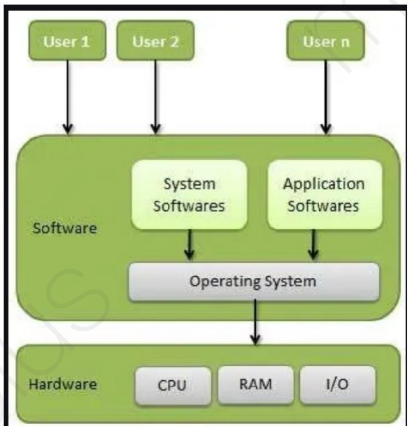
What is an Operating System?

- An operating system is the main software that runs on your computer. Without it, your computer wouldn't know how to perform tasks or communicate with its parts
- Interface between user and hardware
- control program
- Act like a government and control program

Think of it like a manager that:

1. Tells hardware what to do (keyboard, screen, memory, etc.)
2 Runs your programs (games, apps, browser)
3. Keeps things safe and organized



Services of OS :-

- User Interface
- Program Execution
- I/O Operation
- File-System Manipulation
- Error Detection
- Resource Allocation

QUE:-1 Define Operating system and mention its major functions ? (AKTU-21-22,22-23,23-24)

ANS :- Main Functions of an Operating System
Here are the main jobs (functions) in simple words:

1.Memory Management
Keeps track of which app uses how much memory (RAM).
Makes sure one program doesn't mess up another.

2.Process Management
Starts, runs, and stops programs (processes).
Shares the CPU (brain of computer) between many programs so they all get time to work.

3.File Management
Helps you create, save, delete, and organize files and folders.
Controls who can see or change files.

4.Device Management
Connects and controls hardware (keyboard, mouse, printer, etc.).
Makes sure all devices talk to the computer correctly.

5.User Interface
Shows you menus, icons, and windows you can click.
Allows you to give commands (typing or tapping).

6.Security and Access Control
Keeps your data safe with passwords and permissions.
Blocks viruses or unauthorized users.

QUE:-2 Explain in detail about the Operating System services.(AKTU 21-22)

ANS :-
1. Program Execution
—> What it does:
The OS helps start (execute) and stop programs.

It loads your app into memory and runs it.
Example: When you open a game, the OS makes it start properly.

2.I/O Operations
—> What it does:
Handles all Input/Output operations (reading and writing data).

Helps programs use devices like keyboard, mouse, and disk.
Example: When you save a file, the OS writes data to the disk.

3.File System Manipulation
—> What it does:
Helps create, delete, read, write, and organize files and folders.

Manages permissions to keep files safe.
Example: When you make a new folder, the OS organizes it for you.

4.Communication Services
—> What it does:
Allows programs to talk to each other (called Inter-Process Communication).

Supports data sharing between processes.
Example: A chat app using messages to communicate between server and client.

5.Error Detection
—> What it does:
Checks for errors in hardware and software.

Tries to fix problems or inform the user.
Example: If your disk is full, the OS shows an error message.

6.Resource Allocation
—> What it does:

Distributes computer resources (CPU time, memory, devices) among programs.
Example: While you watch a video and browse the web, the OS gives time to both tasks.

7.Security and Protection
—> What it does:
Keeps your data and programs safe from unauthorized access.

Manages passwords and permissions.
Example: You need a password to log into your computer.

Summary

Operating System services are like helpers that:

- Start and manage programs
- Handle input/output
- Organize files
- Enable communication
- Detect errors
- Share resources
- Keep everything safe

What is a Batch Processing Operating System?

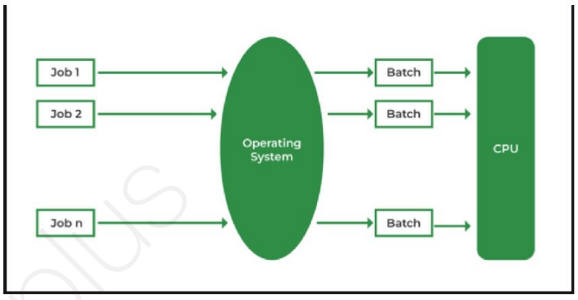
Batch Processing Operating System (BPOS) is a type of system that:
. Groups jobs together in batches (like putting similar work in one big folder).
. Runs them one by one automatically, without needing people to sit and watch.

—> How It Worked
. In the old days (1970s):

Users could NOT work directly on the computer.

- They prepared their work on special cards called punch cards.
- These cards had holes in them to store data.
- They gave the cards to a computer operator.
- The operator sorted all jobs with similar needs together into batches.
- For example, all printing jobs in one batch, all calculation jobs in another.
- The computer ran each batch automatically.

—> Why Was It Popular?
. Fast for big amounts of data.
. No need for people to keep giving commands.
. Good for businesses that process lots of records, like:



Time Sharing Operating System

What it does:

It allows many programs to run at the same time by quickly switching between them.

This switching happens so fast that each user feels like only their program is running, even though many users are sharing the system.

—> How it works:

The system gives each program a small amount of time to run (this is called a time slice).
When one time slice ends, the system switches to the next program, and so on.

—>Why it's good:
All users feel like they have the computer to themselves.
Perfect for multi-user environments like shared servers or labs.

Real-Time Operating System

What it does:

In this system, tasks must be done within a fixed time limit (deadline).
The system not only needs to give correct answers but must also give them on time.

—>Why timing is important:

If the task is late, the system fails, even if the answer is right.

—>Where it is used:

In places where timing is critical, like:
Airbag systems in cars (must work instantly in an accident).
Medical devices (must monitor patients in real-time).
Industrial robots (must react without delay).

What is a Multiprocessor System?

A multiprocessor system is a computer that uses two or more CPUs (processors) to work together.
These processors share the same memory and work on tasks at the same time.

.This means the computer can do more work faster by dividing tasks between processors.

—>Example:
. Think of it like a kitchen with multiple chefs:
Instead of one chef making all the food,
Each chef handles a different dish at the same time,
So the food is ready much faster!

