CBCS SCHEME

USN | 1 | C | R | 2 | 5 | M | C | 0 | 5 | 3 |

MMC104

First Semester MCA Degree Examination, Dec.2025/Jan.2026
Operating Systems

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	CO
Q.1	a.	Define an Operating System. Discuss various Computing Environments.	08	L1	CO1
	b.	Define Process. Explain Process Control Block (PCB). Discuss various states of process with state transition diagram.	12	L1	CO2
		OR			
Q.2	a.	Define layered architecture. Explain the layered architecture of Unix operating System.	10	L1	CO1
	b.	Explain Inter-process Communication in detail.	10	L2	CO2
		Module – 2			
Q.3	a.	Explain Peterson algorithm for 2-Process solution	10	L2	CO2
	b.	The processes with given BT, determine AWT and ATT for FCFS,SJF (Non Pre-emptive) , Priority(High Value High Priority) and RR with QT=2.	10	L3	CO2

Processes	P1	P2	P3	P4	P5
BT	2	1	8	4	5
Priority	2	1	4	2	3

		OR			
Q.4	a.	Define semaphore. Explain operations on semaphore.	05	L1	CO2
	b.	Explain Reader-Writer problem using semaphore.	05	L2	CO2
	c.	Illustrate Dining philosopher problem using semaphore with an example.	10	L3	CO2
		Module – 3			
Q.5	a.	Illustrate Banker's Algorithm for safety and Resource-Request for deadlock avoidance along with data structures used in it with an example.	10	L3	CO2
	b.	Define Deadlock. Explain deadlock prevention strategies.	10	L1	CO2
		OR			
Q.6	a.	Five processes with 4 resource type A=3 , B=17 , C=16 , D=12 For the problem determine need matrix and compute safe sequence	10	L3	CO2

Process	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 1 1 0	0 2 1 0	1 5 2 0
P1	1 2 3 1	1 6 5 2	
P2	1 3 6 5	2 3 6 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

	b.	Explain Deadlock detection for Single instance and Multiple instances.	10	L2	CO2

		Module – 4			
Q.7	a.	Illustrate Paging with Hardware and TLB.	10	L3	CO3
	b.	Explain Segmentation with a neat diagram.	05	L2	CO3
	c.	Explain different strategies used in contiguous memory allocation.	05	L2	CO3
		OR			
Q.8	a.	Elaborate Steps involved in handling page fault with neat diagram.	08	L3	CO3
	b.	For the following page reference string: 2 3 2 1 5 2 4 5 3 2 5 2 How many page faults would occur in FIFO, LRU and Optimal technique with frame numbers as 3.	12	L2	CO3
		Module – 5			
Q.9	a.	Discuss on the File Access Methods in details	10	L2	CO3
	b.	Discuss the various Directory structures.	10	L2	CO3
		OR			
Q.10	a.	Define File. Explain the various operations performed on a file.	08	L1	CO3
	b.	Discuss three secondary storage allocation methods.	12	L2	CO3

* * * * *

FIRST SEMESTER MCA DEGREE EXAMINATION, DEC 2025/JAN 2026

MMC104-OPERATING SYSTEM

Module-1

Q1.a. Define an operating system. Discuss various computing environments.

Operating system:

An operating system (OS) is a software program that acts as an intermediary between a user and their computer's hardware, allowing users to interact with and use their devices.

Types of Computing Environments :

Personal Computing Environment : In personal computing environment there is a stand-alone machine. Complete program resides on computer and executed there. Different stand-alone machines that constitute a personal computing environment are laptops, mobiles, printers, computer systems, scanners etc. That we use at our homes and offices.

Time-Sharing Computing Environment : In Time Sharing Computing Environment multiple users share system simultaneously. Different users (different processes) are allotted different time slice and processor switches rapidly among users according to it. For example, student listening to music while coding something in an IDE. Windows 95 and later versions, Unix, IOS, Linux operating systems are the examples of this time sharing computing environment.

Client Server Computing Environment : In client server computing environment two machines are involved i.e., client machine and server machine, sometime same machine also serve as client and server. In this computing environment client requests resource/service and server provides that respective resource/service. A server can provide service to multiple clients at a time and here mainly communication happens through computer network.

Distributed Computing Environment : In a distributed computing environment multiple nodes are connected together using network but physically they are separated. A single task is performed by different functional units of different nodes of distributed unit. Here different programs of an application run simultaneously on different nodes, and communication happens in between different nodes of this system over network to solve task.

Grid Computing Environment : In grid computing environment, multiple computers from different locations works on single problem. In this system set of computer nodes running in cluster jointly perform a given task by applying resources of multiple computers/nodes. It is network of computing environment where several scattered resources provide running environment for single task.

Cloud Computing Environment : In cloud computing environment on demand availability of computer system resources like processing and storage are availed. Here computing is not done in individual

technology or computer rather it is computed in cloud of computers where all required resources are provided by cloud vendor. This environment primarily comprised of three services i.e software-as-a-service (SaaS), infrastructure-as-a-service (IaaS), and platform-as-a-service (PaaS).

Cluster Computing Environment : In cluster computing environment cluster performs task where cluster is a set of loosely or tightly connected computers that work together. It is viewed as single system and performs task parallelly that's why also it is similar to parallel computing environment. Cluster aware applications are especially used in cluster computing environment.

Advantages of different computing environments:

Mainframe: High reliability, security, and scalability, making it suitable for mission-critical applications.

Client-Server: Easy to deploy, manage and maintain, and provides a centralized point of control.

Cloud Computing: Cost-effective and scalable, with easy access to a wide range of resources and services.

Mobile Computing: Allows users to access information and applications from anywhere, at any time.

Grid Computing: Provides a way to harness the power of multiple computers for large-scale computations.

Embedded Systems: Enable the integration of software into devices and products, making them smarter and more functional.

Disadvantages of different computing environments:

Mainframe: High cost and complexity, with a significant learning curve for developers.

Client-Server: Dependence on network connectivity, and potential security risks from centralized data storage.

Cloud Computing: Dependence on network connectivity, and potential security and privacy concerns.

Mobile Computing: Limited processing power and memory compared to other computing environments, and potential security risks.

Grid Computing: Complexity in setting up and managing the grid infrastructure.

Embedded Systems: Limited processing power and memory, and the need for specialized skills for software development

Q1.b. Define Process. Explain process control block(PCB).Discuss various states of process with state transition diagram.

Process: A process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables. A process may also include a heap, which is memory that is dynamically allocated during process run time.

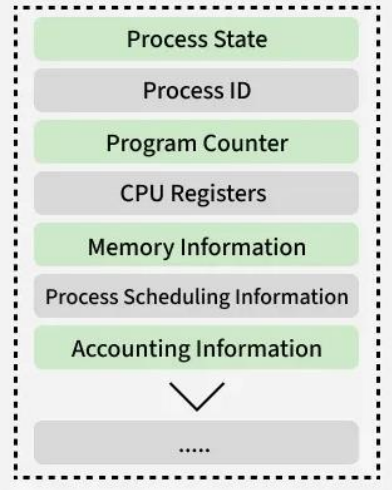
Process Control Block

A Process Control Block (PCB) is a data structure used by the operating system to keep track of process information and manage execution. It helps the OS monitor and control process execution

- Each process is given a unique Process ID (PID) for identification.
- The PCB stores details such as process state, program counter, stack pointer, open files, and scheduling info.
- During a state transition, the OS updates the PCB with the latest execution data.
- It also includes register values, CPU quantum, and process priority.
- The Process Table is an array of PCBs that maintains information for all active processes.

Terminologies used in Process Control Block

- **Process state:** Stores whether the process is running, waiting, ready, or terminated
- **Process number Or PID:** Every process is assigned a unique id known as process ID or PID.
- **Program counter:** Program Counter stores the address of the next instruction that is to be executed for the process.
- **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
- **List of Open files:** This information includes the list of files opened for a process.



Applications

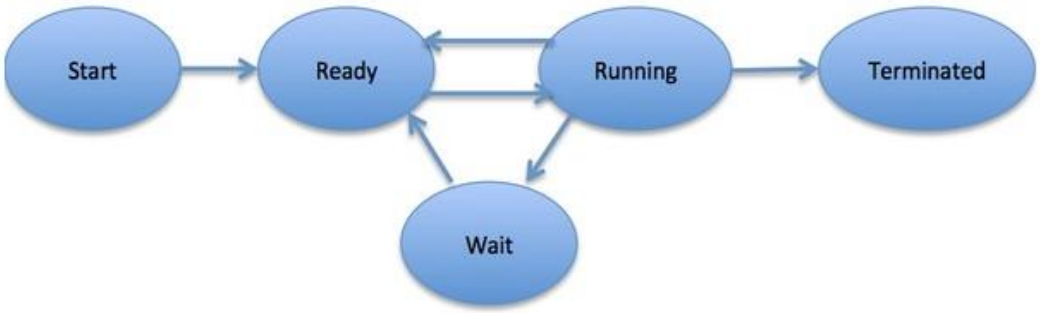
- Helps the operating system schedule processes and allocate CPU resources efficiently.
- Enables efficient resource utilization and sharing by providing detailed resource-usage information.
- Supports context switching through stored CPU registers and stack pointer information.
- Assists in process synchronization by keeping track of waiting states and required resources.
- Tracks process states and resource usage to determine which process should be executed next.

Process State

The five-state process lifecycle expands the basic two-state idea by separating processes that are simply waiting for CPU time from those that cannot run because they are waiting for an external event. This makes the model more accurate for real operating systems, where processes may pause for input, data, or device responses before continuing.

The five states are:

- **New:** The process has just been created. It hasn't started running yet, but its PCB (Process Control Block) is prepared.
- **Ready:** The process is loaded into memory and prepared to run as soon as the CPU becomes free.
- **Running:** The CPU is currently executing this process. With a single CPU system, only one process can be in this state at a time.
- **Blocked/Waiting:** The process cannot continue execution because it is waiting for an event like I/O completion or data input.
- **Exit/Terminate:** The process has completed its execution or has been stopped, and the operating system removes it from memory.



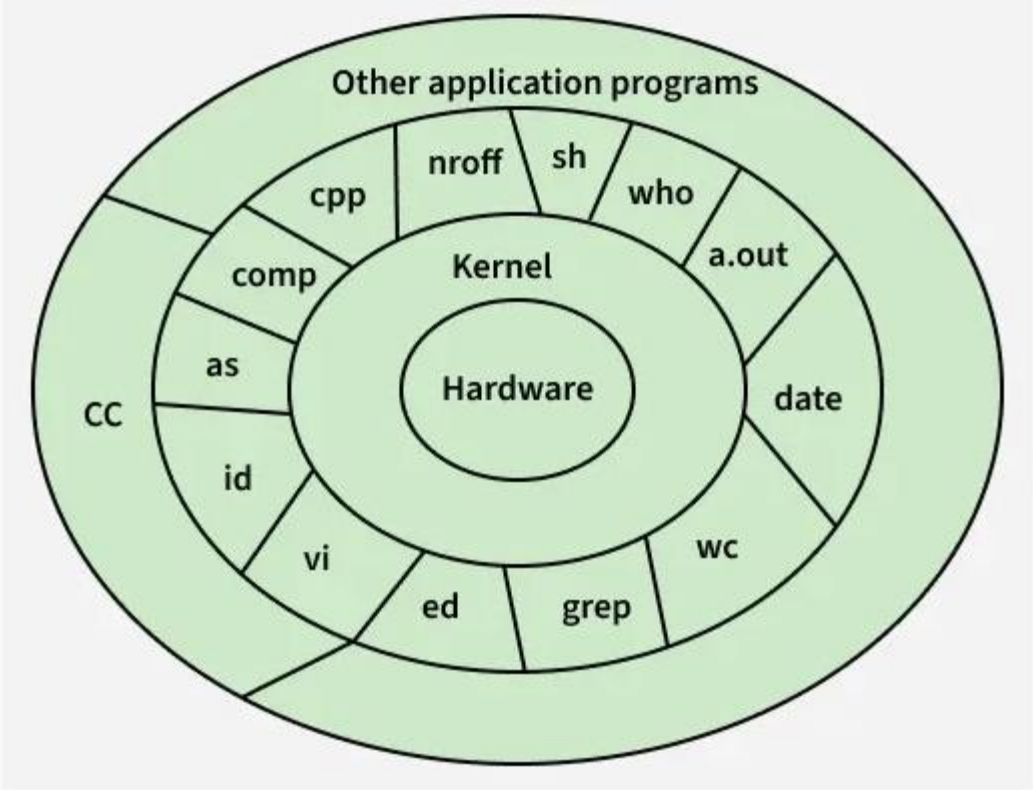
OR

Q2.a. Define layered architecture. Explain the layered architecture of Unix operating system.

