive CPU time
   B. A set of processes are waiting for each other indefinitely for resources
   C. When all system resources are used simultaneously
   D. When a process completes execution without releasing resources

2. **Which of the following is not one of the necessary conditions for deadlock to occur?**
   A. Mutual Exclusion
   B. Preemption
   C. Hold and Wait
   D. Circular Wait

3. **In the Hold and Wait condition for deadlock, a process:**
   A. Must request all resources at once
   B. Is preempted from all held resources
   C. Is holding at least one resource and waiting to acquire others
   D. Cannot proceed unless all resources are released

4. **Which of the following strategies ensures deadlock prevention by avoiding Hold and Wait?**
   A. Request all resources at once before execution
   B. Allow circular wait to form
   C. Use a timer to preempt processes
   D. Randomly assign resources to processes

5. **Which approach involves periodically checking for deadlocks by analyzing the system state?**
   A. Deadlock Avoidance
   B. Deadlock Detection
   C. Deadlock Prevention
   D. Deadlock Ignorance

6. **What data structure is used in deadlock detection when each instance of a resource type is single?**
   A. Wait-for Graph
   B. Linked List
   C. Resource Matrix
   D. Allocation Table

7. **Which statement correctly describes the Banker's Algorithm?**
   A. It is used to detect circular wait conditions
   B. It allocates resources to maximize CPU utilization
   C. It ensures the system stays in a safe state before granting a request
   D. It kills a process immediately if it causes a deadlock

8. **A system is said to be in a safe state if:**
   A. Deadlock is currently occurring
   B. No process is executing
   C. There exists a sequence to allocate resources such that all processes complete
   D. At least one process has no allocated resource

9. **In the context of deadlock recovery, what does resource preemption involve?**
   A. Waiting for user input to release resources
   B. Forcing a process to release its held resources
   C. Avoiding the allocation of requested resources
   D. Killing all blocked processes

10. **Which of the following is true about Resource Allocation Graphs (RAGs)?**
   A. Cycles always indicate a deadlock
   B. A cycle in a RAG is a necessary but not sufficient condition for a deadlock if resources have multiple instances.
   C. They help prevent race conditions
   D. In RAGs, edges are only drawn from resources to processes

ANSWER KEY									
1.	(B)	2.	(B)	3.	(C)	4.	(A)	5.	(B)
6.	(A)	7.	(C)	8.	(C)	9.	(B)	10.	(B)

Inter-process Communication

1. **When is Inter-process communication (IPC) used in OS?**

   When **two or more processes** running on the **same or different computers need** to **exchange data**, they use **IPC**.

2. **In which tasks inter-process communication is essential? (Not very important)**

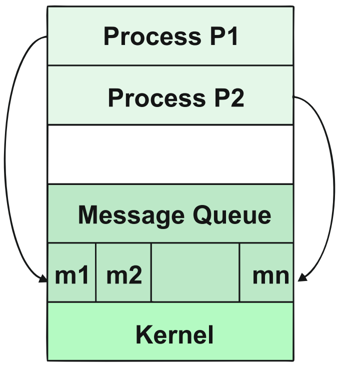
   IPC is essential in many real-world computing tasks where **multiple processes need** to **coordinate**, **share data**, or **work in parallel** to achieve a common goal.
   Here are some key tasks where IPC is essential:
   - **Client-server communication:** Clients request services, and servers respond.
     - IPC is needed for: Transmitting data between client and server (requests, responses).
   - **Component-Based Applications:** A large application is split into smaller, independent modules or services.
     - IPC is needed for: Communication between components.
   - **Parallel Processing:** Divide a large task into smaller parts to be processed in parallel.
     - IPC is needed for: Sharing intermediate results and coordinating processes.
   - **Data Synchronization:** Processes need to work on shared data without conflict.
     - IPC is needed for: Syncing access to shared resources.
   - **Transaction Systems:** Multiple steps of a transaction are managed by different services.
     - IPC is needed for: Coordinating and confirming steps.

3. **Explain different approaches used for Inter-Process Communication.**

   - **Message Passing*:** Processes **send** and **receive** structured messages to communicate. **No memory** is shared between them. Example: Two people writing letters and sending them through a mailbox.

     Characteristics:
     - Simple to implement.
     - Works across machines (network).
     - Safer (no shared memory, so no accidental overwrites).

     Use cases: Client-server models, network communication, distributed systems.

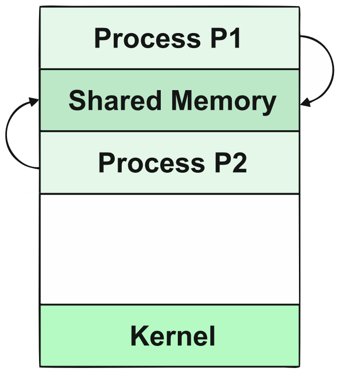


Message Passing Model

   - **Shared Memory*:** Processes **share a region of memory** where they can read and write data. Example: Two people writing on the same whiteboard.

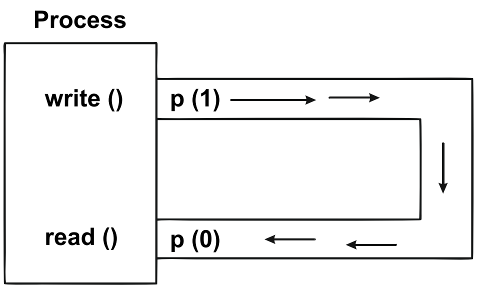
     Characteristics:
     - **Very fast** (no message overhead).
     - Needs synchronization (to avoid data clashes).

     Use cases: High-performance systems like real-time video/audio streaming.



Message Passing Model

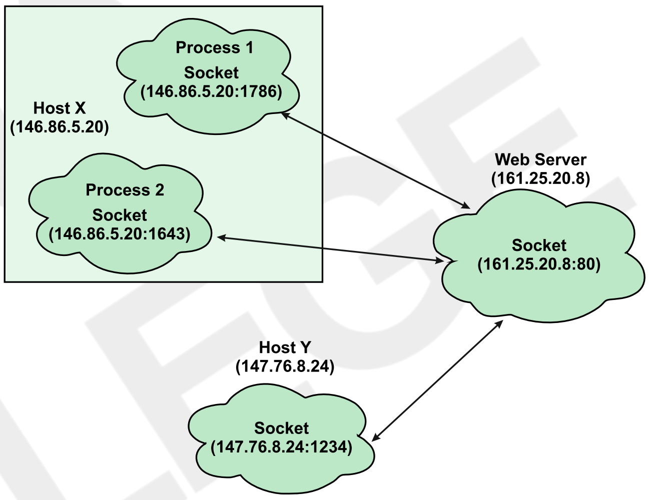
- **Pipes: One-way** communication, usually between **related processes**. Example: parent-child process.
  - **FIFOs (Named Pipes):** One-way but can be used between **unrelated processes.** It is identified by a name (like a named mailbox).



Characteristics:
- Simple.
- Unidirectional (need two pipes for two-way).
- Intermediate speed.

Use Case: Shell command chaining: ls | grep txt.

- **Sockets:** Allows communication between processes on **different machines or the same machine** over a network.
  Example: Phone call, both people talk and listen.