What is a Multiuser System?
.A multiuser system is a computer system that many people can use at the same time.

—> Example:
Think of a library computer:
Many students can log in with their own accounts.

Each person can do their work independently, like typing, browsing, or printing.

QUE :- 3 Compare and contrast time-sharing and multiprogramming operating system concepts? (AKTU-23-24)

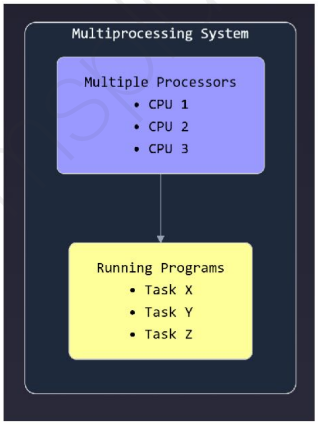
ANS:-

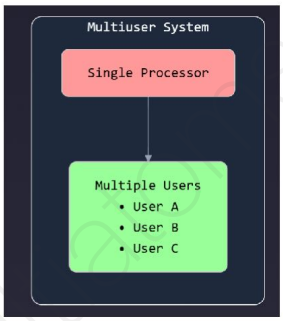
Feature	Multiprogramming	Time-Sharing
Users	Usually one user at a time	Many users at the same time
Goal	Keep CPU busy	Make all users feel they have the computer
Interaction	No or very little user interaction	Users interact directly (typing, clicking)
Time Slice	No fixed time slice	Each user gets a time slice of CPU time
Example	Batch job processing	Multiple users logged into a server

QUE:-4 How does a multiuser operating system differ from a multiprocessing system ? (AKTU-23-24)

ANS:-

- Multiuser System: Like a shared office where many people work together using the same computer resources
- Multiprocessing System: Like having multiple workers (processors) doing different tasks simultaneously



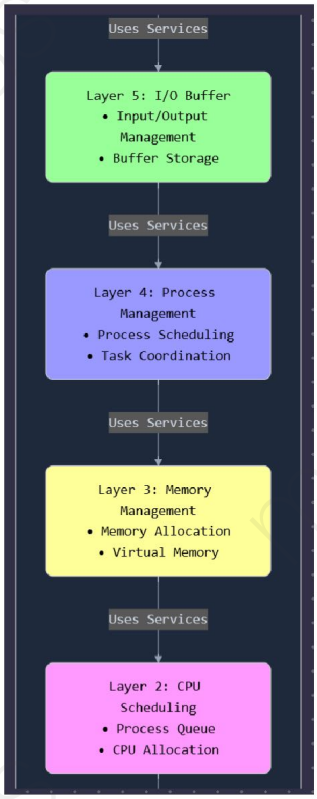


FEATURE	MULTIUSER SYSTEM	MULTIPROCESSING SYSTEM
Hardware	Single processor	Multiple processors
Users	Many users share resources	Usually single user
Work Style	Taking turns using resources	Working simultaneously
Speed	Slower for individual tasks	Much faster overall
Best For	Shared office environments	Heavy calculations and gaming

QUE:-5 Explain the structure and components of an operating system with a layered architecture? (AKTU-23-24)

ANS :-**What is Layered Structure OS?**

- It is an operating system that is divided into layers.
- Each layer does a special job.
- These layers are arranged one over the other, like steps.



Key Components and Their Functions

- **Hardware Layer (Layer 1)**

Bottom-most layer that controls physical devices
Manages basic computer operations
Handles keyboard, mouse, and other hardware components

- **CPU Scheduling Layer (Layer 2)**

Manages CPU time allocation
Controls process execution order
Handles process queues and CPU allocation

- **Memory Management Layer (Layer 3)**

Controls memory allocation
Manages both RAM and ROM
Handles virtual memory operations

- **Process Management Layer (Layer 4)**

Manages multiple processes

Controls process scheduling
Handles process coordination and communication

- **I/O Buffer Layer (Layer 5)**

Manages input/output operations
Provides temporary data storage
Handles device communication

- **User Programs Layer (Layer 6)**

Top-most layer for user interaction
Manages application programs
Provides user interface

1. **L1** – Hardware
2. **L2** – CPU Scheduling (deciding which process gets the CPU)
3. **L3** – Memory Management (managing RAM)
4. **L4** – Process Management (handling running programs)
5. **L5** – I/O Management (managing devices like keyboard, mouse, disk)
6. **L6** – User Interface (screen and interaction with the user)

Advantages :-

1. Easy Debugging:
If there's a problem, you can check layer by layer to find and fix it easily.

2.Easy Update:
You can change one layer without affecting others, making updates simpler.

3.No Direct Access to Hardware:
Programs cannot directly touch hardware, which keeps the system safer.

Disadvantages :

1. Complex:
Designing all the layers can be complicated.

2.Slower Execution:
Because everything must pass through layers, it can be a bit slower.

3. Limited Communication:
Layers cannot directly talk to non-adjacent layers, which can make some things harder

A Layered OS is an operating system divided into layers, each doing a different job.

Bottom layer: Controls hardware.
Top layer: Interacts with the user.
Middle layers: Manage CPU, memory, processes, and devices.

This makes the OS organized and easy to manage, but a little slow and complex.

System Components

- **Process Management**

Creates, executes, and terminates programs
Manages process scheduling
Handles process communication

- **Memory Management**

Allocates and deallocates memory
Manages virtual memory
Prevents memory conflicts

- **File Management**

Organizes and stores files
Provides access control

Manages file operations

- **I/O Management**

Controls input/output operations
Manages device drivers
Handles data transfer

Reentrant Kernel

The Kernel is the main part of the Operating System.

It controls everything your computer does.

What does Reentrant mean?

- Reentrant simply means:

The same piece of code can be used by many processes at the same time, without causing problems.