Layered architecture is a design approach in which an operating system is divided into different layers (levels), each built on top of the lower layer.

- Each layer performs a specific function.
- A layer only interacts with the layer directly below or above it.
- This improves modularity, debugging, and system maintenance.

The layered structure of the UNIX operating system organizes its components into hierarchical levels-hardware, kernel, system utilities, and applications to ensure efficient, secure, and modular system functioning.



structure of the UNIX operating system

The following points describe the role and functioning of each layer in the UNIX architecture:

Layer-1: Hardware

- Represents the physical components of the computer, such as the CPU, memory, storage devices, and input/output hardware.
- All system operations depend on this layer, as it forms the base of the entire UNIX architecture.

Layer-2: Kernel

- The kernel is the core of the UNIX operating system and directly interacts with the hardware.
- It manages essential tasks such as memory allocation, process scheduling, file handling, device control, and overall resource management.

Layer-3: System Programs, Commands, and Utilities

- This layer includes compilers (cc, as, ld), editors (vi, ed), text processors (nroff), and the shell (sh).

- It also contains common UNIX tools and commands like wc, grep, date, who, and a.out, which rely on the kernel to perform different functions such as text processing, file management, and system monitoring.

Layer-4: Application Layer

- Represents user-developed or external application programs that run on top of system utilities and shell environment.
- These applications use underlying layers (system utilities and kernel) to execute tasks successfully and provide services to the user.

Q2.b. Explain inter process communication in detail.

Inter-Process Communication or IPC is a mechanism that allows processes to communicate and share data with each other while they are running. Since each process has its own memory space, IPC provides controlled methods for exchanging information and coordinating actions. It helps processes work together efficiently and safely in an operating system.

- It helps processes synchronize their activities, share information and avoid conflicts while accessing shared resources.
- There are two method of IPC, shared memory and message passing. An operating system can implement both methods of communication.

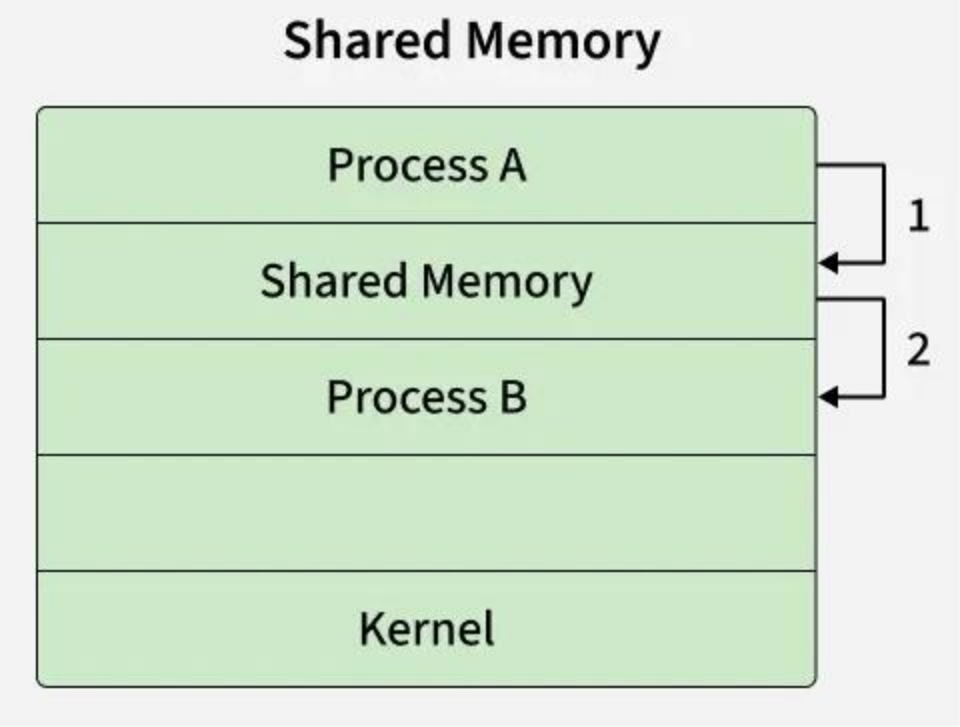
Example: A simple example of IPC is a bank ATM system, where one process reads the card and PIN, another checks the account balance, and a third dispenses cash. These processes communicate and coordinate to complete the transaction correctly.

Shared Memory

Communication between processes using shared memory requires processes to share some variable and it completely depends on how the programmer will implement it. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Shared Memory

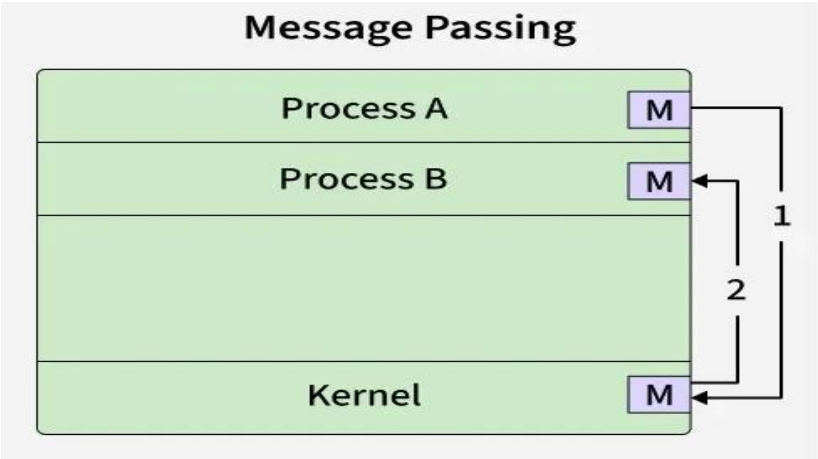
- In the above shared memory model, a common memory space is allocated by the kernel.
- Process A writes data into the shared memory region (Step 1).
- Process B can then directly read this data from the same shared memory region (Step 2).
- Since both processes access the same memory segment, this method is fast but requires synchronization mechanisms (like semaphores) to avoid conflicts when multiple processes read/write simultaneously.
- Example: Multiple people can edit the document at the same time in shared google doc.



Shared Memory

Message Passing

Message Passing is a method where processes communicate by sending and receiving messages to exchange data. One process sends a message and the other process receives it, allowing them to share information. Message Passing can be achieved through different methods like Sockets, Message Queues or Pipes.



In the above message passing model, processes exchange information by sending and receiving messages through the kernel.

- Process A sends a message to the kernel (Step 1).
- The kernel then delivers the message to Process B (Step 2).
- Here, processes do not share memory directly. Instead, communication happens via system calls (send(), recv(), or similar).
- This method is simpler and safer than shared memory because there's no risk of overwriting shared data, but it incurs more overhead due to kernel involvement.
- Example: Multiple people send updates to a group chat, but each message goes through the server before others see it, like processes sending messages instead of directly sharing memory.

Problems in Inter-Process Communication (IPC):

Inter-Process Communication (IPC) faces challenges when multiple processes share resources. Improper synchronization can cause race conditions, deadlock, and starvation, while shared data may suffer data inconsistency. IPC also adds overhead and can raise security issues. Managing many processes can lead to scalability problems.

Some common classical IPC problem are:

Dining Philosophers Problem
This problem illustrates deadlock and starvation. The Dining Philosophers Problem involves five philosophers sitting around a table, each needing two forks (shared resources) to eat. If all philosophers pick up one fork at the same time, none can eat, resulting in deadlock.
Solution
- Use semaphores or monitors to control access to forks.
- Allow only one philosopher to pick forks at a time or limit eating to four philosophers.
- Enforce an order of picking forks to avoid circular wait.
- Prevents deadlock and starvation.

Producer–Consumer Problem
This problem deals with synchronization and buffer management. The Producer–Consumer Problem describes producers generating data and placing it in a shared buffer, while consumers remove data from it. The main challenge is preventing producers from adding data to a full buffer and consumers from removing data from an empty buffer.
Solution
- Use mutex to ensure mutual exclusion on the shared buffer.
- Use counting semaphores to track empty and full buffer slots.
- Producer waits if buffer is full; consumer waits if buffer is empty.
- Ensures proper synchronization and data consistency.

Readers–Writers Problem

This problem focuses on concurrent access to shared data. The Readers–Writers Problem allows multiple readers to read data simultaneously, while writers require exclusive access. The challenge is avoiding starvation of either readers or writers.
Solution

- Use reader–writer locks or semaphores.
- Allow multiple readers to read simultaneously.
- Grant writers exclusive access to shared data.
- Apply priority rules to avoid starvation.

Sleeping Barber Problem
This problem demonstrates process coordination. The Sleeping Barber Problem models a barber who sleeps when there are no customers and is awakened when a customer arrives and a chair is available. The difficulty lies in managing waiting chairs and customer arrivals correctly.
Solution

- Use semaphores to manage customer arrival and barber availability.
- Barber sleeps when no customers are present.
- Customers wait if chairs are available; otherwise, they leave.
- Ensures proper process coordination without deadlock.

Module-2

Q3.a. Explain Peterson algorithm for 2-process solution

Peterson's Algorithm is a classic software-based solution for the critical section problem in operating systems. It ensures mutual exclusion between two processes, meaning only one process can access a shared resource at a time, thus preventing race conditions.

The algorithm uses two shared variables:

- **flag[i]:** shows whether process i wants to enter the critical section.
- **turn:** indicates whose turn it is to enter if both processes want to access the critical section at the same time.

The Algorithm
For process Pi:

```
do {
flag[i] = true; // Pi wants to enter
turn = j; // Give turn to Pj
while (flag[j] && turn == j); // Wait if Pj also wants to enter
// Critical Section
flag[i] = false; // Pi leaves critical section
// Remainder Section
} while (true);
```

For process Pj:

```
do {
flag[j] = true;
turn = i;
```

while (flag[i] && turn == i);
// Critical Section
flag[j] = false;
// Remainder Section
} while (true);

Step-by-Step Explanation

1. **Intent to Enter:** A process sets its flag to true when it wants to enter the critical section.
2. **Turn Assignment:** It sets the turn variable to the other process, giving the other process the chance to enter first if it also wants to.
3. **Waiting Condition:** A process waits if the other process also wants to enter and it is the other's turn.
4. **Critical Section:** Once the condition is false, the process enters the critical section safely.
5. **Exit:** On leaving, the process resets its flag to false, allowing the other process to proceed.

This guarantees:

- **Mutual Exclusion:** Only one process enters CS.
- **Progress:** A process will eventually enter CS if no other is inside.
- **Bounded Waiting;** No process waits indefinitely.

Example Use Cases
Peterson's Algorithm can be used (theoretically) in:

- **Accessing a shared printer**: Peterson's solution ensures that only one process can access the printer at a time when two processes are trying to print documents.
- **Reading and writing to a shared file**: It can be used when two processes need to read from and write to the same file, preventing concurrent access issues.
- **Competing for a shared resource**: When two processes are competing for a limited resource, such as a network connection or critical hardware, Peterson's solution ensures mutual exclusion to avoid conflicts.