Types:
- **TCP Sockets**: Reliable connection.
- **UDP Sockets**: Faster, but no delivery guarantee.

Characteristics:
- Full-duplex (two-way).
- Works over LAN, Internet.

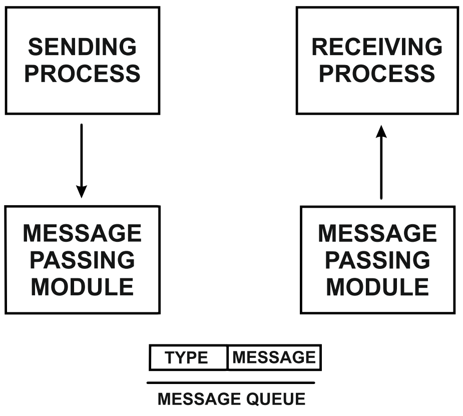
Use cases: Web servers, chat apps, multiplayer games.

- **Signals:** A signal is a **notification sent by the OS** to a process to inform it that an event occurred. Example: Tapping someone on the shoulder to get their attention.

Characteristics:
- Asynchronous (can happen anytime).
- Limited information sent.
- Can be handled or ignored (except SIGKILL).

Use cases: Graceful shutdowns, event notification, alarm handling.

- **Message Queues:** A **queue (buffer)** where processes can store and retrieve messages. Example: A ticket counter queue, requests are processed in order.



Characteristics:
- FIFO order.
- Can send priority messages.
- Persistent across process restarts (System V).

Use cases: Task queues, email systems, print spoolers.

MCQs

1. **Which of the following is true about message passing in IPC?**
   A. It allows processes to share the same memory space
   B. It requires processes to be on the same machine
   C. Communication occurs through sending and receiving messages
   D. It is faster than shared memory communication in all cases

2. **What is a key advantage of shared memory over message passing?**
   A. It offers better synchronization control
   B. It is more secure
   C. It avoids the need for synchronization
   D. It allows direct access to common data, leading to faster communication

3. **Which IPC mechanism is unidirectional and used for parent-child communication?**
   A. Socket
   B. FIFO
   C. Pipe
   D. Signal

4. **What distinguishes a FIFO (named pipe) from a regular pipe?**
   A. FIFO can only be used within the same process
   B. FIFO allows communication between unrelated processes
   C. FIFO is bidirectional
   D. FIFO doesn't require any file system support

5. **Signals are typically used for:**
   A. High-volume data transfer between processes
   B. Low-level process control and simple notifications
   C. Creating shared memory
   D. Managing sockets and FIFOs

ANSWER KEY									
1.	**(C)**	**2.**	**(D)**	**3.**	**(C)**	**4.**	**(B)**	**5.**	**(B)**

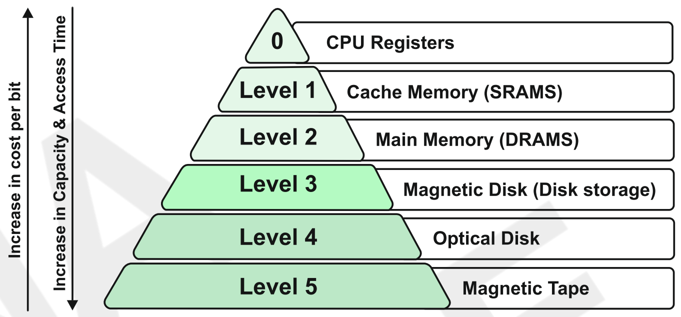
Memory Management

1. **What is Memory Hierarchy?**

Our systems consist of **various types of memory devices** like register, cache memory, main memory etc., each of these components has **different performance rates** and **specific usages.**

Memory Hierarchy is an **arrangement** and **visualization** of these various memory devices considering their **performance, access time, and cost per bit,** which proves to be helpful while designing a new system to balance its overall performance-to-cost ratio.

Memory Hierarchy Design



2. **What is memory management?**

Memory management is how the OS **organizes** and **controls system memory**.
Example: It is like a librarian who manages everything in a library so that people can work peacefully.

3. **Use of Memory Management in OS**

- It **allocates memory** to processes and retakes it when task is completed.
- It **tracks** space in memory (used/free).
- It ensures **isolation** (no process accesses others' memory).
- It supports **multitasking** and **virtual memory**.

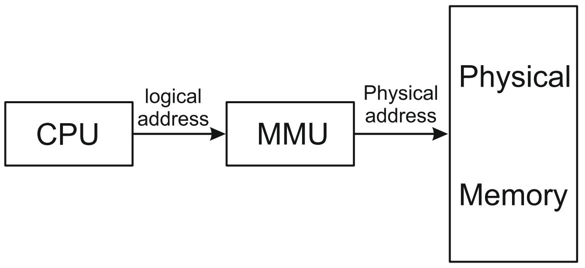
4. **What is a Memory Management Unit?**

The **Memory Management Unit (MMU)** is a **hardware component** responsible for **handling memory-related operations** in a computer system.
It **translates logical (virtual) addresses** generated by the CPU into **physical addresses** used in RAM.

Functions of MMU:

- **Address Translation:** It converts **logical address → physical address.**
- **Memory Protection:** It ensures isolation.
- **Access control:** It controls read/write/execute permissions for memory blocks.
- **Segmentation and Paging support:** It supports **segmented** or **paged memory systems.**

☆ 5. Difference between Logical and Physical Address?



Feature	Logical Address	Physical Address
Definition	Address generated by the **CPU**	Actual address in the **main memory (RAM)**
Visible To	**User/program**	**Hardware only**, not visible to the user
Generated By	**CPU** during program execution	**Memory Management Unit (MMU)**
Used In	**Logical address space**	**Physical address space**
Mapping	Requires **translation** using page table/MMU	Directly accessed by **hardware**
Example	0x000004	0x1F0004 (after translation)

6. What is swapping?

Swapping moves **inactive processes to disk** (swap space) to free up RAM and brings back when needed.

- It increases **multitasking**.
- Although it is slower due to disk access.

☆ 7. How does the OS manage isolation and protection?

- Each process gets its own **virtual(logical) address space**.
- MMU translates **virtual addresses to physical addresses** and enforces memory boundaries.
- Pages marked as **read-only, write, or execute-only**, prevents illegal memory operations.
- **User Mode** and **Kernel Mode** ensures user processes can't harm the system.
- Define **allowed memory** range per process if access outside range causes an OS trap.
- Each process runs in **its own protected space** and **no direct access** to another process's data.
- Files and devices have **permissions (read, write, execute)** and OS checks user/process rights before access.

8. What is fragmentation and its types?

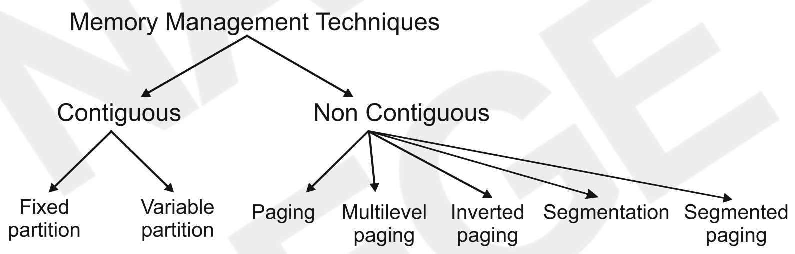
Fragmentation is a condition where **memory is wasted** or **cannot be used efficiently**, even though **enough total memory is free.**
It happens due to **improper memory** allocation and causes performance issues.