A reentrant kernel works the same way:

- Multiple programs can use the same kernel code simultaneously
- Each program gets its own "space" to work in
- The kernel code itself doesn't change
- Each program thinks it has the kernel all to itself

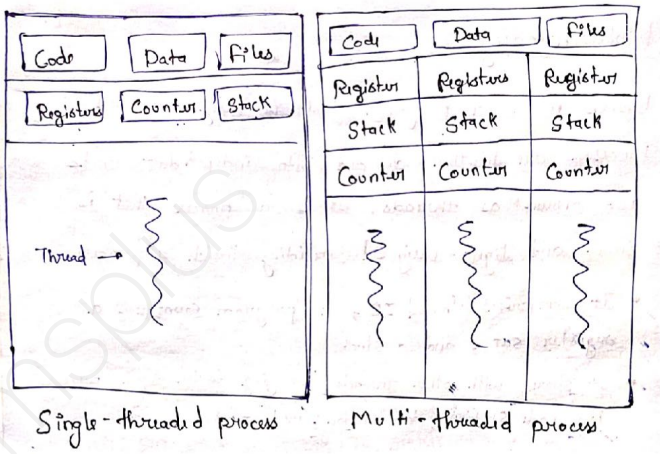
QUE :- 6 Explain in detail about the Multiuser Systems and Multithreaded Systems. (AKTU-23-24)

ANS :- Multithreaded OS

- Single program splits into multiple threads
- Threads share the same memory space
- Threads can work together easily
- Faster communication between parts

Process means a program that is being executed processes are further divided into independent units also known as threads .It is a process that is small and light weighted residing inside a process .

It comprises a thread ID , a program counter , a register set and a stack.



Benefits of Multithreaded Systems
Faster execution
Efficient resource use
Better application performance
Improved responsiveness

Challenges of Multithreaded Systems
Complex to program
No memory protection
Synchronization issues
Potential data conflict

Monolithic and Microkernel Systems.

1. Monolithic Kernel

—>What is it?
It is a type of kernel where everything related to the operating system is put together in one big block (module) inside the kernel space.

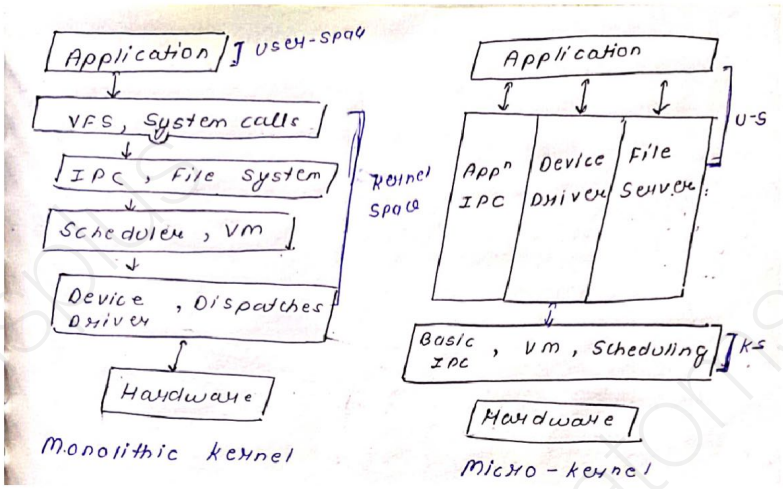
—> Main Points:
All OS parts are together in the same place.
If one service crashes, the whole OS crashes.
It is big in size but fast to run.
It is hard to debug (find and fix errors).
If you want to add something new, you must change the entire OS.
Security issues are more common because all services are mixed (no separation).
Examples: Linux, DOS, Unix.

2. Microkernel

—> What is it?
A microkernel is different because it keeps only the most important parts inside the kernel space and moves other parts out into user space.

—>Main Points:
Non-essential components (like device drivers, file systems) are moved outside the kernel.
Only essential functions stay inside the kernel:
Communication between processes

Memory management
Scheduling
If one service fails, the rest of the OS still works fine.
It is smaller in size but slower to execute.
Adding new services is easy because they are separate.
Debugging is easier.
Examples: macOS, Windows NT, QNX.



Feature	Monolithic Kernel	Microkernel
Structure	All OS services are together in **one big block**	Only essential parts in the kernel; others kept outside
Size	**Large** in size	**Small** in size
Speed	**Fast execution** because everything is in one place	**Slower execution** because parts communicate separately
Stability	If **one service fails, entire OS can crash**	If one service fails, **rest of the OS keeps working**
Adding New Features	**Hard**, you need to modify the whole kernel	**Easy**, you can add or update services separately
Debugging	**Difficult** to debug (fix problems)	**Easier** to debug
Security	**Less secure** (less isolation bet[illegible]n parts)	**More secure** (services separated in user space)

UNIT-2

Process Concept

What is a Process?
A process is a program in execution.

Process = Program + Running State

A process includes:
The code of the program
The data it uses

The current activity (like what it's doing right now)
The memory it's using
Information like open files, network, etc.

QUE:-1 What do you mean by Concurrent Processes? (AKTU-21-22 & 23-24)

ANS:- Principle of Concurrency

Concurrency means many tasks happening at the same time.

Multiple processes (or programs) are running together and sharing resources like CPU and memory.

—>Problems concurrency solves:
Keeps the CPU busy
Makes sure slow tasks (like waiting for input) don't block everything
Allows multiple users or processes to share the system

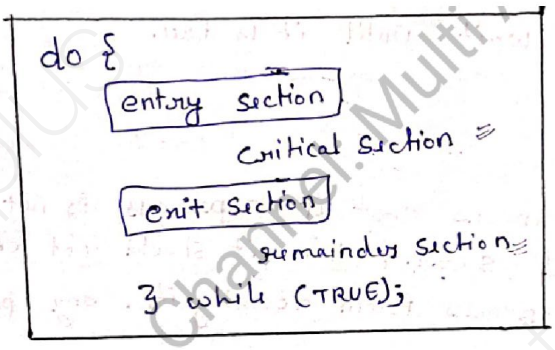
—> Issues with concurrency:
Concurrency also brings challenges:

Race conditions: Two processes trying to change the same data at the same time
Deadlock: Processes waiting forever for each other
Starvation: One process never gets CPU time

Critical Section (AKTU-21-22)

The critical section problem is used to design a protocol followed by a group of processes, so that when one process has entered its critical section, no other process is allowed to execute in its critical section.