Q3.b. The process with given BT ,determine AWT and ATT for FCFS,SJF (Non Preemptive) ,Priority(High value high priority) and RR with QT=2

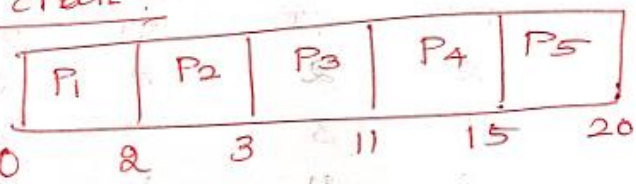
Process	P1	P2	P3	P4	P5
BT	2	1	8	4	5
Priority	2	1	4	2	3

CPU scheduling Algorithm

Soln:- i) FCFS (Non-preemptive)

Process	BT	AT	CT	WT	TAT
P_1	2	0	2	0	2
P_2	1	0	3	2	3
P_3	8	0	11	3	11
P_4	4	0	15	11	15
P_5	5	0	20	15	20

Gantt chart :-



Formula :-

(TAT) Turn Around time = CT − AT

Waiting time ⇒ WT = TAT − BT

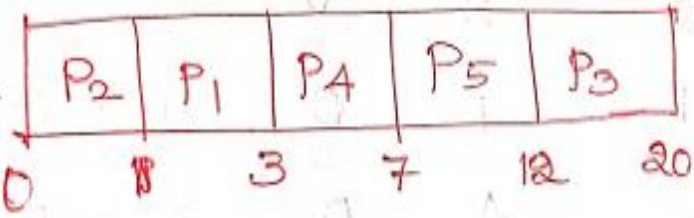
$$ATAT = \frac{2+3+11+15+20}{5} = 5\frac{1}{5} = 10.2$$

$$AWT = \frac{0+2+3+11+15}{5} = 3\frac{1}{5} = 6.2$$

ii) SJF (Non-Preemptive)

Process	BT	AT	CT	WT	TAT
P1	2	0	3	1	3
P2	1	0	1	0	1
P3	8	0	20	12	20
P4	4	0	7	3	7
P5	5	0	12	7	12

Gantt Chart:-



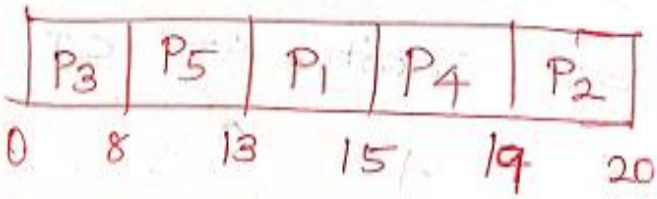
$$ATAT = \frac{3+1+20+7+12}{5} = \frac{43}{5} = 8.6$$

$$AWT = \frac{1+0+12+3+7}{5} = \frac{23}{5} = 4.6$$

iii) Priority (High value high Priority)

Process	BT	Priority	AT	CT	WT	TA
P_1	2	2	0	15	13	15
P_2	1	1	0	20	19	20
P_3	8	4	0	8	0	8
P_4	4	2	0	19	15	19
P_5	5	3	0	13	8	13

Gantt chart:-



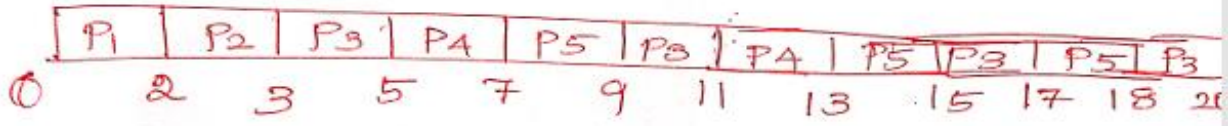
$$ATAT = \frac{15+20+8+19+13}{5} = \frac{75}{5} = 15$$

$$AWT = \frac{13+19+0+15+8}{5} = \frac{55}{5} = 11$$

iv) Round Robin (TQ = 2 ms)

Process	BT	AT	CT	WT	TAT
P_1	~~2~~ 0	0	2	0	2
P_2	~~1~~ 0	0	3	2	3
P_3	~~8 6 4 2~~ 0	0	20	12	20
P_4	~~4 2~~ 0	0	13	9	13
P_5	~~5 3~~ 1	0	18	13	18

Gantt chart:-



$$ATAT = \frac{2+3+20+13+18}{5} = \frac{56}{5} = 11.2$$

$$AWT = \frac{0+2+12+9+13}{5} = \frac{36}{5} = 7.2$$

Final Answer:-

Algorithm	AWT	ATAT
FCFS (Non Preemptive)	6.2	10.2
SJF (Non Preemptive)	4.6	8.6
Priority (High value high priority)	11	15
RR (QT=2)	7.2	11.2

OR

Q4 a. Define Semaphore. Explain operations on semaphore.

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows ?

Wait

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
  while (S<=0);

  S--;
}
```

Signal

The signal operation increments the value of its argument S.

```
signal(S)
{
  S++;
}
```

Types of Semaphores

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows ?

Counting Semaphores

These are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the

resources are added, semaphore count automatically incremented and if the resources are removed, the count is decremented.

Binary Semaphores

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Advantages of Semaphores

Some of the advantages of semaphores are as follows

Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.

There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.

Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores

Some of the disadvantages of semaphores are as follows ?

Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.

Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.

Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

Q4. b. Explain Reader-writer problem using semaphore

A data set is shared among a number of concurrent processes

– Readers – only read the data set, do not perform any updates

– Writers – can both read and write the data set (perform the updates).

• If two readers read the shared data simultaneously, there will be no problem. If both a

reader(s) and writer share the same data simultaneously then there will be a problem.

• In the solution of reader-writer problem, the reader process share the following data

structures: Semaphore Mutex, wrt;

int readcount;

• Where Semaphore mutex is initialized to 1.

 Semaphore wrt is initialized to 1.

 Integer readcount is initialized to 0.

The structure of a writer process

while (true) {

wait (wrt) ;

// writing is

performed signal

(wrt) ;

}

The structure of a reader process

while (true) {

wait (mutex) ;

readcount ++ ;

if (readcount == 1) wait

(wrt) ; signal (mutex)

// reading is performed

wait (mutex) ; readcount -

- ;

if (readcount == 0) signal

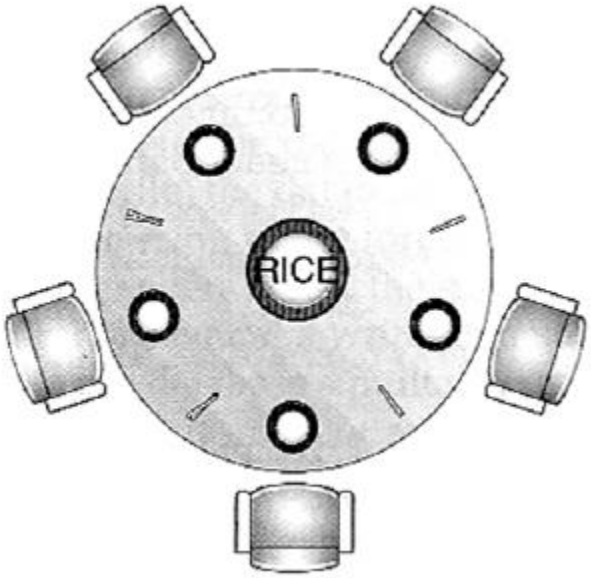
(wrt) ; signal (mutex) ;

}

Q4. c. Illustrate Dining Philosopher problem using semaphore with an example.

Dining-Philosophers Problem

Five philosophers are seated on 5 chairs across a table. Each philosopher has a plate full of noodles. Each philosopher needs a pair of forks to eat it. There are only 5 forks available all together. There is only one fork between any two plates of noodles. In order to eat, a philosopher lifts two forks, one to his left and the other to his right. if he is successful in obtaining two forks, he starts eating after some time, he stops eating and keeps both the forks down.



What if all the 5 philosophers decide to eat at the same time ?

All the 5 philosophers would attempt to pick up two forks at the same time. So,none of them succeed. One simple solution is to represent each fork with a semaphore.a philosopher tries to grab a fork by executing wait() operation on that semaphore.he releases his forks by executing the signal() operation.This solution guarantees that no two neighbours are eating simultaneously. Suppose all 5 philosophers become hungry simultaneously and each grabs his left fork,he will be delayed forever.

The structure of Philosopher i:

```
do{
wait ( chopstick[i] );
wait ( chopStick[ (i + 1) % 5] );
// eat
signal ( chopstick[i] );
signal (chopstick[ (i + 1) % 5] );
// think
} while (TRUE);
```

Several remedies:

1) Allow at most 4 philosophers to be sitting simultaneously at the table.

2) Allow a philosopher to pickup his fork only if both forks are available.

3) An odd philosopher picks up first his left fork and then right fork. an even philosopher picks up his right fork and then his left fork.

Module-3

Q5.a. Illustrate Banker's algorithm for safety and resource request for deadlock avoidance along with data structures used in it with an example.

The Banker's Algorithm for resource allocation and deadlock prevention is useful when allocating resources to several processes in an operating system and keeping the system safe. It tracks the maximum resources a process may utilise, the current resources allocated for the processes, and how many resources are available. When a process requests for the allocation of resources, the algorithm checks the system to ensure granting the request will not lead the system into a deadlock state by checking if it can eventually attain a safe state where all processes finish. If it cannot, that request will be declined, thus avoiding potential deadlock.

Data Structures Used in Banker's Algorithm

Implementing the Banker's Algorithm in an operating system adopts several key data structures, which aid in administrating and tracking resources and processes within the system. The following data structures are required:

Available: A vector of length "m" that tracks the number of available instances for each resource type in the system. For example, if Available[j] = k, there are k instances of resource type Rj available.

Max: A matrix of size [n × m] expressing the maximum number of resources each process can demand. If Max[i][j] = k, process Pi can request up to k instances of resource Rj.

Allocation: A matrix of size [n × m] representing the number of resources allocated to each process. For example, Allocation[i][j] = k means that process Pi has been allocated k of resource Rj.

Need: A matrix of size [n × m] that provides the number of resources still needed for each process. For example, Need[i][j] = k means process Pi still needs k more instances of resource Rj to complete.

Finish: A vector of length n, where each entry is a Boolean value (TRUE or FALSE). It shows if the process has finished executing and released all its resources. Hence, if Finish[i] = TRUE, process Pi is said to have completed execution with the resources freed.

Key Concepts in Banker's Algorithm

Here are the key concepts in banker's algorithm:

- **Safe State:** A system is said to be in a secure state when there is a sequence of processes such that by each process obtaining required resources, finishing, and then releasing resources that might be needed by other methods, such other processes have a possibility of finishing. In a safe state, no deadlock can occur in the system.
- **Unsafe State:** It is called unsafe when resources may still be allocated to processes, allowing them to proceed but with no guarantees that all will be completed without causing deadlocks. In this case, a deadlock occurs, and the process may not finish.

There are two types of banker's algorithms such as:

Safety Algorithm

Here are the steps involved in the safety algorithm:

Step 1: The algorithm will initialise vectors Work and Finish.

```
Initialize: Work = Available

Finish[i] = false; for i = 1, 2, 3, 4....n
```

Step 2: It searches for processes not finished yet for available resources.

```
Find an i such that both:

   2.1 Finish[i] = false

   2.2 Needi <= Work

   if no such i exists, go to step (4)
```

Step 3: The processes are finished after available resources are updated.

```
Work = Work + Allocation[i]

Finish[i] = true

Go to step (2)
```

Step 4: It ensures a safe state if all processes are successfully finished.

```
If Finish[i] = true for all i, then the system is in a safe state
```

Resource-Request Algorithm

Here are the steps for resource request algorithm:

Step 1: This requires a check to see if the request for resources by a process is valid

```
If Requesti <= Needi, go to step (2);
```

Step 2: The process request for resources does not exceed the maximum claim.

```
If Requesti <= Available, go to step (3);
```

Step 3: If requested resources are unavailable in the system, the process waits until they are released.