Types of fragmentation:

Type	Description	Example	Solution
1. Internal Fragmentation	Wasted space **inside allocated memory blocks**.	A process gets 8 KB memory, but uses only 6 KB. The remaining 2 KB is wasted.	Use **variable partitioning** or paging
2. External Fragmentation	Wasted space **between allocated blocks**, not enough for new processes.	100 MB free, but scattered in small chunks, can't fit a 20 MB process.	Use **compaction** or **paging**

☆ **9. Explain different memory management techniques.**

In contiguous memory allocation, each process is allocated **one single continuous block** of memory. All the bytes of the process are stored **together in adjacent memory locations**.



This was one of the earliest memory management techniques used in operating systems.

Types of Contiguous Memory Allocation

i. **Fixed Partitioning**

- Memory is divided into **fixed-size partitions** at system startup.
- Each partition holds **exactly one process**.
- If a process is smaller than the partition, **unused space is wasted**.

Advantages:

- Simple to implement.
- Supports multiprogramming.

Disadvantages:

- Internal fragmentation occurs if process size < partition size.
- Limits process size to partition size.

ii. **Variable Partitioning**

- Memory is divided **dynamically** based on the **actual size** of incoming processes.
- Partitions are created **at runtime**.

Advantages:

- Better **memory utilization** than fixed partitioning.
- **No internal fragmentation.**

Disadvantages:

- **External fragmentation** occurs.
- Requires memory compaction to combine scattered free spaces.

In **non-contiguous allocation**, a process is **not stored in a single continuous block** of memory. Instead, it is **divided and placed in different locations** in memory.
This helps in **efficient memory usage** and eliminates **external fragmentation**.

Types of Non-Contiguous Memory Allocation

i. Paging

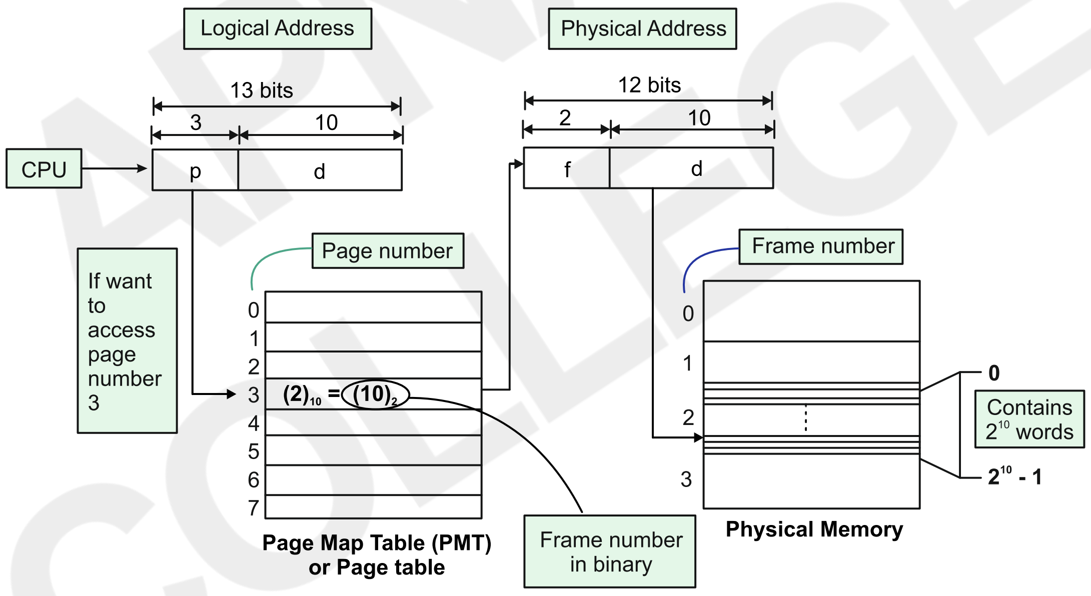
- **Memory** is divided into fixed-size **frames**.
- **Process** is divided into fixed-size **pages** (same size as frames).
- A **page table** maintains the mapping of each page to a frame.

Advantages:

- Eliminates fragmentation.
- Simple implementation.

Disadvantages:

- Requires extra memory for page tables.



ii. Multilevel Paging

- When page tables are large, they are **split into multiple levels**.
- A **first-level table** points to **second-level page tables**, and so on.

Advantages:

- **Reduces** memory needed for storing page tables.
- Supports **large address** spaces efficiently.

Disadvantages:

- **Increases** memory access time due to multiple lookups.
- **More complex** to implement.

iii. Inverted Paging (Inverted Page Table)

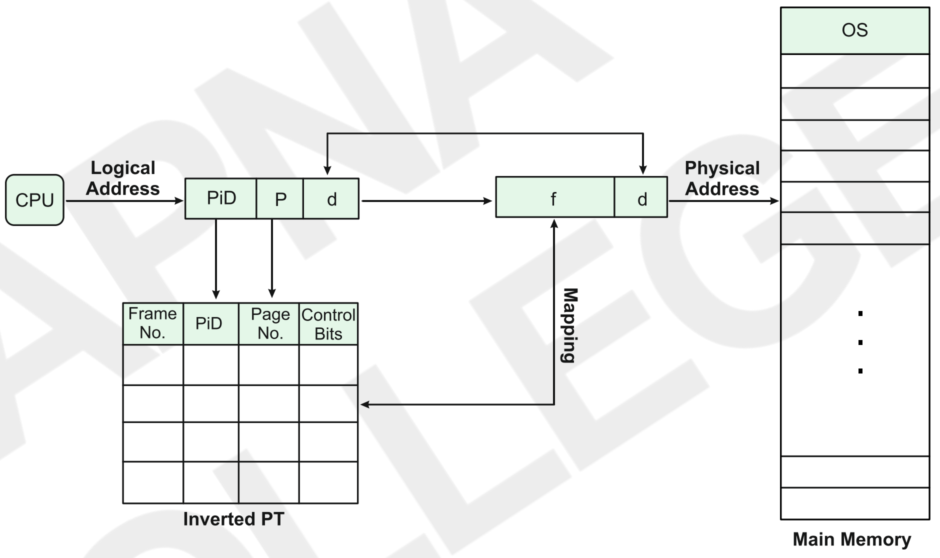
- Instead of one page table per process, there is **one global page table** for the entire system.
- Each entry corresponds to a **physical frame** and stores the **process ID and page number**.

Advantages:

- It saves memory space because only one global page table is used for the entire system.
- **Scales** with physical memory size.