The critical section refers to the segment of code where processes access shared resources, such as common variables and files, and perform write operations on them.



Solutions to the critical section problem

Any solution to the critical section problem must satisfy the following requirements:

Mutual exclusion: When one process is executing in its critical section, no other process is allowed to execute in its critical section.

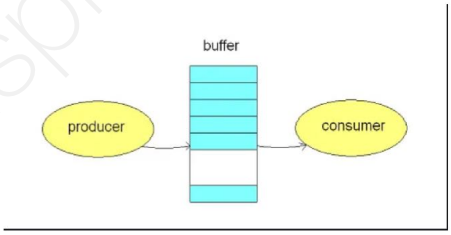
Progress: When no process is executing in its critical section, and there exists a process that wishes to enter its critical section, it should not have to wait indefinitely to enter it.

Bounded waiting: There must be a bound on the number of times a process is allowed to execute in its critical section, after another process has requested to enter its critical section and before that request is accepted.

Producer / Consumer Problem (AKTU-22-23)

Producer: Makes data and puts it into a buffer (storage).
Consumer: Takes data out of the buffer to use it.

Problems to avoid:
. **Overflow:** Producer puts data when the buffer is full.
. **Underflow:** Consumer tries to take data when the buffer is empty.
. **Race conditions:** Producer and consumer both trying to change the buffer at the same time.



What is a Semaphore?
A semaphore is like a special counter with rules that helps processes wait or continue safely.

There are two main types:
Counting Semaphore: Counts how many resources are available.
Binary Semaphore (Mutex): Acts like a lock (only 0 or 1).

Solution Using Semaphores
We need 3 semaphores:

1. Mutex (binary semaphore):
Controls mutual exclusion (only one process can access buffer at a time).
Starts with value 1 (unlocked).

2. Empty (counting semaphore):
Counts how many empty slots are in the buffer.
Starts with N (if buffer size = N).

3. Full (counting semaphore):
Counts how many items are in the buffer.
Starts with 0 (buffer is empty at the beginning).

—>Steps for Producer:
1.Wait(empty)
(If there is no empty space, Producer waits.)

2.Wait(mutex)
(Lock the buffer so no one else can use it.)

3.Add item to buffer
(Put data in.)

4.Signal(mutex)
(Unlock the buffer.)

5.Signal(full)
(One more item is now available.)

—> Steps for Consumer:
1. Wait(full)

(If there is nothing to consume, wait.)

2. Wait(mutex)
(Lock the buffer.)

3. Remove item from buffer
(Get data.)

4. Signal(mutex)
(Unlock the buffer.)

5. Signal(empty)
(One more empty slot is available.)

Dekker's Solution

What is it?
Dekker's solution is a software way to make sure that two processes don't enter their critical section at the same time.

It was the first known algorithm to solve the critical section problem for two processes.

—>How it works?
Dekker's solution uses:
1. A flag array to show if a process wants to enter the critical section:

flag[0] for Process 0
flag[1] for Process 1

2.A turn variable to decide whose turn it is to enter.

—> Steps:

1.A process sets its flag to true (meaning "I want to enter").

2.If the other process also wants to enter (other flag is true):
Check whose turn it is.
If it's the other process's turn, wait (don't enter).
Otherwise, proceed.

3.When exiting the critical section:
Change turn to the other process.
Set your flag to false (meaning "I'm done").

—>Important:
It ensures mutual exclusion (only one process inside).
It avoids deadlock and starvation.

Peterson's Solution

What is it?

- Another software solution for two processes to avoid both entering the critical section together.
- Simpler and easier to understand than Dekker's.

—>How it works?

Peterson's solution also uses:
1.A flag array to show intent (wanting to enter).
2. A turn variable to decide whose turn it is.

Steps (for Process i):

Set flag[i] = true (I want to enter).

Set turn = j (let the other process go first).

While (flag[j] == true AND turn == j) :

Wait (busy wait).

Enter critical section.

When done:

Set flag[i] = false.

—>Why this works:
Mutual Exclusion: Only one process can enter.
Progress: If only one wants to enter, it goes without waiting.
Bounded Waiting: No starvation.

—>Advantages over Dekker's:
Easier to understand.
Shorter code.
Still only for two processes.

QUE:-2 Compare and contrast Dekker's and Peterson's algorithms for mutual exclusion, considering their advantages and limitation? (AKTU-23-24)

ANS:-
Dekker's Algorithm

—>Advantages:
First algorithm to solve the critical section problem in software
Guarantees mutual exclusion and bounded waiting

—>Limitations:
Complex logic with many steps
Harder to understand and maintain

Peterson's Algorithm
—>Advantages:

Simpler and more elegant
Easier to implement and understand

—> Limitations:
Still only for 2 processes
Relies on memory visibility—some modern processors may need memory fences to work reliably

Simple Words:

- Dekker's Algorithm is like the first draft: it solves the problem but in a complicated way.
- Peterson's Algorithm is a cleaner, improved version: it solves the same problem with k

Aspect	Dekker's Algorithm	Peterson's Algorithm
Number of Processes	2 processes only	2 processes only
Type	Software solution	Software solution
Main Idea	Uses flags + turn, with extra checks to resolve conflicts	Uses flags + turn with simpler logic
Complexity	More complicated logic	Simpler and clearer logic
Fairness (Bounded Waiting)	Yes (no starvation)	Yes (no starvation)
Mutual Exclusion	Ensured (only one enters)	Ensured (only one enters)
Progress	Guaranteed	Guaranteed
Ease of Understanding	Harder to understand and implement	Easier to understand and implement
Performance	Slightly more overhead due to more checks	More efficient and clean
Hardware Needs	No special hardware needed	No special hardware needed
Limitations	Only for 2 processes; tricky to extend; complex	Only for 2 processes; still not ideal for modern multiprocessor systems

What is Test and Set?
Test and Set is a special hardware instruction used to achieve mutual exclusion (make sure only one process enters the critical section at a time).