```
Available = Available - Requesti

Allocationi = Allocationi + Requesti

Needi = Needi - Requesti
```

Example of Banker's Algorithm in OS

Here is the banker's algorithm example with solution. We have the following tables for processes, allocations, maximum needs, and available resources:

Process Table:

Process	Allocation (A, B, C)	Max (A, B, C)	Availabl
P0	(1, 2, 5)	(4, 3, 2)	(3, 2, 1)
P1	(2, 1, 2)	(4, 3, 2)	
P2	(0, 1, 9)	(6, 2, 3)	

P3	(2, 0, 8)	(6, 4, 4)	
P4	(1, 1, 2)	(2, 2, 5)	

1. Calculate the Need Matrix

The Need matrix is calculated using the formula:

```
Need[i]=Max[i]−Allocation[i]
```

So for each process:

Process	Need (A, B, C)
P0	(4-1, 3-2, 2-5) = (3, 1, -3)
P1	(4-2, 3-1, 2-2) = (2, 2, 0)
P2	(6-0, 2-1, 3-9) = (6, 1, -6)
P3	(6-2, 4-0, 4-8) = (4, 4, -4)
P4	(2-1, 2-1, 5-2) = (1, 1, 3)

2. Is the system in a safe state? If Yes, then what is the safe sequence?

To determine if the system is in a safe state, we will follow the Safety Algorithm.

Step 1: Initialize Work and Finish

- Work = Available resources = (3, 2, 1)
- Finish = [False, False, False, False, False] (None of the processes are finished initially)

Step 2: Find a process that can finish

Look for a process i where Need[i] <= Work.

- P0: Need = (3, 1, -3), but this contains a negative value, which is invalid. Therefore, P0 cannot finish.
- P1: Need = (2, 2, 0) and Work = (3, 2, 1). Since Need ≤ Work, P1 can finish.
    - After P1 finishes, update Work:
      Work = Work + Allocation[P1] = (3, 2, 1) + (2, 1, 2) = (5, 3, 3)
    - Finish[P1] = True.
- P2: Need = (6, 1, -6), but this contains a negative value, so P2 cannot finish.
- P3: Need = (4, 4, -4), which also contains a negative value, so P3 cannot finish.
- P4: Need = (1, 1, 3) and Work = (5, 3, 3). Since Need ≤ Work, P4 can finish.
    - After P4 finishes, update Work:
      Work = Work + Allocation[P4] = (5, 3, 3) + (1, 1, 2) = (6, 4, 5)
    - Finish[P4] = True.

Step 3: Repeat the process

Now, we have the updated Work = (6, 4, 5). We will check again for other processes.

- P2: Need = (6, 1, -6), but this still contains a negative value, so P2 cannot finish.
- P3: Need = (4, 4, -4), which also contains a negative value, so P3 cannot finish.

- P0: Need = (3, 1, -3), but again, there are negative values, so P0 cannot finish.

Safe Sequence:

- Since P1 and P4 can finish, we have a safe <P1, P4> sequence.

Q5. b. Define Deadlock. Explain deadlock prevention strategies.

The first method to handle a deadlock is to prevent it from occurring in the first place itself. This is known as Deadlock Prevention. Read this chapter to learn all the methods used for deadlock prevention in operating systems.

How to Prevent Deadlock in an Operating System?

A deadlock will occur only if all the following four necessary conditions are occurring simultaneously in the system −

Mutual Exclusion

Hold and Wait

No Preemption

Circular Wait

To prevent a deadlock, the operating system just needs to ensure that at least one of the above four conditions never takes place. In the next section, we will see how to prevent each of these conditions from occurring.

Prevent Mutual Exclusion

Mutual Exclusion is a condition that states to occur deadlock in an operating system, **at least one resource** must be held in a **non-shareable mode**, meaning that only one process can use that resource at any given time. If another process requests the same resource, it must wait for the resource to be released.

To prevent occurring of **mutual exclusion**, the operating system can **make resources shareable** whenever possible. For example, read-only resources such as files can be shared among multiple processes simultaneously without causing any issues. But **writeable resources** like printers or memory **cannot be shared** to multiple processes at the same time. Because when two processes try to write data to the same resource simultaneously, it can cause data corruption and inconsistency.

So preventing mutual exclusion is not always a choice for deadlock prevention. But whenever possible, the operating system should make resources shareable to avoid deadlock.

Prevent Hold and Wait

Hold and Wait is a condition that occurs when a process holding at least one resource is waiting to get additional resources that are currently being held by other processes. This means that a process that already has resources may request more resources, which it cannot obtain because they are held by other processes.

To prevent hold and wait, the operating system can use one of the following two strategies −

- **Declare the Needed Resources Upfront** − Before a process starts executing, it must request and be allocated all the resources it will need during its execution. This is known as the **"all-or-nothing"** approach. If all the requested resources are not available, the process should wait until they are available.
- **Release Resources Before Requesting New Ones** − If at any point during its execution, a process needs additional resources, it must first release all the resources it currently holds before requesting the new ones. This ensures that a process does not hold onto resources while waiting for others.

Both of these strategies can help prevent hold and wait. However, they can also lead to decreased resource utilization and increased waiting times for processes. So, the operating system must carefully consider the trade-offs when implementing these strategies. End of the day, preventing preventing deadlock comes with some cost.

Example

Consider two processes P1 and P2 and two resource types R1 and R2.

Process P1 - Needs R1 and R2
Process P2 - Needs R1 and R2

Scenario 1 − Without Preventing Hold and Wait

P1 - Allocated R1, Waiting for R2
P2 - Allocated R2, Waiting for R1
Result: Deadlock occurs.

Scenario 2 − Preventing Hold and Wait by Declaring Resources Upfront

P1 - Requests R1 and R2 together

P1 – Allocated R1 and R2

P1 – Completes execution and releases R1 and R2

P2 – Requests R1 and R2 together

P2 – Allocated R1 and R2

P2 – Completes execution and releases R1 and R2

Result: No Deadlock occurs.

Prevent No Preemption

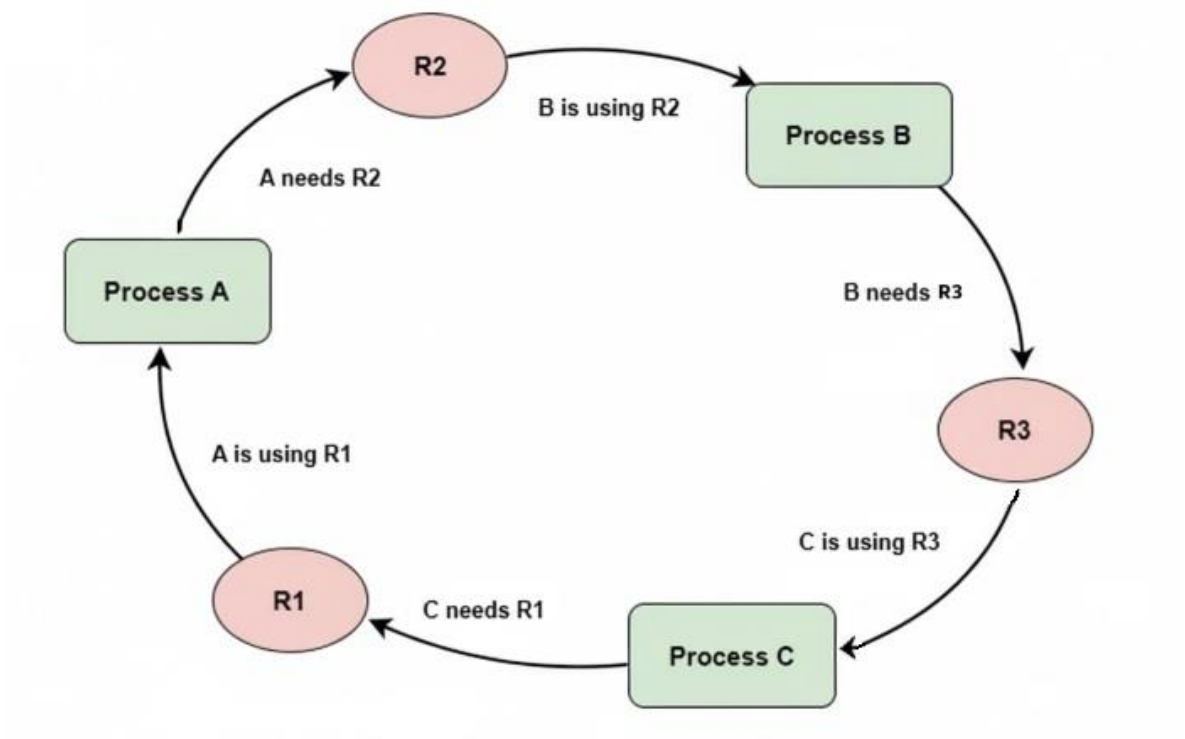
No Preemption is a condition that states resources cannot be taken away from a process holding them. Meaning a process must release the resource voluntarily when it is no longer needed.

To prevent no preemption condition, the operating system can **allow preemption** of resources. If a process holding some resources is waiting for additional resources, the operating system can preempt the resources currently held by the process and assign them to other processes. The preempted process can be restarted later when it can get all the resources it needs.

However, preempting resources can lead to issues such as **data inconsistency** and **starvation**. So, the operating system must carefully manage the preemption process to ensure that it does not lead to other problems.

Prevent Circular Wait

Circular Wait says that a set of processes exist such that each process in the set is waiting for a resource that another process in the set holds, thus forming a cycle of waiting. It can be visualized as in the diagram below −



To prevent circular wait, the operating system can implement a total ordering method. In this method, **all resource** types are assigned a **unique numerical order**. Each process is required to request resources in an **increasing order** of their enumeration. This means that a process can only request a resource with a higher number than the resources it currently holds. By implementing this order, the operating system can prevent the formation of circular wait conditions among processes.

Example

Consider three resource types R1, R2, and R3 with the following order −

- R1 - 1
- R2 - 2
- R3 - 3

Process P1 - Holds R1, Can request R2 or R3 Process P2 - Holds R2, Can request R3 Process P3 - Holds R3, Cannot request any other resource as of now Result: No Circular Wait can occur.

OR

Q6. a. Five process with four resource type A=3,B=17,C=16,D=12.For the problem determine need matrix and compute safe sequence

Process	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 1 1 0	0 2 1 0	1 5 2 0
P1	1 2 3 1	1 6 5 2	
P2	1 3 6 5	2 3 6 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

Step 1: Given Data

Allocation Matrix

Process	A B C D
P0	0 1 1 0
P1	1 2 3 1
P2	1 3 6 5
P3	0 6 3 2
P4	0 0 1 4

Max Matrix

Process	A B C D
P0	0 2 1 0
P1	1 6 5 2
P2	2 3 6 6
P3	0 6 5 2
P4	0 6 5 6

Available

A B C D = **1 5 2 0**

Step 2: Compute Need Matrix

Formula:
Need = Max − Allocation

Process	A B C D
P0	0 1 0 0
P1	0 4 2 1
P2	1 0 0 1
P3	0 0 2 0
P4	0 6 4 2

Step 3: Safety Algorithm Execution

Initial Available:

Work = (1, 5, 2, 0)

Step 1: Select P0

Need(P0) = (0,1,0,0) ≤ (1,5,2,0) ✓
New Work = Work + Allocation(P0)
= (1,5,2,0) + (0,1,1,0) = **(1,6,3,0)**

Step 2: Select P3

Need(P3) = (0,0,2,0) ≤ (1,6,3,0) ✓
New Work = (1,6,3,0) + (0,6,3,2) = **(1,12,6,2)**

Step 3: Select P1

Need(P1) = (0,4,2,1) ≤ (1,12,6,2) ✓
New Work = (1,12,6,2) + (1,2,3,1) = **(2,14,9,3)**

Step 4: Select P2

Need(P2) = (1,0,0,1) ≤ (2,14,9,3) ✓
New Work = (2,14,9,3) + (1,3,6,5) = **(3,17,15,8)**

Step 5: Select P4

Need(P4) = (0,6,4,2) ≤ (3,17,15,8) ✓
New Work = (3,17,15,8) + (0,0,1,4) = **(3,17,16,12)**

Step 4: Safe Sequence

Safe Sequence is:
P0 → P3 → P1 → P2 → P4

Final Answer

Need Matrix