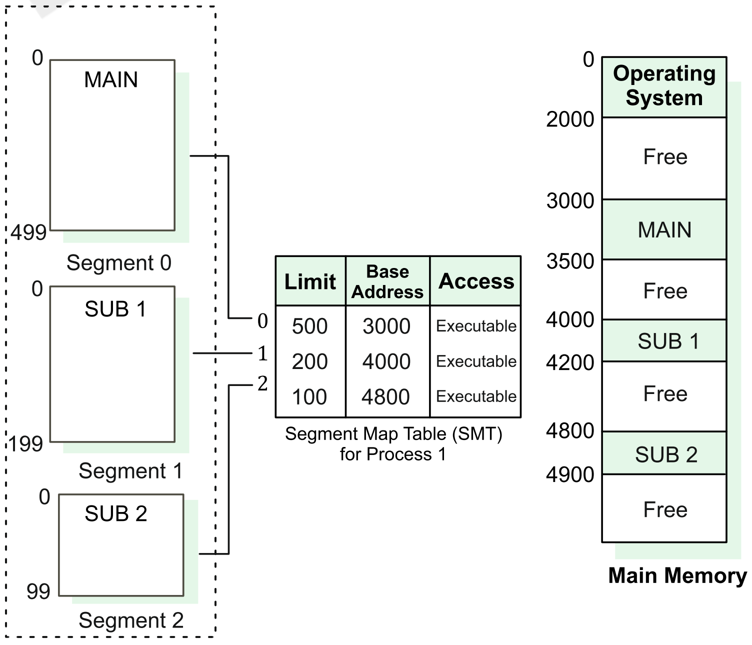
Disadvantages:

- **Slower** address translation (requires searching or hashing).
- **More complex** management.



iv. Segmentation

- A process is divided into **logical segments** like code, stack, heap, data, etc.
- Each segment has a **base address** and **limit**.
- Segment table maps each segment to a physical address.



	Limit	Base Address	Access
0	500	3000	Executable
1	200	4000	Executable
2	100	4800	Executable

Advantages:

- **Logical division** improves modularity.
- Easy to **share** or **protect** segments individually.

Disadvantages:

- **External fragmentation** may occur.
- Needs **complex management.**

v. **Segmented Paging**

- Combines **segmentation and paging**.
- Process is divided into **segments**, and each segment is further divided into **pages**.
- Uses a **segment table** and **page table per segment**.

Advantages:

- **Logical view** from segmentation with efficient memory use from paging.
- **Reduces** both **external** and **internal** fragmentation.

Disadvantages:

- **Complex address translation** (needs both segment and page lookups).
- **Higher** memory management **overhead**.

10. Explain various allocation methods used in Contiguous memory allocation.

There are 4 allocation methods used in Contiguous memory allocation

i. **First fit:** It allocates the **first available free block** that is large enough for the process.
Example:
Free blocks: 10KB, 20KB, 15KB.
Request: 12KB → Allocated to 20KB block (first that fits).

ii. **Next fit:** Similar to first fit, but it **starts searching from where it last left off**, not from the beginning.

iii. **Best Fit:** It allocates the **smallest free block** that is **big enough** for the process.

iv. **Worst Fit:** It allocates the **largest free block**, assuming large remaining space will be useful later.

☆ **11. What is Virtual Memory?**

Virtual Memory is a memory management technique that makes the computer think it has **more RAM than it actually does** by using a part of the **hard disk** as extra memory.

Key points:

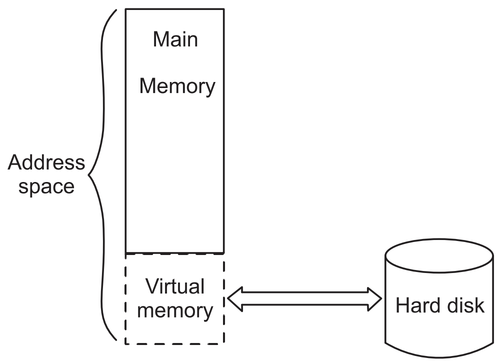
- Allows you to **run large programs** even with **limited physical RAM**.
- OS moves data **between RAM and disk** as needed.
- Helps in **efficient memory usage and process isolation**.

Advantages:

- Enables **larger programs** to run.
- **Isolation** between processes.
- **Efficient memory utilization.**

Disadvantages:

- Accessing data from disk is **slower** than RAM.
- Can lead to **thrashing** if overused.



Logical view of virtual memory

12. What is demand paging?

In demand paging, **pages are loaded into memory only when they are needed**, not in advance.

How it works:

- Process starts with **no pages** in memory.
- When a page is needed → **Page Fault occurs**.
- OS loads the page from disk to RAM.

Advantages:

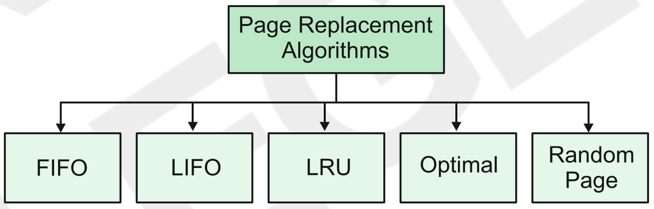
- Saves memory.
- Faster program startup.

Disadvantages:

- Causes initial **page faults**.
- Page faults are **costly** if frequent.

13. Explain different page replacement algorithms.

When memory is full and a new page is needed, **one page must be replaced**. These algorithms decide **which page to remove**.



i. **FIFO (First-In First-Out):** Remove the **oldest page** (the one loaded first).
Example: Page Reference: 1 2 3 (3 frames), Next frame is 4 → Page fault for 4 → remove 1 (first in).

Characteristics:

- Simple to implement (use a queue).
- May remove frequently used pages.
- **Belady's Anomaly** (more frames, more page faults) can be caused by the FIFO page replacement algorithm. It does not affect LRU or Optimal algorithms.

ii. **LIFO (Last-In First-Out):** Remove the **most recently loaded page**.
Example: Page Reference: 1 2 3 → Next frame is 4 → remove 3 (last loaded).

Characteristics

- Simple stack implementation.
- Not practical or efficient in real-world use.
- May remove the most needed page.

iii. **LRU (Least Recently Used):** Remove the page that **hasn't been used for the longest time.**
Example: Reference: 1 → 2 → 1 → 3. Since 1 was recently used, 2 is least recently used and is removed.

Characteristics:

- Good approximation of real-world behavior.
- Needs to track usage history (can be complex).

iv. **Optimal:** Remove the page that **won't be used for the longest time in future**.
Example: Reference: 1 2 3 4 1 2, on needing 5, if 3 will not be used for the longest time in the future, then it is selected for replacement.

Characteristics:

- **Best possible performance.**
- Not implementable (requires **future knowledge**).
- Used for **benchmarking.**

v. **Random Page Replacement:** Remove a **randomly selected page** from memory.

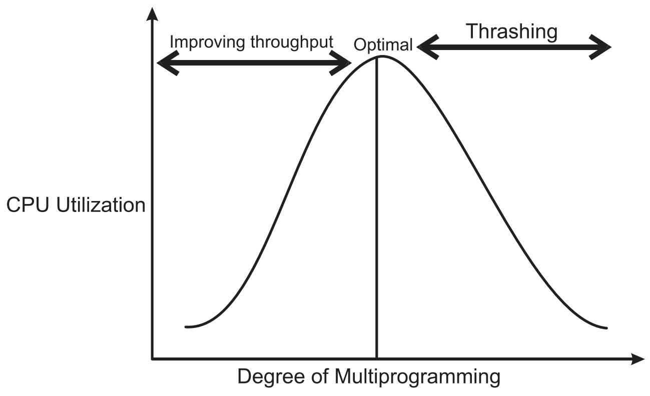
Characteristics:

- Easy and fast to implement.
- Avoids pattern-based failures.
- Can evict frequently used pages.
- Performance is unpredictable.