It is often used to create locks.

—>What does it do?
Test and Set does two things together in one step:

1.Test: Check the value of a variable (usually a lock variable).
2. Set: Change the variable to a new value.

This happens atomically, meaning no other process can interrupt it while it's happening.

QUE-3 Explain in detail about the Dining Philosopher Problem (AKTU-21-22)

ANS:- **Problem Statement**

Dining Philosopher Problem
Imagine K philosophers sitting around a round table.
Between each philosopher, there is 1 chopstick.
To eat, a philosopher needs both chopsticks next to him (left and right).
One chopstick can be used by only one philosopher at a time

Example:
If you have 5 philosophers:

Philosopher 1 needs chopstick 1 and chopstick 2.
Philosopher 2 needs chopstick 2 and chopstick 3.
...and so on.

Semaphore Solution

- Semaphore is a variable that helps control access to resources (here, chopsticks).
- The solution uses semaphores to avoid two philosophers picking the same chopstick at the same time.

Steps:

Thinking
The philosopher is not hungry.
He is just thinking.

Picking up chopsticks
The philosopher waits (using semaphore) to pick:
The left chopstick
The right chopstick

wait(chopstick[i]);
wait(chopstick[(i+1) mod 5]);

Eating

After getting both chopsticks, the philosopher eats.

Putting down chopsticks

Once finished, the philosopher releases the chopsticks:

signal(chopstick[i]);
signal(chopstick[(i+1) mod 5]);

—>Three States of a Philosopher
THINKING – Not hungry, just thinking.

HUNGRY – Wants to eat, trying to pick up chopsticks.

EATING – Has both chopsticks and is eating.
Semaphores Used
There are two types of semaphores here:

1. Mutex Semaphore:
Makes sure only one philosopher at a time can pick up or put down chopsticks.
This prevents conflicts.

2.Semaphore Array for Chopsticks:
Controls the availability of each chopstick.

Sleeping Barber Problem

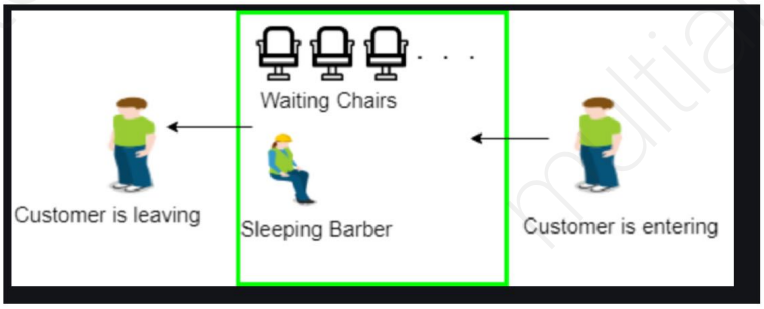
What is the Sleeping Barber Problem?
It's a fun way to understand how to handle multiple people (processes) sharing a limited number of resources (like chairs), in computer systems.

The Story:

- There is one barber.
- He has one barber chair (for cutting hair).
- There are some chairs in the waiting area.
- Customers come randomly to get a haircut.

What Happens in the Shop:

- If no customers are in the shop, the barber goes to sleep.
- When a customer comes:
- If the barber is sleeping, the customer wakes him up.
- If the barber is cutting someone's hair:
- The new customer looks for a free chair in the waiting room.
- If a chair is free, the customer waits.
- If all chairs are full, the customer leaves.
- When the barber finishes, he checks the waiting room:
- If a customer is waiting, he calls the next one.
- If no one is waiting, he goes back to sleep.



Why This Problem Matters (in OS):
This helps explain how programs (like customers) use a shared resource (like a printer or CPU) without crashing into each other or getting stuck.

Tools We Use: Semaphores
Semaphores are like counters with signals that help manage who gets to do what and when.

We use three semaphores in this problem:

—>Customers Semaphore
Counts how many customers are waiting.
If it's 0, the barber sleeps.

—>Barber Semaphore
Shows whether the barber is ready.
Used to wake the barber up.

—>Mutex Semaphore (Lock)
Makes sure only one person can check or change the chairs at a time.

What Do Customers Do?
When a customer comes in:
Lock the shop (mutex) so nobody else changes chair count.

Check if there's an empty chair.
If yes, sit and wait.
If no, leave.
If sitting, increase customer count (signal barber).
Unlock the shop (release mutex).
Wait until barber calls you for haircut.

What Does the Barber Do?
When the barber starts:

Sleeps (waits) until a customer shows up.

When a customer comes, barber wakes up.

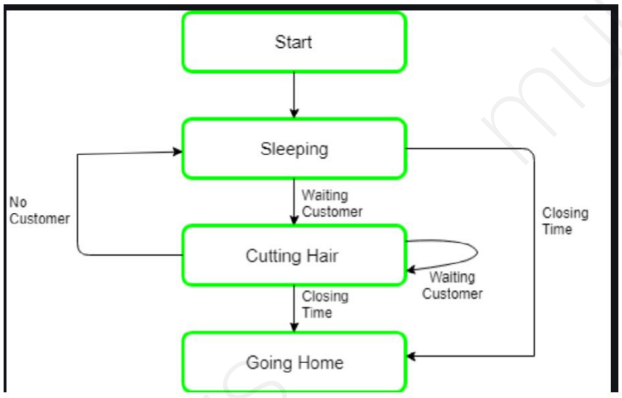
Locks the shop to call a customer in.

Cuts hair.

Finishes and checks: Any more customers?

If yes, go to step 3 again.

If no, go back to sleep.



—>What Does This Solve?
No two customers try to take the same chair.

The barber only works when there are customers.