```
P0: 0 1 0 0
P1: 0 4 2 1
P2: 1 0 0 1
P3: 0 0 2 0
P4: 0 6 4 2
```

Safe Sequence

P0 → P3 → P1 → P2 → P4

Q6. b. Explain Deadlock detection for Single and multiple instances

Deadlock Detection

Deadlock detection is a method used by an operating system to determine whether a deadlock has occurred and to identify the processes involved.

1. Deadlock Detection for Single Instance of Each Resource Type

Concept

- Each resource type has **only one instance**.
- Deadlock detection is done using a **Wait-For Graph (WFG)**.

Wait-For Graph (WFG)

- Nodes → Processes
- Edge Pi → Pj means **Pi is waiting for a resource held by Pj**

Detection Method

- Convert Resource Allocation Graph (RAG) into WFG by removing resource nodes.
- If the graph contains a **cycle**, then:

 Cycle ⇒ Deadlock exists

Example

If:

- P1 waits for P2
- P2 waits for P3
- P3 waits for P1

Then:
P1 → P2 → P3 → P1 (Cycle)
✔Deadlock detected

2. Deadlock Detection for Multiple Instances of Resource Types

Concept

- Each resource type can have **multiple instances**.
- Use a detection algorithm similar to **Banker's Algorithm**.

Data Structures Used

- **Available** → Available resources vector
- **Allocation** → Resources allocated to each process
- **Request** → Current request of each process

Detection Algorithm Steps

1. **Initialize:**
    - Work = Available
    - Finish[i] = false for all processes
2. **Find a process Pi such that:**
    - Finish[i] == false
    - Request[i] ≤ Work
3. **If found:**
    - Work = Work + Allocation[i]
    - Finish[i] = true
    - Repeat Step 2
4. **If no such process found:**
    - If any Finish[i] == false → **Deadlock exists**

Difference Between Single and Multiple Instance Detection

Feature	Single Instance	Multiple Instance
Technique	Wait-For Graph	Detection Algorithm
Representation	Graph-based	Matrix-based
Deadlock Condition	Cycle in graph	Unfinished processes
Complexity	Simple	More complex

Module-4

Q7.a. Illustrate paging hardware and TLB

Paging requires hardware support to translate logical addresses into physical addresses. The hardware components involved are

Page Table:

- Maintains a mapping of page numbers to frame numbers.
- Each process has its own page table.

Translation Lookaside Buffer (TLB):

- A high-speed cache storing a subset of the page table for faster access.
- Helps reduce the time required for address translation.

Working of Paging with TLB:

Logical Address Division:

The CPU generates a logical address consisting of:

- Page number (p): Index in the page table.
- Page offset (d): Displacement within the page.

TLB Lookup:

- The page number (p) is searched in the TLB.
- If found (TLB hit), the corresponding frame number is fetched, and the physical address is generated directly.

Page Table Lookup (if TLB Miss):

- If not found in the TLB (TLB miss), the page table in memory is accessed to retrieve the frame number.

Address Translation:

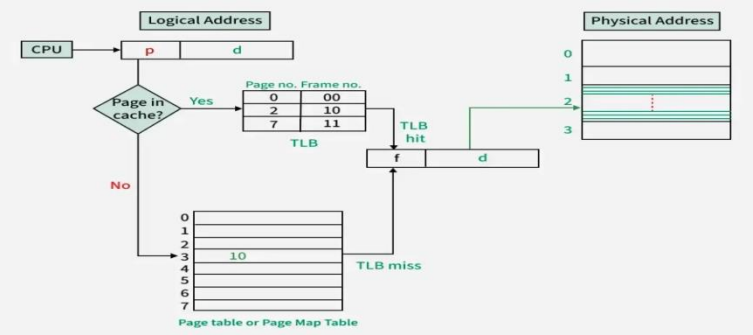
- The frame number and page offset (d) are combined to generate the physical address.

TLB Update:

- If the page table is accessed due to a TLB miss, the new mapping is added to the TLB for future reference.

Advantages of Using TLB

- Faster Address Translation:
- Reduces the overhead of repeatedly accessing the page table.
- Improved System Performance:
- Enhances execution speed by reducing memory access times



Q7.b. Explain segmentation with a neat diagram

In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment

2. Limit: It is the length of the segment.

Why Segmentation is required?

Till now, we were using Paging as our main memory management technique. Paging is more close to Operating system rather than the User. It divides all the process into the form of pages regardless of the fact that a process can have some relative parts of functions which needs to be loaded in the same page.

Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

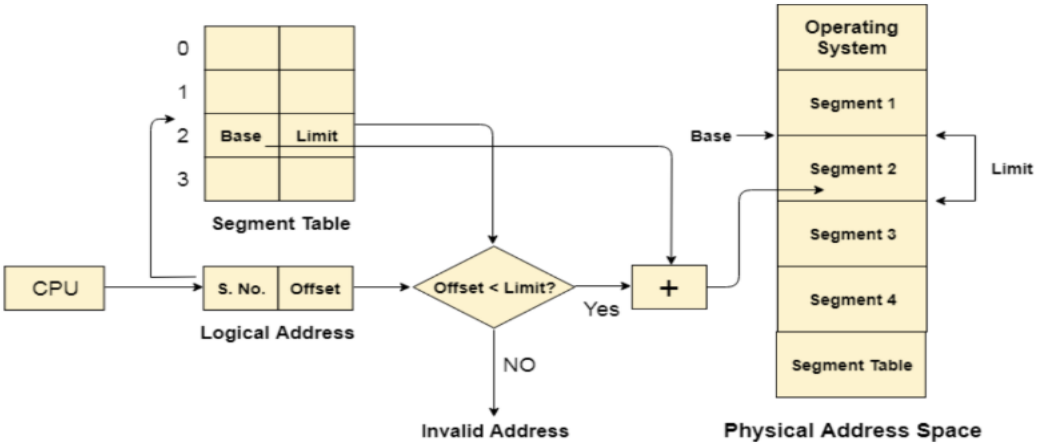
Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

1. Segment Number

2. Offset

The Segment number is mapped to the segment table. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid. In the case of valid address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.



Advantages of Segmentation

1. No internal fragmentation

2. Average Segment Size is larger than the actual page size.

3. Less overhead

4. It is easier to relocate segments than entire address space.

5. The segment table is of lesser size as compare to the page table in paging.

Disadvantages

1. It can have external fragmentation.

2. it is difficult to allocate contiguous memory to variable sized partition.

3. Costly memory management algorithms.

Q7.c. Explain different strategies used in contiguous memory allocation

Contiguous Memory Allocation – Strategies

In **contiguous memory allocation**, each process is allocated a **single continuous block of memory**. When multiple free memory blocks (holes) are available, the OS uses different strategies to decide **where to allocate memory**.

1. First Fit

Concept

- Allocate the **first hole** that is large enough.

Working

- Scan memory from the beginning.
- Pick the first block that satisfies the request.

Advantages

- Fast and simple
- Less searching time

Disadvantages

- May lead to **external fragmentation**
- Larger holes may get broken early

2. Best Fit

Concept

- Allocate the **smallest hole** that is sufficient.

Working

- Search entire memory.
- Choose the block with the **minimum leftover space**.

Advantages

- Reduces wasted space (internal fragmentation)

Disadvantages

- Requires full search → **slower**
- Creates many small unusable holes → **external fragmentation**

3. Worst Fit

Concept

- Allocate the **largest available hole**.

Working

- Select the biggest free block.
- Allocate required memory from it.

Advantages

- Leaves large leftover holes for future use

Disadvantages

- Poor memory utilization
- May waste large blocks unnecessarily

Q8 a. Elaborate steps involved in handling page fault with neat diagram

- The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need (right away.)
- Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. (The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive.)
- If the process only ever accesses pages that are loaded in memory (***memory resident*** pages), then the process runs exactly as if all the pages were loaded in to memory.
- On the other hand, if a page is needed that was not originally loaded up, then a ***page fault trap*** is generated, which must be handled in a series of **steps**:

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU.)

- In an extreme case, NO pages are swapped in for a process until they are requested by page faults. This is known as ***pure demand paging.***
- In theory each instruction could generate multiple page faults. In practice this is very rare, due to ***locality of reference***, covered in section 9.6.1.