Algorithm	Removes which Page	Complexity	Practical Use	Accuracy
FIFO	Oldest loaded	Easy	Moderate	Low
LIFO	Most recently loaded	Easy	Rare	Poor
LRU	Least recently used	Moderate	Common	Good
Optimal	Not used for longest in future	Hard	No (theoretical)	Best
Random	Random page	Easy	Rare	Varies

14. What is Thrashing?

Thrashing occurs when the system **spends more time swapping pages** in and out of memory than executing actual processes.



Causes:

- Too many processes with not enough RAM.
- **Frequent page faults**.
- **Insufficient working set** (set of pages a process needs regularly).

Effects:

- Drastic **drop in performance**.
- High **CPU utilization** but low actual work.

Solutions:

- Reduce the **degree of multiprogramming**.
- Use a working **set model** to allocate enough pages.
- Apply better **page replacement algorithms**.

Extra topics

15. What is a Translation Lookaside Buffer (TLB)?

The **TLB** is a **small, fast cache** used by the **MMU** to **store recent translations** of **logical addresses to physical addresses**.

How it works:

- The CPU generates a **logical address**.
- MMU checks the TLB for the page number:
  - **TLB Hit** → Gets physical address quickly.
  - **TLB Miss** → Looks up the page table in memory, updates TLB.
- Translated **physical address** is sent to RAM.

16. Why is TLB needed?

- In paging systems, **each memory access** requires a **page table lookup**.
- Accessing the page table in memory is **slow**.
- TLB stores recent address mappings to **speed up address translation**.

17. What is Overlay in OS?

Overlay was historically used to run programs larger than the available memory, before virtual memory became common.
It allows a program to be **divided into mutually exclusive modules**, where **only one module (or overlay) is loaded into memory at a time**.

Key Concept:

- Instead of loading the **entire program** into memory,
- Only the **needed part (overlay)** is loaded and executed.
- When another part is needed, it **replaces the current one** in memory.

Disadvantages:

- Programmer must **manually design overlays.**
- Can be **complex to manage.**
- Rarely used today due to **virtual memory.**

18. What is Caching?

Caching is the process of **storing frequently accessed data** in a **faster memory** (cache) so that **future requests** for that data can be served **faster**.

Example:

- **CPU cache** stores frequently used instructions or data from RAM.
- **Web browser cache** stores visited web pages locally to load them faster next time.

19. What is Compaction in OS?

Compaction is the process of **shifting all the allocated memory blocks together** to one end of memory and **combining all the free spaces** into one large block to reduce **external fragmentation** in **contiguous memory allocation** systems.

Key Points:

- Done by the **Operating System**.
- Compaction can be time-consuming because it involves moving processes and updating memory references.
- Usually done **in systems that support relocation**.

Characteristics:

- **Eliminates external fragmentation.**
- Allows loading of **larger processes**.
- **Overhead of data movement** (CPU time, performance hit).
- Not suitable for **real-time systems**.

MCQs

1. **What is the main purpose of memory management in an operating system?**
   A. To compress memory usage
   B. To divide the CPU for multitasking
   C. To allocate and protect memory for processes
   D. To monitor hardware devices

2. **The Memory Management Unit (MMU) is responsible for:**
   A. Handling cache replacement
   B. Translating logical addresses to physical addresses
   C. Managing disk fragmentation
   D. Swapping files

3. **Which address is generated by the CPU during a program's execution?**
   A. Physical address
   B. Logical address
   C. Virtual address
   D. Swap address

4. **Swapping is a technique where:**
   A. Pages are locked in memory permanently
   B. The CPU is exchanged between processes
   C. A process is temporarily moved out of memory
   D. Memory is copied to the hard drive

5. **In internal fragmentation, memory is wasted:**
   A. Between allocated and free blocks
   B. Outside memory modules
   C. Within allocated memory blocks
   D. During disk I/O

6. **Which allocation strategy scans from the location of the last allocation?**
   A. First Fit
   B. Best Fit
   C. Worst Fit
   D. Next Fit

7. **In paging, a logical address is divided into:**
   A. Page number and segment number
   B. Frame number and offset
   C. Page number and page offset
   D. Segment and offset

8 **The page table stores:**
   A. Physical memory addresses of processes
   B. Mapping of logical pages to physical frames
   C. Process execution time
   D. File descriptors

9. **What is the main function of the Translation Lookaside Buffer (TLB)?**
   A. Handling disk swapping
   B. Reducing context switch time
   C. Caching recent page table entries
   D. Storing memory allocation algorithms

10. **Which paging scheme uses a page table where each entry points to another page table?**
    A. Segmentation
    B. Inverted Paging
    C. Multilevel Paging
    D. Direct Mapping

11. **In segmentation, a logical address consists of:**
    A. Segment number and offset
    B. Page number and frame
    C. Block and byte
    D. Page and segment

12. **Which of the following is true about segmented paging?**
    A. Pages are divided into segments
    B. Segments are divided into pages
    C. It uses only virtual memory
    D. No page table is used

13. **Which memory allocation method often causes the least internal fragmentation?**
    A. Fixed partitioning
    B. Best Fit
    C. Worst Fit
    D. First Fit

14. **What does virtual memory enable a process to do?**
    A. Run without CPU access
    B. Access disk files directly
    C. Use more memory than physically available
    D. Disable swapping

15. **In demand paging, pages are:**
    A. Pre-loaded into memory
    B. Loaded only when needed
    C. Locked into cache
    D. Copied to a new address

16. **Which page replacement algorithm uses a queue and replaces the oldest page first?**
    A. LRU
    B. Optimal
    C. FIFO
    D. Random

17. **Which page replacement algorithm uses future knowledge to minimize faults?**
    A. FIFO
    B. Optimal
    C. LRU
    D. Random

18. **Thrashing occurs when:**
    A. CPU overheats
    B. Cache is full
    C. The system spends most time swapping pages
    D. The TLB is invalid

19. **What is the role of caching in memory management?**
    A. It stores future predictions
    B. It replaces the page table
    C. It provides fast access to frequently used data
    D. It handles segmentation

20. **What is the purpose of overlay in memory management?**
    A. To implement page tables
    B. To allow execution of large programs in small memory
    C. To avoid fragmentation
    D. To enhance CPU speed

ANSWER KEY									
1.	(C)	2.	(B)	3.	(B)	4.	(C)	5.	(C)
6.	(D)	7.	(C)	8.	(B)	9.	(C)	10.	(C)
11.	(A)	12.	(B)	13.	(B)	14.	(C)	15.	(B)
16.	(C)	17.	(B)	18.	(C)	19.	(C)	20.	(B)

File Systems

1. **What is a File System in OS?**

   A **File System** organizes and manages how data is **stored** and **retrieved** on storage devices.
   - **Main functions:** File **creation/deletion, directory management**, **storage allocation**, **access control**.
   - **Common examples:** NTFS (Windows), ext4 (Linux), FAT32, HFS+.