If the shop is full, new customers leave peacefully (no chaos).

No one gets stuck or waits forever (no deadlock or starvation).

Inter Process Communication (IPC) (AKTU 21-22)

Processes need to communicate with each other in many situations. Inter-Process Communication or IPC is a mechanism that allows processes to communicate. It helps processes synchronize their activities, share information, and avoid conflicts while accessing shared resources.

Types of Process
Let us first talk about types of processes:

Independent process: An independent process is not affected by the execution of other processes. Independent processes do not share any data or resources with other processes. No inter-process communication is required in this case.

Co-operating process: Interact with each other and share data or resources. A co-operating process can be affected by other executing processes. Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of cooperation between them.

What is Inter Process Communication (IPC)?

Inter Process Communication (IPC) means:
Different programs (processes) talking to each other and sharing data.

Why do we need it?

So processes can work together.
So they can share information without messing each other up.

—>Two Main Models of IPC
There are two basic ways processes can communicate:

1. Shared Memory
Processes share some part of memory.
They can read or write data there.

Who controls it?
The programmer must write the code to handle this sharing.

Uses:
Faster sharing of big data.

2. Message Passing
Processes send messages to each other.

Example:

Like sending a text message.
One process sends().
The other process receives().

Uses:
Works well even if processes are on different machines.

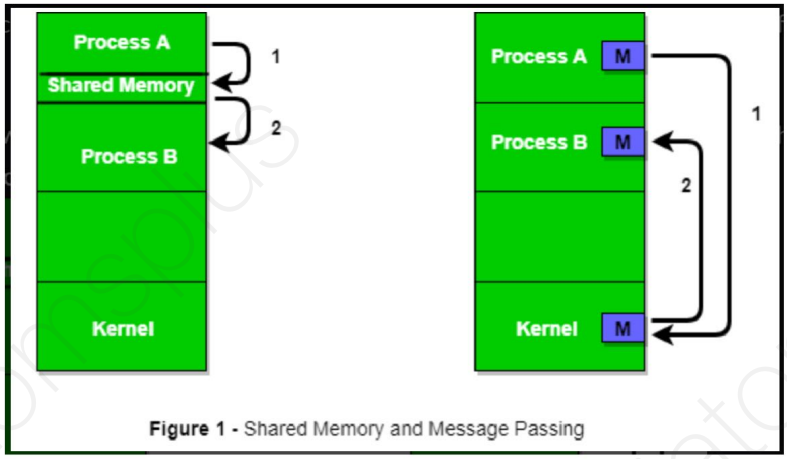


Figure 1 - Shared Memory and Message Passing

Process generation

What is a Process?
A process is simply a program that is running.

Example:
When you open a web browser – that's a process.
When you run a game – that's another process.

What is Process Generation?
Process Generation means creating new processes in the operating system.

You can think of it like having a parent process giving birth to child processes.

Why Do We Create New Processes?
New processes are created for many reasons:

- To run a new program.
- To handle tasks separately (like printing while typing).
- To do work in parallel (multitasking).

QUE:- 4 Explain the challenges of achieving mutual exclusion in concurrent programming? (AKTU23-24)

ANS :-
What is Mutual Exclusion?
Mutual exclusion simply means:Only one process or thread at a time can enter the critical section (the place where shared data is changed).

Why Is It Difficult?
Here are the main challenges, explained simply:

1. Race Conditions

Problem:

If two processes run at the same time, their actions can mix eup unpredictably.
This can cause wrong or inconsistent data.

2. Timing is Unpredictable

Problem:

You can't control when processes run.
CPU switches between processes at any time.
So it's hard to know which process is using the resource right now.

3. Deadlock

Problem:

Two or more processes wait forever, each waiting for the other to release a resource.
Nothing moves.

4. Starvation

Problem:

A process waits too long because others keep taking the resource.
It never gets its turn.

5. Fairness

Problem:

It's hard to make sure all processes get a fair chance.
Some processes might get the resource again and again.

6. Complexity

Problem:

Writing correct code to handle mutual exclusion is hard.
Small mistakes can cause big problems, like crashes or data corruption

UNIT-3

What is Scheduling?
Scheduling means deciding which process should run on the CPU and when

Why Do We Need Scheduling?

Because:

- Many processes want the CPU at the same time.
- The CPU can work on only one process at a time (per core).
- Scheduling shares CPU time fairly and efficiently.

What are Performance Criteria?

Performance criteria are the rules or goals used to measure how good a scheduling algorithm is.

Important Performance Criteria

1.CPU Utilization

What it means: How **busy** the CPU is.

—> Goal: Keep the CPU working as much as possible so it doesn't sit idle.

—> Example: If CPU is busy 95% of the time, it's better than being busy only 50%.

2.Throughput

What it means: How many processes finish in a certain time period.

—>Goal: **More throughput = better performance.**

—> Example: If 10 processes finish per minute, it's better than 5 processes per minute.

3.Turnaround Time

What it means:Total time taken to finish a process (from submission to completion).

—>Goal: **Keep turnaround time low.**

—> Example: You submit a job at 10:00 am, it finishes at 10:10 am → turnaround time = 10 minutes.

4.Waiting Time

What it means: Time a process **spends waiting** in the ready queue (not running).

—>Goal: **Less waiting time is better.**

—> Example: If your process waits 2 minutes before running, it's better than waiting 10 minutes.

5.Response Time

What it means: Time from **sending a request** to getting **first response**.

—>Goal: Important for **interactive systems** (like typing or clicking), where you want **fast feedback**.

—> Example: You click a button, and it responds in 1 second. That's good response time.

6. Fairness

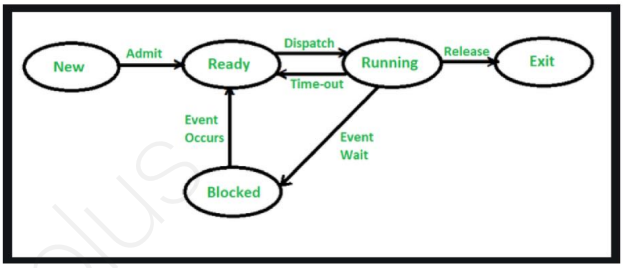
What it means: All processes get **a fair share of CPU time**.