- The hardware necessary to support virtual memory is the same as for paging and swapping: A page table and secondary memory. (***Swap space,*** whose allocation is discussed in chapter 12.)
- A crucial part of the process is that the instruction must be restarted from scratch once the desired page has been made available in memory. For most simple instructions this is not a major difficulty. However there are some architectures that allow a single instruction to modify a fairly large block of data, (which may span a page boundary), and if some of the data gets modified before the page fault occurs, this could cause problems. One solution is to access both ends of the block before executing the instruction, guaranteeing that the necessary pages get paged in before the instruction begins.

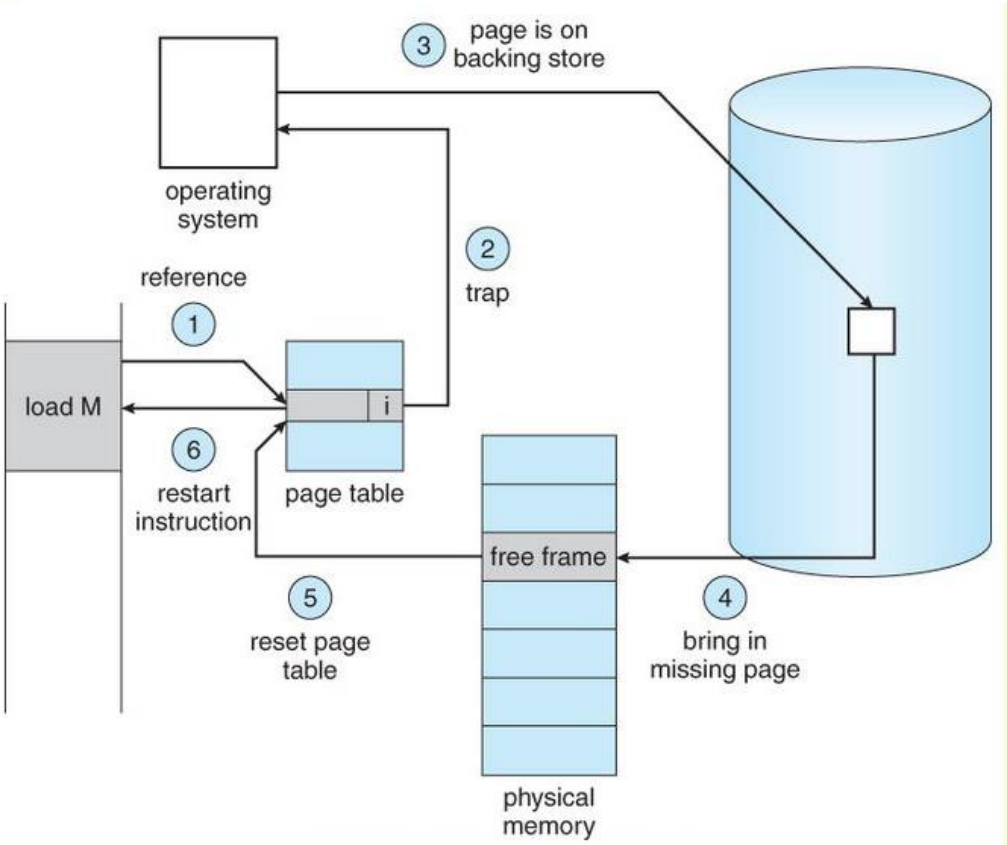
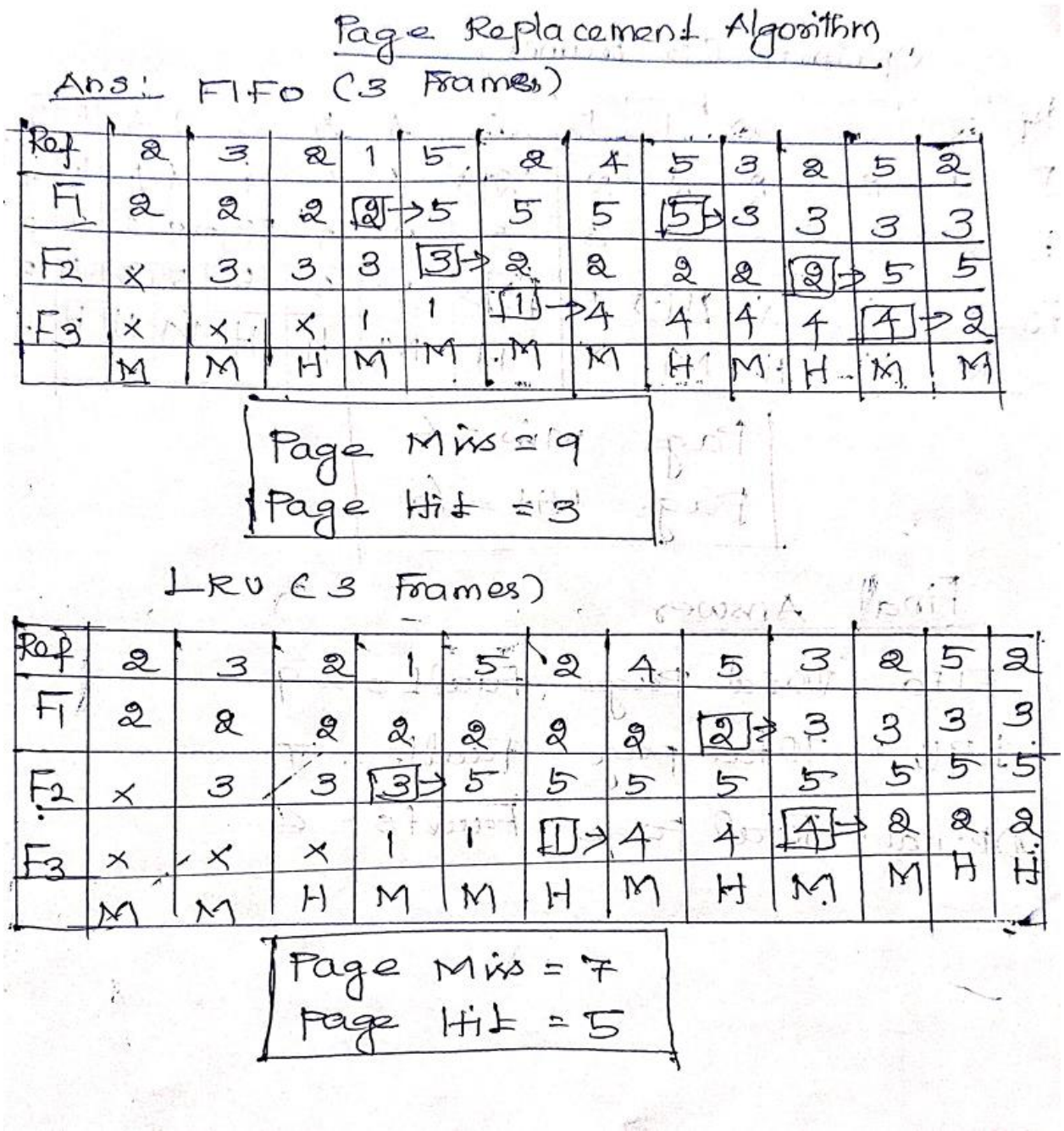


Figure 9.6 - Steps in handling a page fault

OR

Q8 b. For the following page reference string:

2 3 2 1 5 2 4 5 3 2 5 2 How many page faults would occur in FIFO ,LRU and optimal technique with frame number as 3



Page Replacement Algorithm

Ans: FIFO (3 Frames)

Ref	2	3	2	1	5	2	4	5	3	2	5	2
F1	2	2	2	[2]→5	5	5	[5]→3	3	3	3	3	
F2	x	3	3	3	[3]→2	2	2	2	[2]→5	5		
F3	x	x	x	1	1	[1]→4	4	4	4	[4]→2		
	M	M	H	M	M	M	M	H	M	H	M	M

Page Miss = 9
Page Hit = 3

LRU (3 Frames)

Ref	2	3	2	1	5	2	4	5	3	2	5	2
F1	2	2	2	2	2	2	2	[2]→3	3	3	3	
F2	x	3	3	[3]→5	5	5	5	5	5	5	5	
F3	x	x	x	1	1	[1]→4	4	4	[4]→2	2	2	
	M	M	H	M	M	H	M	H	M	M	H	H

Page Miss = 7
Page Hit = 5

Optimal (3 Frames)

	2	5	2	1	5	2	1	5	3	2	2	5	2
F1	2	2	2	2	2	[2]→	1	1	1	1	1	1	1
F2	x	5	5	5	5	5	3	3	[3]→	2	2	2	2
F3	x	x	x	[1]→	5	[5]→	5	5	5	5	5	5	5
	M	M	H	M	M	H	M	H	H	M	H	H	H

Page Miss = 6
Page Hit = 6

Final Answer

FIFO : Total Page Faults = 9

LRU : Total page Faults = 7

Optimal : Total Page Faults = 6

Module-5

Q9 a. Discuss on the file access methods in details

File access methods are techniques used by an OS to read and write data in files. They define how information is organized, retrieved, and modified. Choosing the right method is important for performance and data management.

There are three ways to access a file in a computer system:

- Sequential-Access
- Direct Access
- Index sequential Method

Sequential Access

A file access method where data is read or written **in order, one record after another**, starting from the beginning. The file pointer moves forward automatically after each operation.

Key Points related to Sequential Access

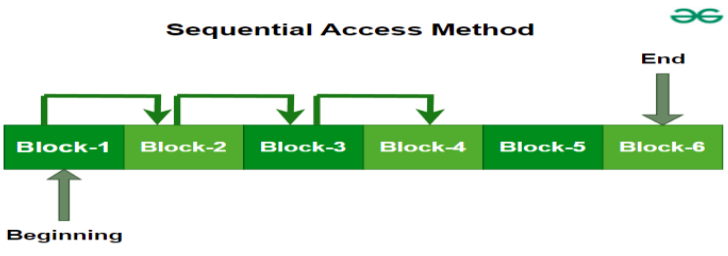
- Data is accessed from one record right after another record in an order.
- When we use the read command, it moves ahead pointer by one.
- When we use the write command, it will allocate memory and move the pointer to the end of the file.
- Such a method is reasonable for tape.

Advantages of Sequential Access Method

- It is simple to implement this file access mechanism.
- It uses lexicographic order to quickly access the next entry.
- It is less prone to data corruption as the data is written sequentially and not randomly.

Disadvantages of Sequential Access Method

- Slow for searching specific records.
- Inserting/updating in the middle is difficult.
- Can waste storage if records have varying lengths.



Direct Access Method

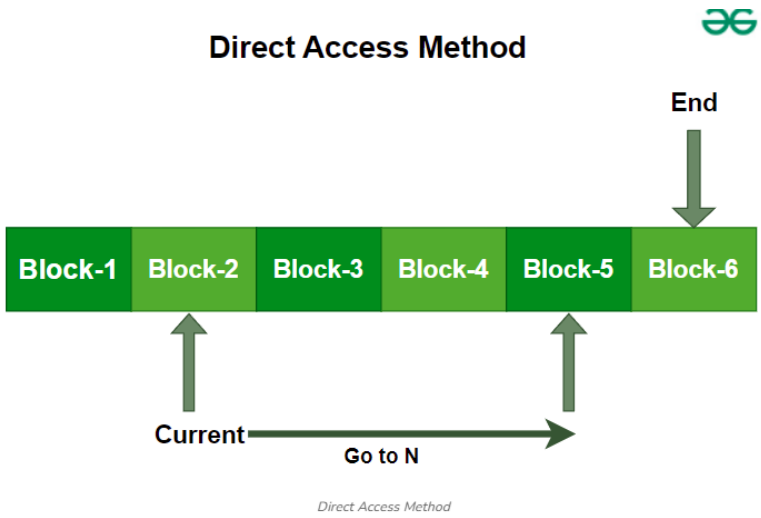
A file access method that allows data to be **read or written directly at any block or record**, using its address (block number). It supports **random access** without scanning previous records.

Advantages of Direct Access Method

- The files can be immediately accessed decreasing the average access time.
- In the direct access method, in order to access a block, there is no need of traversing all the blocks present before it.

Disadvantages of Direct Access Method

- **Complex Implementation** : Implementing direct access can be complex, requiring sophisticated algorithms and data structures to manage and locate records efficiently.
- **Higher Storage Overhead** : Direct access methods often require additional storage for maintaining data location information (such as pointers or address tables), which can increase the overall storage requirements.



Direct Access Method

Index Sequential method

It is the other method of accessing a file that is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index, and then by the help of pointer we access the file directly.

Key Points Related to Index Sequential Method

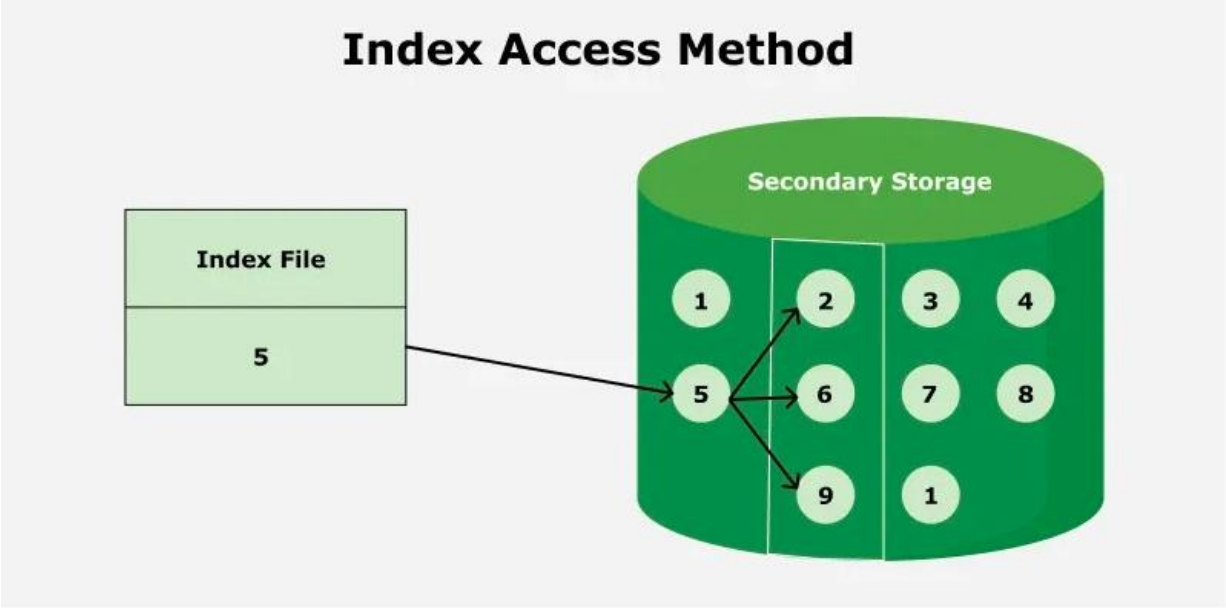
- It is built on top of Sequential access.
- It control the pointer by using index.

Advantages of Index Sequential Method

- **Fast Searching**: Index enables quick lookups.
- **Flexible:** Supports both sequential and random access.
- **Reduced Access Time:** Quickly locates data in large files.

Disadvantages of Index Sequential Method

- **Complexity:** Harder to implement and maintain than sequential access.
- **Extra Storage:** Requires additional space for indexes.
- **Slower Updates:** Both data and index must be updated during insertions, deletions, or modifications.
- **Maintenance Overhead:** Indexes need frequent updates in dynamic environments.



Index Access Method

Q9.b. Discuss the various directory structures.

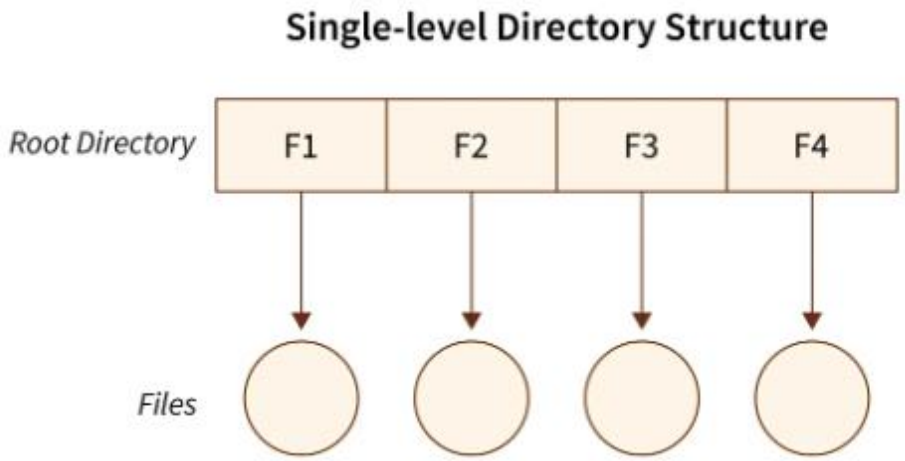
A directory is a container that stores files and folders, organizing them hierarchically. The Directory Structure in OS manages entries of files, including file names, locations, protection info, and more. This structure enables efficient file retrieval.

Single-Level Directory Structure

The single-level directory structure is the simplest and easiest directory structure out of all the other directories. In this directory structure, all the folders/files are contained under the same directory which is called the root directory. The single-level directory structure gathers all the files under one directory or the root directory, this makes it easy to support and understand.

Now as the different files are under the same-root directory the users are not allowed to create the different sub-directories serving their requirements. This also creates a barrier with the single-level directory as when the number of files increases or more than one user logs into the system both of these need to maintain the standards of giving a unique name to it. This also means that if two users call their files 'apple', then this, in turn, will violate the unique name standardization.

Below is the pictorial representation of The Single-level directory structure in OS :



Advantages

The implementation of a single-level directory structure is simple and easier as compared to other directory structures in OS.

If the file size is smaller, then the searching of such files with the single-level directory structure becomes simpler.

The single-level directory structure allows the operations such as searching, creation, deletion and updating as well.

Disadvantages