2. **Advantages of using a File System in OS.**
   - **Data Organization**: Structures data into files and directories.
   - **Access Control**: Enforces **permissions** and **user restrictions**.
   - **Data Sharing**: Allows **sharing** of files among **users** and **applications**.
   - **Data Recovery**: Supports **backup** and **restore operations**.
   - **Efficient Storage**: **Allocates** and **reclaims** space efficiently**.**

3. **Disadvantages of using a File System in OS**
   - **Overhead**: Requires **additional** space for **metadata**.
   - **Fragmentation**: Can cause **slow access** due to **scattered** storage.
   - **Complexity**: Advanced file systems are **harder** to manage.
   - **Performance Bottlenecks**: **Poor allocation** or **indexing** may **slow** operations.

4. **Properties of a File System.**
   - **Persistence**: Data is **stored permanently**.
   - **Security**: Provides **controlled access**.
   - **Scalability**: Handles **large volumes** of data.
   - **Efficiency**: Optimizes performance for **storage** and **retrieval**.
   - **Reliability**: Prevents data **corruption** and enables **recovery**.

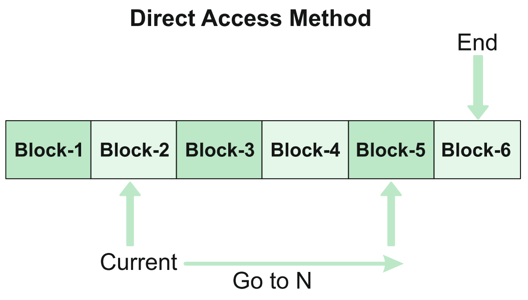
5. **What are File Attributes in OS?**

   File attributes are metadata that describes the file. Common attributes include:
   - **Name**: Human-readable identifier.
   - **Type**: Format or usage of file (e.g., .txt, .exe).
   - **Size**: File size in bytes.
   - **Location**: Physical location on the disk.
   - **Timestamps**: Created, modified, and accessed times.
   - **Permissions**: Read, write, execute rights.
   - **Owner**: User who owns the file.

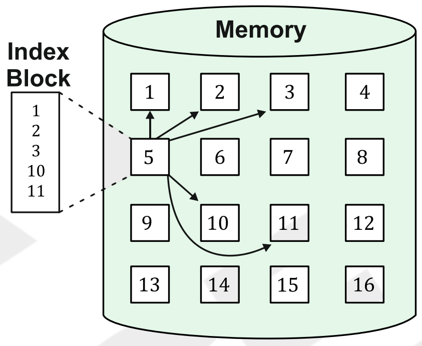
6. **What are commonly used terms in File Systems?**
   - **File Descriptor**: Identifier used by the OS to access files.
   - **Metadata**: Data about data (e.g., size, timestamps).
   - **Mounting**: Making a file system accessible at a directory.
   - **Pathname**: Location of a file in the directory structure.
   - **Inode (in UNIX)**: Stores file attributes and disk block pointers.

7. Explain different File Access Methods.

i. Sequential Access
- Data accessed in **order** (e.g., audio files).
- **Simple**, **fast** for **large reads**.
- **Slow** for **random** access.



ii. Direct Access
- **Jump** to **any block** directly (e.g., database).
- **Fast**, **flexible**.
- More **complex** than **sequential**.

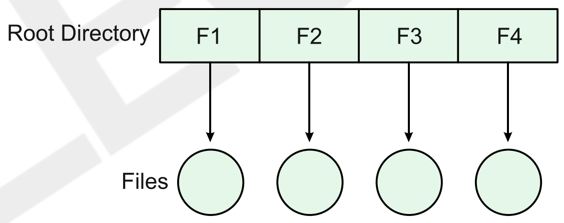


Index Access of File

iii. Indexed Access
- Uses an **index block or table** to locate data blocks of a file.
- **Efficient** for **random access**.
- **Overhead** of maintaining indexes.

8. Directory Structures

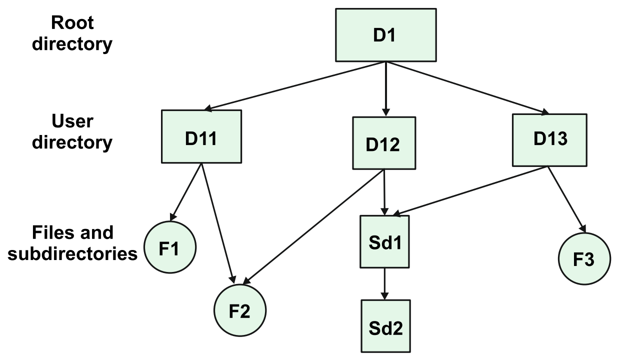
Single-level Directory Structure



i. Single-Level Directory
- **One** directory for **all files**.
- Simple.
- Name **conflicts**, not **scalable**.

ii. Two-Level Directory
- **Separate directory** for each user.
- **No** name **conflict** between users.
- **No sharing** between users.

iii. Tree Structure
- **Hierarchical**, allows subdirectories.
- **Organized**, scalable.
- **Complex** traversal.



iv. Acyclic Graph (DAG)
- **Allows shared** files/directories (links).
- Efficient **sharing**.
- Deletion is complex because multiple directory entries may reference the same file.

9. File Types in an OS

i. **Text Files**: Human-readable (.txt, .csv).
ii. **Binary Files**: Machine-readable (.exe, .jpg).
iii. **Directories**: Containers for files.
iv. **Special Files**: Devices or system files (e.g., /dev/null in Linux).

☆ 10. Explain different file allocation methods in the OS.

There are **three main file allocation methods**:

i. **Contiguous Allocation:** Each file occupies a set of **contiguous (adjacent)** blocks on the disk.
Example: If a file needs 5 blocks, it may be stored in blocks **10–14**.

Characteristics:

- **Fast access**: both **sequential** and **direct** access are fast.
- **Simple** to implement.
- **External fragmentation**: free space is scattered, making allocation difficult.
- **Difficult to grow** files: may require moving the entire file if more space is needed.

ii. **Linked Allocation:** Each file is a **linked list** of blocks; each block contains a **pointer** to the next block.
Example: File → Block **7** → Block **12** → Block **3** → null.

Characteristics:

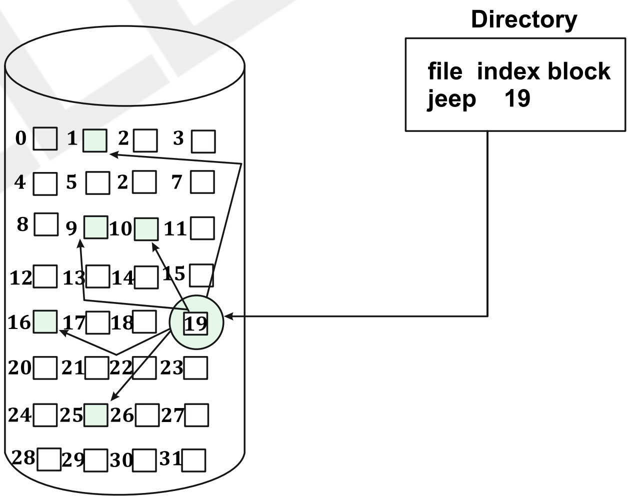
- **No external fragmentation**.
- **Easy to grow** files.
- **Slow direct access**: must traverse from the beginning to reach a specific block.
- **Space overhead**: each block needs a **pointer**.
- **Reliability issue**: if one pointer is **corrupted**, the file may become inaccessible.

iii. **Indexed Allocation:** Each file has a separate **index block** that contains all the addresses of its data blocks.