—>Goal: No process should **starve** or be ignored forever.

—> Example:Even small jobs should get a turn, not be stuck behind big jobs forever.

QUE:-1 Define a process state? (AKTU 23-24)

ANS :-



1. New
The process is just created.

It hasn't started yet, but it's ready to be set up by the OS.
—>Like when you click to open a program, but it hasn't appeared yet.

2. Ready
The process is waiting for the CPU.

It's prepared to run but must wait its turn.
—>Like students raising their hands, waiting for the teacher to call on them.

3. Running
The process is currently using the CPU.

Only one process can run at a time (on one CPU).
—>Like the student currently talking to the teacher.

4. Blocked / Waiting
The process can't run now because it's waiting for something (like input from keyboard, or data from disk).
—>Like a student who can't speak yet because they're waiting to get their book.

5. Terminated / Exit
The process has finished its job or was stopped.

What is a Scheduler?
A scheduler is just a special part of the operating system that decides which process should run next on the CPU.

QUE:- 2 Explain in detail about the Process Control Block (PCB) in CPU Scheduling ?(AKTU 21-22 , 23-24)

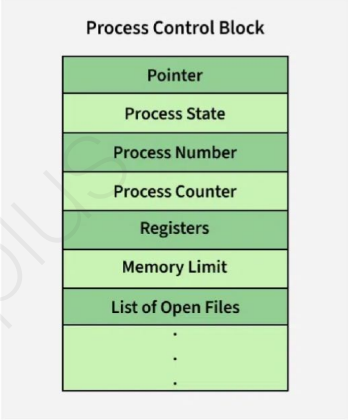
ANS:-

What is a PCB?

- PCB stands for Process Control Block.
- It is like a file or record card that the operating system keeps for every process.
- It stores all the information about that process, such as:

Who it is
What it is doing
What resources it is using
What state it is in

WHY do we need it?
When the CPU switches from one process to another (called context switching), the PCB helps remember exactly where the old process left off, so it can resume later without errors.



1. Process Number or ID

This is a unique number given to each process.
It helps the system identify which process is which.

—> Think of it like a roll number or ID card.

2. Process State

Shows what the process is doing right now:

New (just created)
Ready (waiting to run)
Running (using the CPU)
Waiting (blocked)
Terminated (finished)

3. Program Counter

This holds the memory address of the next instruction the process will run.

—> It's like a bookmark in a book, so the process knows where to continue.

4. Registers

These are small storage areas in the CPU the process is using:

—>They save temporary data the process needs to work.

5. List of Open Files

Shows which files the process is using right now.

For example, if a process is reading a text file or writing to a log.

6. CPU Scheduling Information

This includes:
The priority of the process (how important it is)

Pointers to the queues where the process is waiting
Any other scheduling settings

—>Helps the CPU decide which process should run next.

7. Memory Management Information

Information about the memory the process is using, such as:
Page tables or segment tables (depending on memory system)
Base register (start of memory)
Limit register (size of memory)

—>This keeps the process within its own area of memory.

8. I/O Status Information

Lists:
I/O devices the process is using (like printer, disk)
Files it is working with

—>Helps track what hardware and files the process needs.

9. Accounting Information

Includes:
How much CPU time the process has used
Process start time
User or account numbers
Any limits set for the process

This is useful for monitoring and billing.

Process Address Space
What is it?

- Every process (program running in memory) needs its own memory area to work in.
- This memory area is called the process address space.

Process Identification Information
What is it?

The operating system must be able to recognize and track each process.
So, it keeps some identification details for every process.

The main identification info includes:

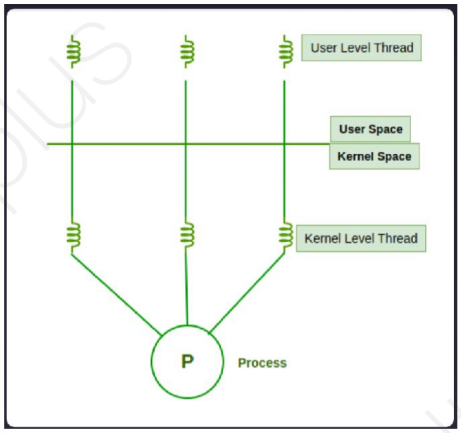
—>Process ID (PID)
A unique number given to every process.
Like an ID card number for the process.

—>Parent Process ID (PPID)
The ID of the parent process (the process that created this one).
Helps the system keep track of who created whom.

—>User ID (UID)
Shows which user started the process.
Useful for permissions and security.

QUE:-3 Explain in detail about the Threads and their management. (AKTU 21-22)

ANS :- Think of threads like workers in an office. Just as multiple workers can share the same office space and resources while doing different tasks, threads are lightweight processes that share common resources while performing different jobs simultaneously



What is a Thread?
A thread is like a single worker who has:

Their own desk (stack space)
Personal tools (register set)
Task list (program counter)
Unique ID (thread ID)
But unlike separate offices, threads share many common resources like files, printers, and meeting rooms

Why Do We Need Threads?
Imagine you're working on Microsoft Word while writing a document. Here's what happens behind the scenes:

One thread handles your typing
Another thread does automatic formatting
Yet another saves your work periodically
All this happens smoothly without interrupting your work

Types of Thread Management
There are two main ways threads are managed in computers:

1. **User-Level Threads**

Like office workers managing themselves
No direct involvement from operating system
Fast switching between tasks
Simple to implement

2.Kernel-Level Threads

Operating system actively manages threads
Better control over resources
Can utilize multiple processors efficiently
More complex but powerful

Thread Management

The operating system or a **thread library** (special software) handles the creation, scheduling, and termination of threads.

1. Creating Threads

- You can **create a thread** in a process to do a specific task.