As several users can log in at the same system to log their files maintaining a unique name becomes difficult leading to a collision. This also means that if the file with the same name is created then the old file will get destroyed first, then the new file (having the same name) created will replace it.

If the size of the files is bigger than searching the files in one root directory of the single-level directory structure will become time-consuming and hence difficult.

The single-level directory structure restricts the grouping of the same type of files together.

Two-Level Directory Structure

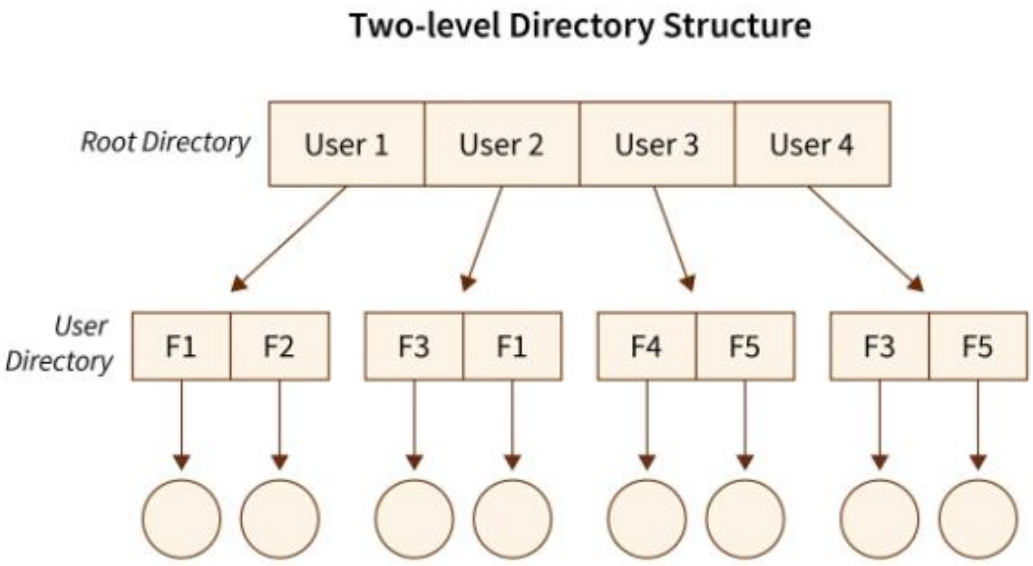
Overcoming the drawbacks posed by the single-level directory structure, i.e., the confusion created by the same file names given by several users - The Two-level directory structure in OS came into the picture.

The two-level directory structure in OS offers a unique solution to the problem caused by single-level that is, this directory structure gives each user the right to have their own user files directory commonly called User File Directory(UFD). The User File Directory or UFD has a similar structure as that of the single level, but each UFD lists only the files of a single user who owns that UFD. To root all the UFDs, the system's Master File Directory or (MFD) searches whenever a new user id's logged into the directory structure.

This can also be defined as the two-level directory structure in OS which gives each of its users the right to create a directory directly inside the root directory. Here the directories created by the user are called the UFDs and to check who logged in as a user the Master File Directory or MFDs are responsible for the same.

Here the MFDs are indexed by username or account number which are pointed to the UFDs with each entry point of the user. In a two-level directory structure searching files becomes, even more, easier as there is only one user's list, which is required to be traversed along with having a pathname for each file such as /User-name/directory-name/ which is also defined here.

Below is the pictorial representation of The Two-level directory structure in OS :



Advantages:

In the two-level directory structure in OS different users have the right to have the same directory as well as a file name as the user has its own USD which can give a filename that can match other users but won't cause an issue.

We can also see that searching for files become much simpler.

As we have a user-defined directory this also provides privacy related to files stored as no user can enter the other user's directory without permission.

In a two-level directory structure we cannot group the files which are having the same name into a single directory for a specific user.

Disadvantages:

In a two-level directory structure a user is not allowed to share files with other users.

We also find that scalability is not present in a two-level directory structure as two files of the same type cannot be grouped together in the same user.

Here users cannot create subdirectories only one user file directory can be defined under one master file directory.

Tree-Structured Directory Structure

As observed in the two-level directory structure in OS the drawback of users not having the ability to create sub-directories is resolved with The Tree-structured directory structure in OS coming into the picture.

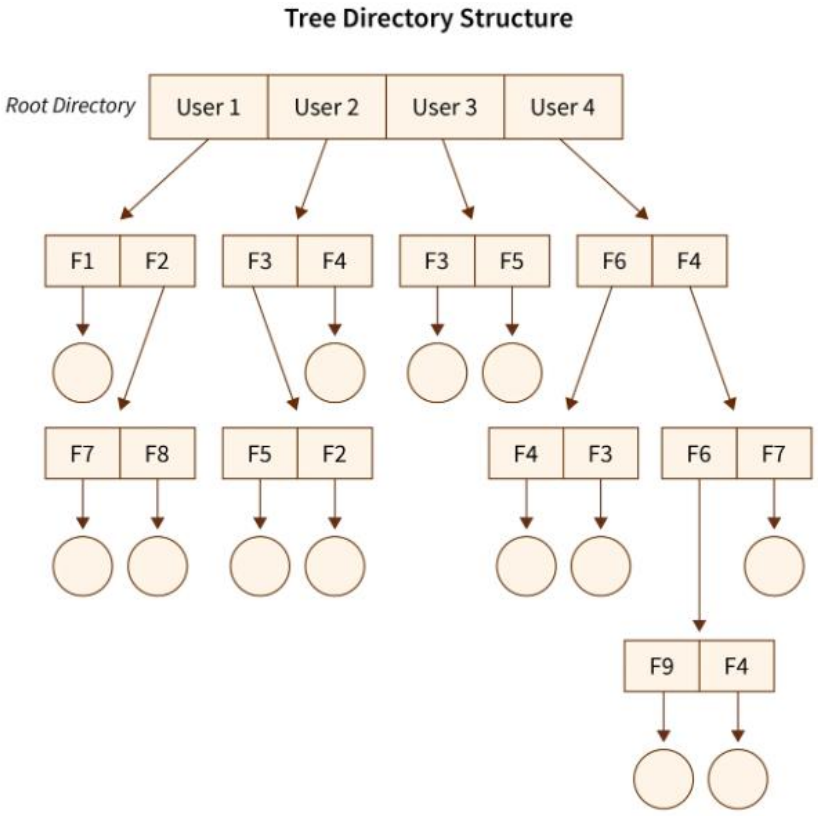
The Tree-Structured directory structure in OS is said to be the most common directory structure among users as it gives the users the capability to create sub-directories under their defined directory. Here we have the natural generalization which extends the directory structure to a tree of arbitrary height whereas, in the case of two-level, it was the tree of two heights. This generalization in tree structure allows the user the ability to create their own subdirectories and organize their files accordingly.

The tree-structured directory structure has separate parent directories for the sub-directories owned by each of their specific users and the parent directories of the users are all under the master-root directory which makes it a tree structure. This helps in total separation between the users which provides complete naming freedom and privacy to users' information.

The system administrator/ UFD admin only has full access to the root directory. In the Tree-structured directory structure searching is quite effective where we use the current **working**

concept that is we can access the file by using two kinds of paths that are either absolute or relative.

Below is the pictorial representation of The Tree-structured directory structure in OS :



Advantages:

In the Tree-structured directory structure searching is quite effective where we use the current working concept that is we can access the file by using two kinds of paths that are either absolute or relative.

Here we can group the same type of files into one directory.

In this directory structure the chances of collision of names/types etc are less and hence we can say that the directory structure in OS is scalable.

Disadvantages:

In the tree-structure directory structure in OS the files cannot be shared between users. Also, the users cannot modify/update the root directory of other users.

This directory structure in OS as we have to go under multiple directories to access a file we can say that it is said to be inefficient.

Here each file does not fit into the hierarchal model and so we have to save the files into multiple directories.

Acyclic Graph Directory Structure

Overcoming the drawbacks posed by the tree-structured directory structure,i.e, the restriction that it cannot have multiple parent directories and also cannot share files between users - The Acyclic Graph directory structure in OS came into the picture.

In the Acyclic Graph directory structure in OS can be defined as the directory structure that allows a directory or a file to have multiple parent directories so that it can be a shared file in a directory that gets pointed by the other user directories which if has the access to that shared file via the links provided to it. It is often said to be a natural generalization of the tree-structured directory.

Mostly, this is used in situations such as, when two users or two programmers are collaborating on a project and they need to access the files. So we have the associated files which are stored in a subdirectory mostly separated from other projects and files of other programmers/users. Now as they are working on a joint project they want the sub-directories to be present in their own directories. Therefore these common sub-directories where two or more users can collaborate would be shared so that the files can be stored in their individual locations and this is where we use Acyclic directories.