Example: Index block → [**5, 9, 13, 18**]

Characteristics:

- **Random (direct) access** supported.
- **No external fragmentation**.
- **Easy to grow** files dynamically.
- **Additional space** required for index blocks.
- For large files, it may need **multi-level indexing**, adding complexity.



Feature	Contiguous Allocation	Linked Allocation	Indexed Allocation
Access Type	Fast (both)	Slow (direct)	Fast (direct)
Fragmentation	External	None	None
File Growth	Difficult	Easy	Easy
Space Overhead	None	Pointer per block	Index block needed
Complexity	Simple	Moderate	Moderate/High

11. How is Free Space Management done in OS?

- **Bitmaps**:
  - A bitmap (or bit vector) is a series of bits where each bit corresponds to a disk block.
  - Typically, a '1' indicates that the block is allocated, and a '0' indicates it is free.
- **Free List**:
  - Linked list of free blocks.
  - Easy allocation.
- **Grouping**:
  - Store addresses of free blocks in groups.
- **Counting**:
  - Store starting address + count of free blocks.

12. File System Mounting. (Less Imp)

It is a process of making a file system accessible to the OS.

- **Mount Point**: Directory where new FS is attached.
- **Steps**:
  1. User requests mount.
  2. OS verifies the file system type.
  3. Updates mount table.

Example: Mounting USB drive under /media/usb.

Extra topics

13. Unix File System (UFS)

- A foundational file system for many UNIX-based operating systems. Its principles have influenced modern file systems, including those used in Linux.
- **Structure**:
  1. **Boot Block** – Contains bootloader.
  2. **Superblock** – Contains metadata about the file system.
  3. **Inode Table** – Stores metadata of individual files (permissions, size, etc).
  4. **Data Blocks** – Actual file contents.

Features:

- Hierarchical (tree-structured) directories.
- Uses **inodes** to manage file metadata.
- Supports access control using permissions.

MCQs

1. **What is the primary purpose of a file system in an operating system?**
   A. To compress data for storage
   B. To allocate CPU time
   C. To manage how data is stored and retrieved
   D. To enhance security

2. **Which of the following is a major advantage of using a file system?**
   A. Unlimited memory allocation
   B. Direct hardware access
   C. Organized data storage and access
   D. Reduced execution time

3. **A key disadvantage of file systems is:**
   A. Easy multi-user access
   B. File corruption risks
   C. Fixed file size
   D. Automatic sorting of files

4. **The access time, modification time, and permissions of a file are all examples of:**
   A. File structure
   B. File attributes
   C. File types
   D. File extensions

5. **In sequential access file method:**
   A. Records can be accessed in any order
   B. Each record has a unique index
   C. Records are accessed one by one, in order
   D. Records can be accessed using offset.

6. **Which access method allows direct jump to any block of a file?**
   A. Indexed
   B. Sequential
   C. Tree-based
   D. Paged

7. **Indexed access is most efficient when:**
   A. Accessing records randomly
   B. Appending data continuously
   C. Using fixed-length records
   D. Accessing large directories

8. **In the single-level directory structure:**
   A. Each user has their own directory
   B. Directories form a tree
   C. All files are placed in one directory
   D. Files can be nested

9. **Which directory structure allows efficient searching and shared subdirectories?**
   A. Single-level
   B. Two-level
   C. Tree
   D. DAG (Directed Acyclic Graph)

10. **A special file in a file system refers to:**
    A. A password-protected file
    B. A file used for backups
    C. Device files that represent hardware
    D. Encrypted text files

11. **In contiguous allocation, a file:**
    A. Is stored in separate disk sectors
    B. Can be stored in any available block
    C. Occupies a sequence of adjacent blocks
    D. Is allocated using a hash table

12. **What is a major problem with linked allocation?**
    A. It wastes memory
    B. It requires large tables
    C. Random access is inefficient/slow
    D. Files cannot be modified

13. **In indexed allocation, each file has:**
    A. A pointer to the first block
    B. A linked list of blocks
    C. An index block containing pointers to all data blocks
    D. A tree of block numbers

14. **Which of the following is not a free space management technique?**
A. Bit vector
B. Grouping
C. Paging
D. Counting

15. **In Unix file systems, the mount operation is used to:**
A. Compress files
B. Rename a file
C. Make a file system accessible under a directory
D. Format the disk

ANSWER KEY									
1.	**(C)**	**2.**	**(C)**	**3.**	**(B)**	**4.**	**(B)**	**5.**	**(C)**
6.	**(A)**	**7.**	**(A)**	**8.**	**(C)**	**9.**	**(D)**	**10.**	**(C)**
11.	**(C)**	**12.**	**(C)**	**13.**	**(C)**	**14.**	**(C)**	**15.**	**(C)**

Disk Management

1. What is Disk Management in OS?

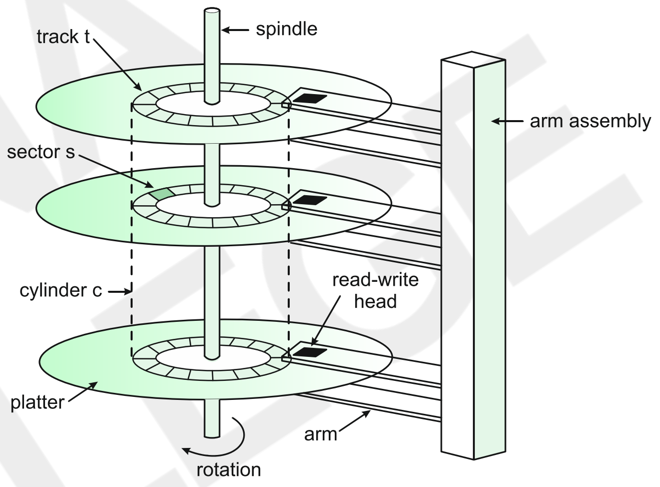
Disk Management is a core function of the OS responsible for **controlling** the use of disk storage.

- Manages space allocation, formatting, partitioning, and file systems.
- Ensures efficient read/write operations and prevents data loss or corruption.

2. Structure of a Disk.

i. Platters

- Circular disks made of metal or glass.
- Coated with magnetic material for storing data.
- Each platter has two **magnetic surfaces** (top and bottom) for storing data.



ii. Spindle

- Central axis that holds and spins all the platters.
- All platters spin **together at a constant speed** (e.g., 5400 or 7200 RPM).

iii. Read/Write Heads

- Tiny magnetic heads that **read** or **write** data on platter surfaces.
- Each platter surface has its **own head**.
- Heads are attached to an **actuator arm**.

iv. Actuator Arm

- Moves the read/write heads across the disk surface.
- All heads move **in unison** (they are mechanically linked).

v. Tracks

- Concentric circles on each platter surface.
- Data is stored along these circles.

vi. Sectors

- Each track is divided into smaller sections called **sectors**.
- Smallest unit of storage (usually **512 bytes** or **4 KB**).

vii. Cylinders

- A **set of tracks vertically aligned** across all platters (same radius).
- Accessing data on the same cylinder is faster (no head movement needed).

viii. Disk Blocks

- Group of sectors managed by the file system.
- Used for reading/writing in logical units.

ix. **Cache (Buffer)**
  - Small high-speed memory in the disk controller.
  - Temporarily holds frequently accessed data to improve performance.

3. What Does Disk Management Do?

- **Partitioning**: Divides a physical disk into logical sections.
- **Formatting**: Prepares the partition to store files using a file system.
- **File System Management**: Tracks files and directories, their sizes, and locations.
- **Space Allocation**: Allocates/free blocks for files and processes.
- **Disk Scheduling**: Optimizes the order of read/write operations.
- **Defragmentation**: Rearranges data for faster access (in some OSes).
- **Error Checking**: Identifies bad sectors and recovers from disk failures.

4. Key terms in Disk Scheduling Algorithms.

i. **Seek Time:** Time taken by the disk arm to move the read/write head to the track where the data is located.
  - **Importance**: Lower seek time = faster data access.
  - **Optimization Goal**: All scheduling algorithms aim to **minimize seek time**.

ii. **Rotational Latency:** Time taken for the desired sector of the disk to rotate under the read/write head.
  - **Depends on**: Disk's rotational speed (RPM).
  - **Usually smaller than seek time**, but still contributes to total access time.

iii. **Transfer Time:** Time taken to transfer data to/from the disk after the head is positioned.

iv. **Access Time:** Total time taken to read/write data from the disk.
Access Time = Seek Time + Rotational Latency + Transfer Time

v. **Request Queue:** List of pending I/O requests for specific disk tracks.
  - Scheduling algorithms decide the **order** in which to serve these.

vi. **Starvation:** A situation where a request is never served because newer, closer requests keep getting priority.
  - **Common in**: SSTF algorithm.

vii. **Fairness:** Ensuring all requests get serviced in a reasonable time.
  - **Important for user experience** and system stability.

viii. **Direction Bit / Movement Direction:** Indicates whether the disk head is moving inward (toward lower tracks) or outward.
  - **Used in**: SCAN, LOOK, C-SCAN, C-LOOK.

ix. **Head Start Position:** The current position of the disk head before starting to serve requests.
  - **Effects seek time** and the behavior of all algorithms.

☆ **5. Explain different Disk scheduling algorithms.**

i. **FCFS (First Come First Serve):** Requests are processed in the order they arrive.
Example:
If the head is at 50, and requests come for: 82, 170, 43, 140, 24, 16, 190
Seek sequence: 50 → 82 → 170 → 43 → 140 → 24 → 16 → 190

Characteristics:
- Simple and fair.
- High total seek time.
- Does not consider request proximity.

ii. **SSTF (Shortest Seek Time First):** Selects the request closest to current head position.
Example:
Head at 50, requests: 82, 170, 43, 140, 24, 16, 190
Seek sequence: 50 → 43 → 24 → 16 → 82 → 140 → 170 → 190

Characteristics:
- Lower average seek time than FCFS.
- **Starvation** possible for far requests.

iii. **SCAN (Elevator Algorithm):** Head moves in one direction servicing all requests, then reverses.
Example:
Head at 50, direction: right
Requests: 82, 170, 43, 140, 24, 16, 190
Seek sequence: 50 → 82 → 140 → 170 → 190 → reverse → 43 → 24 → 16
(Moves toward higher track numbers first, then reverses to service lower ones).

Characteristics:
- More uniform wait times than SSTF.
- Edge requests may wait longer.

iv. **C-SCAN (Circular SCAN):** Moves in one direction only, then jumps to start and continues.
Example:
Head at 50, direction: right
Requests: 82, 170, 43, 140, 24, 16, 190
Seek sequence: 50 → 82 → 140 → 170 → 190 → jump to 0 → 16 → 24 → 43
(After reaching end, head jumps to track 0 without servicing).

Characteristics:
- Uniform wait time across requests.
- Longer travel distance.

v. **LOOK:** Similar to SCAN but head only goes as far as the last request in each direction (no full sweep).
Example:
Head at 50, direction: right
Requests: 82, 170, 43, 140, 24, 16, 190
Seek sequence: 50 → 82 → 140 → 170 → 190 → reverse → 43 → 24 → 16

Characteristics:

- Avoids unnecessary movement beyond last request.
- Requests in one direction may be serviced sooner, leading to directional bias.

vi. **C-LOOK:** Like C-SCAN but only goes to the last request, then jumps back to the lowest request.
Example:
Head at 50, direction: right
Requests: 82, 170, 43, 140, 24, 16, 190
Seek sequence: 50 → 82 → 140 → 170 → 190 → jump to 16 → 24 → 43

Characteristics:

- More efficient than C-SCAN.
- Slightly more complex to implement.

MCQs

1. **What is the primary purpose of disk management in an operating system?**
   A. To manage RAM allocation
   B. To control access to CPU registers
   C. To manage storage devices and data organization
   D. To schedule processes

2. **Which of the following is not a component of disk structure?**
   A. Platter
   B. Track
   C. Sector
   D. Register

3. **What does disk management do in an OS?**
   A. Controls CPU frequency
   B. Handles memory paging
   C. Manages disk formatting, partitioning, and file system handling
   D. Schedules I/O interrupts

4. **In the context of disk scheduling, seek time refers to:**
   A. Time to transfer data from disk to memory
   B. Time to locate the correct sector on a track
   C. Time to rotate the disk under the read/write head
   D. Time to move the read/write head to the correct track

5. **Which of the following is a disk scheduling algorithm that processes requests in the order they arrive?**
   A. SSTF
   B. SCAN
   C. FCFS
   D. C-LOOK

6. **Which algorithm selects the disk I/O request closest to the current head position?**
   A. FCFS
   B. SSTF
   C. C-SCAN
   D. LOOK

7. **Which disk scheduling algorithm moves the head in one direction, servicing requests, and then reverses direction?**
   A. LOOK
   B. SSTF
   C. SCAN
   D. C-LOOK

8. **The C-SCAN algorithm differs from SCAN in that:**
   A. It services requests only in one direction and jumps to the start
   B. It ignores all incoming I/O requests
   C. It sorts requests in ascending order
   D. It provides better seek time than SSTF in all cases

9. **In the LOOK algorithm, the disk arm:**
   A. Goes to the end of the disk before reversing
   B. Stops where there are no further requests in the current direction
   C. Ignores the last track
   D. Moves continuously without reversing

10. **Which of the following algorithms is designed to reduce starvation in disk scheduling?**
   A. FCFS
   B. SSTF
   C. SCAN
   D. Random

ANSWER KEY									
1.	**(C)**	**2.**	**(D)**	**3.**	**(C)**	**4.**	**(D)**	**5.**	**(C)**
6.	**(B)**	**7.**	**(C)**	**8.**	**(A)**	**9.**	**(B)**	**10.**	**(C)**