There must be a point noted here that the shared file is not the exact copy file, that is if any programmer/user makes some changes in the sub-directory that change will be reflected in both subdirectories. Here we can (with the help of aliases or links) create the acyclic type of directory graph where we can also provide different paths to the same file. We define the Links into two kinds, popularly known as The Hard Link and The Symbolic Link.

a.The Hard Link: This is also called as the physical link. If we want to delete the files in the acyclic graph directory structures then we need to remove the actual files only if all the references to the files are deleted that is, no link that even references the main file should be established. Here we don’t leave a suspended link.

b. The Symbolic/Soft-Link: This is also called as the logical link. If we want to delete the files in the acyclic graph directory structures then we need to simply delete the files and need to keep in mind that only a dangling/ hanging point is left. Here we leave a suspended link.

Advantages:

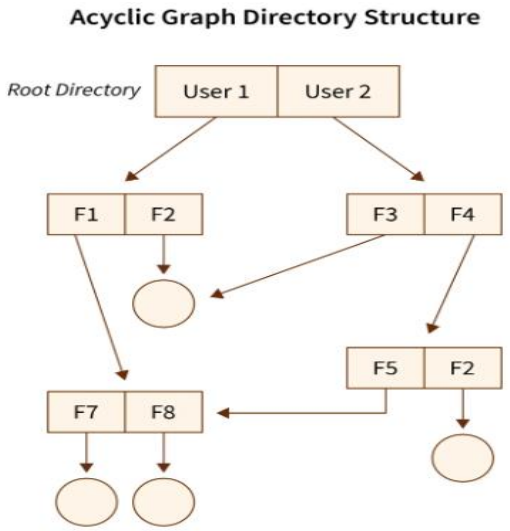
In the Acyclic Graph directory structure in OS we can share files between users.

Here we can search the files easily as compared to the tree-structured directory structure as here we have different-different paths to one file.

Disadvantages:

In the Acyclic Graph directory structure in OS as we can share the files via linking, so there are chances that in the case when we want to delete a file in a directory it may create a problem. a. Also, even if the link is a soft link then after deleting the file we are left with a dangling/suspended point of the link. b. But in the case of hardlink, when we delete a file we have to vanish all the references associated with it, which can lead to issues associated with referring back to files in case a requirement arises

Below is the pictorial representation of The Acyclic graph directory structure in OS :



General Graph Directory Structure

As observed in the acyclic-graph directory structure in OS the drawback of links that are established and need to be either terminated or suspended to reach the files/directory is resolved with The General Graph directory structure in OS taking a vital place in the different types of a directory structure in OS.

In the General Graph directory structure in OS users have the capability to create a cycle of the directory within a directory where we have the power to derive the various directories with the help of more than one parent directory in operating systems. In this type of structure, the users are free to create directories under the root directory along with creating sub-directories under

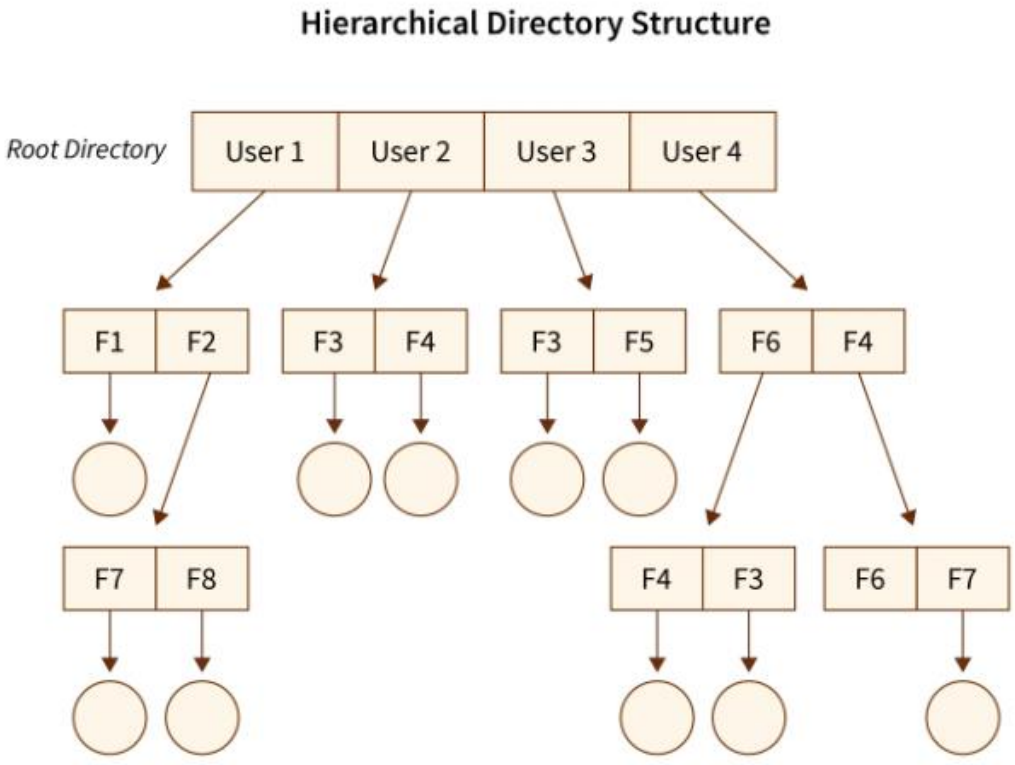
the same structure which can also hold true if the users want to create multiple sub-directories that is, the users are free to create different sub-directories for different file types.

Adding to the above, With the help of the file paths if the users feel the need to access the location of the files then that is made possible by the General Graph Directory Structure. In this structure, the file paths or paths can be categorized into two broad categories to locate the files in the directory structure as below :

The Absolute Path: Here, the path of the desired files can be determined by considering the root directory as the base directory.

The Relative Path: Here, the path of the desired files can be determined by two choices that are, either the file that needs to be retrieved from its directory is considered the base directory, or the user's directory is considered as the base directory.

Below is the pictorial representation of The General Graph directory structure in OS :



Advantages:

The General Graph directory structure allows the cycle or creation of a directory within a directory.

This directory structure is known to be a flexible version compared to other directory structures.

Disadvantages:

The main issue that can overpower this structure is to calculate the total size or space that the directories will take up.

As this directory structure allows the creation of multiple sub-directories a lot of garbage collection can be required.

If compared to the other directory structure in OS the General Graph directory structure is a costly structure to be chosen.

OR

Q10.a. Define File. Explain the various operations performed on a file.

A file is a collection of logically related data that is recorded on the secondary storage in the form of sequence of operations. The content of the files are defined by its creator who is creating the file. The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system. Some common operations are as follows:

1. Create operation:

This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. Open operation:

This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. Write operation:

This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. Read operation:

This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. Re-position or Seek operation:

The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

6. Delete operation:

Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

7. Truncate operation:

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8. Close operation:

When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

9. Append operation:

This operation adds data to the end of the file.

10. Rename operation:

This operation is used to rename the existing file.

Q10.b. Discuss three secondary storage allocation methods.

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide efficient disk space utilization & Fast access to the file blocks.

Contiguous Allocation

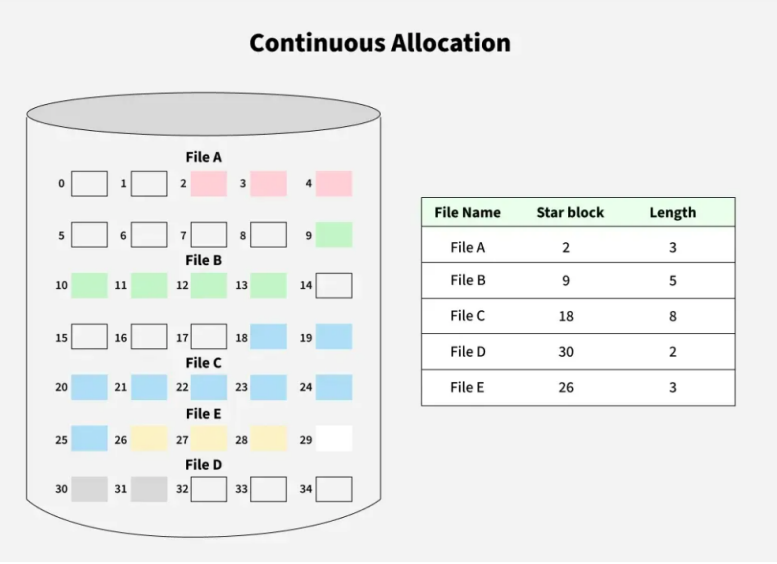
In this scheme, each file occupies a contiguous set of blocks on the disk. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains:

Address of starting block

Length of the allocated portion.

Example: If a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be:

b, b+1, b+2,.....b+n-1.



Contiguous Allocation

Advantages

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k)(*b*+*k*).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

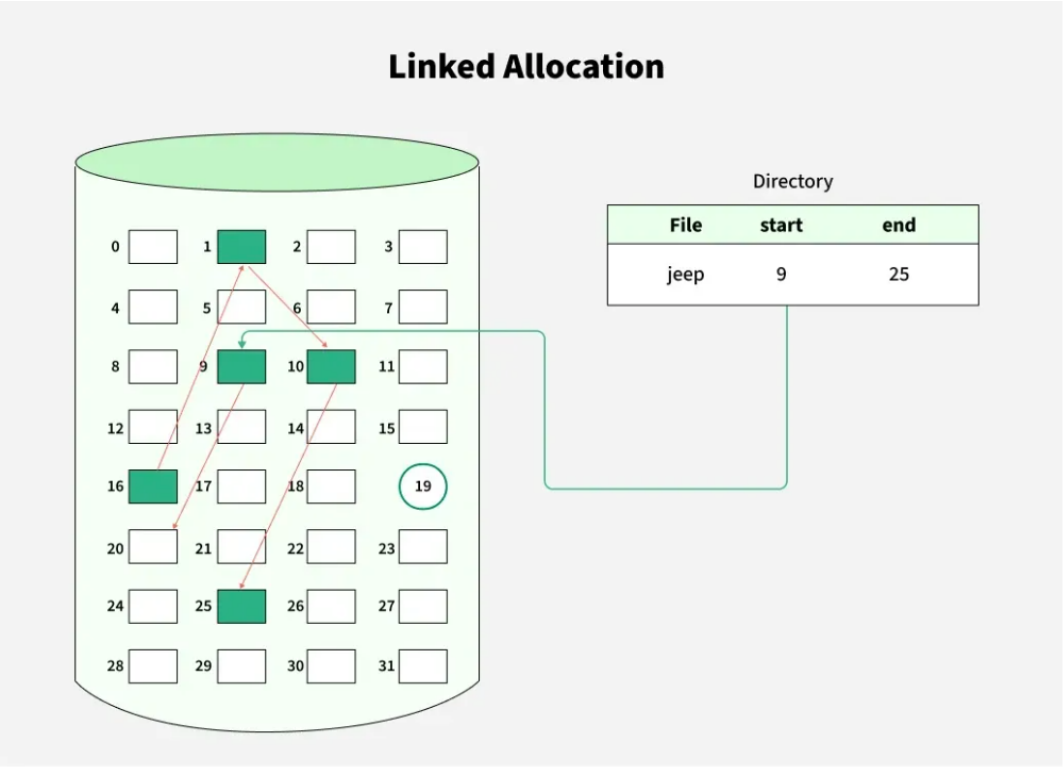
Disadvantages

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked Allocation

In this scheme, each file is a linked list of disk blocks which need not be contiguous. Here,

- The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block.
- Each block contains a pointer to the next block occupied by the file.
- **Example:** The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Linked Allocation

Advantages

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

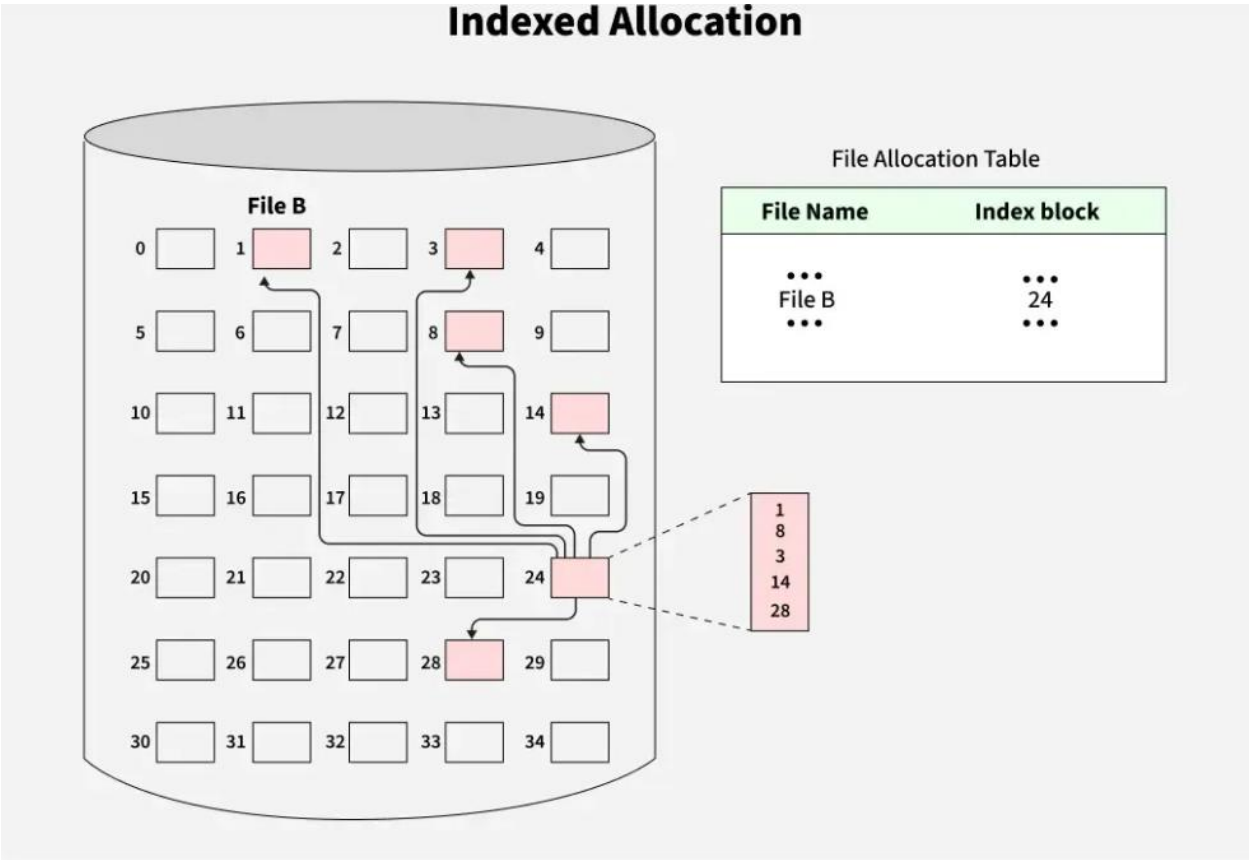
Disadvantages

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Here,

- Each file has its own index block.
- The i^{th} entry in the index block contains the disk address of the i^{th} file block.
- The directory entry contains the address of the index block as shown in the image.



Indexed Allocation

Advantages

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.