- This is usually done using a thread library like:
  - **POSIX Threads (Pthreads)** in Unix/Linux
  - **Win32 Threads** in Windows
  - **Java Threads** in Java

2. Scheduling Threads

- The **thread scheduler** decides which thread runs at any given time.
- Threads can have **priorities** (some are more important than others).
- Scheduling can be **preemptive** (OS can interrupt and switch) or **cooperative** (threads give up control voluntarily).

3. Synchronizing Threads

Since threads **share memory**, they can **interfere with each other** if not careful.Example problem:

- Thread A updates a variable
- Thread B reads it at the same time
- The result may be **wrong**

To avoid this, we use **synchronization tools:**
Mutexes – Lock a resource so only one thread can use it at a time **Semaphores** – Count and control access to resources **Condition variables** – Make threads wait for certain conditions

4. Terminating Threads

- A thread finishes its task and **exits**.
- Sometimes one thread can **cancel another thread** if needed.
- The OS **cleans up resources** after a thread ends.

QUE:- 4 Evaluate strategies for deadlock prevention, avoidance, detection, and recovery in concurrent systems, highlighting their effectiveness and trade-offs? (AKTU 23-24)

QUE:-5 Define deadlock in operating systems, discussing the system model and conditions necessary for deadlock occurrence? (AKTU 23-24)

QUE:-6 Explain in detail about the Deadlock System model and Deadlock characterization. (AKTU 21-22)

ANS :- **What is Deadlock?**
Deadlock happens when processes are stuck forever because they are all waiting for each other to release resources.

Example:

- Process A has a printer, needs a scanner.
- Process B has a scanner, needs the printer.
- Both are **waiting forever**, holding onto one thing and needing the other.

How do you know a deadlock has happened?

A deadlock occurs if all 4 of these conditions happen at the same time (these are called Coffman conditions):

1. Mutual Exclusion:
Each resource is either used by one process or available.

Example: Only one process can use the printer.

2. Hold and Wait:
A process is holding at least one resource and waiting for more.
Example: Process A has the printer and waits for the scanner.

3. No Preemption:
Resources cannot be taken away forcibly.
Example: You can't snatch the printer from process A—it must release it.

4. Circular Wait:
There is a cycle of processes waiting on each other.
Example:

A waits for B's resource.
B waits for C's resource.
C waits for A's resource.
If all 4 are true, you have a deadlock.

Deadlock Prevention
How can you stop deadlocks before they happen?

You break at least one of the 4 conditions:

1.Mutual Exclusion:
Make resources shareable (not always possible—e.g., only one printer).

2.Hold and Wait:
Make processes ask for all resources at once (if they can't get everything, they wait).

3.No Preemption:
If a process is waiting, force it to release resources it already holds.

4.Circular Wait:
Give all resources a numbered order. Processes must request resources in order (no cycles).
Trade-off: Prevention makes the system simpler but less flexible.

Deadlock Avoidance and Detection
—> Deadlock Avoidance
Idea: Check in advance whether granting a resource will be safe.

The most famous method is the Banker's Algorithm (like a bank lending money):

The OS checks if giving resources to a process keeps the system in a safe state (everyone can finish eventually).

If yes, grant them.
If not, make the process wait.

Example:

If 3 processes want the printer, the OS makes sure it can schedule them in some order where all get to finish.

Deadlock Detection
—>Idea: Allow deadlocks to happen, but find them when they occur.

The OS regularly checks:

Resource Allocation Graphs (shows who holds and who waits).
Detection Algorithms to look for cycles.
If a cycle exists: deadlock detected.

Recovery
Once detected, deadlocks need to be resolved. Common recovery methods include:

Process Termination:
Stop one or more processes in the deadlock
Choose based on priority or minimal impact

Resource Preemption:
Take resources from one process
Give to another process in the deadlock
May need rollback recovery

Rollback Recovery:
Save checkpoints of process state
Return to previous safe state
Restart affected processes

UNIT-4

Basic bare machine

Bare Machine means:
A computer with only hardware, no operating system, and no software loaded.

Computer = Hardware + Operating System + Programs
Hardware: CPU, memory, disk, etc.
Operating System: Controls hardware, provides an interface.
Programs: What you actually run (e.g., games, apps).

Bare Machine = Only Hardware
No operating system.
No interface.
No helpers to load or run programs.

What is a Resident Monitor? (AKTU 23-24)

A Resident Monitor is a small control program that stays in memory to load, run, and manage other programs.
A resident monitor is a part of the operating system (OS) that helps manage and control how programs run on a computer. It's like a supervisor that ensures everything runs smoothly and in order.

Here's a simpler way to think about it:

Basic Role: It keeps the system running and makes sure that no program interferes with another. It helps in managing memory and keeps the system from crashing.

Always Active: The resident monitor is always in the background, working constantly. It doesn't turn off, even when you're not actively using your computer.

Task Management: It makes sure that programs or applications are running one at a time in the correct order. It also helps decide which program gets access to the CPU (the brain of the computer) when.

Memory Management: It keeps track of where each program is stored in memory. If the memory is full, it helps decide which programs need to be moved around to free up space.

Supervision: It supervises the execution of tasks and ensures that no program tries to do something it shouldn't (like accessing protected parts of memory).

Multiprogramming with fixed partitions

Multiprogramming with fixed partitions is a method used by the operating system (OS) to make efficient use of the computer's memory by running multiple programs at the same time.

Here's a breakdown :-

1. Fixed Partitions:

- The computer's memory is divided into fixed-sized blocks called partitions.
- Each partition can hold one program.
- The size of the partitions is fixed and doesn't change, even if a program is smaller or larger than the partition.

2. Running Multiple Programs:

- Multiprogramming means running multiple programs at the same time.
- The OS loads several programs into these fixed partitions and runs them simultaneously. While one program is waiting for input or processing, another can use the CPU.

QUE:-1 Compare and contrast multiprogramming with fixed partitions and variable partition (AKTU 23-24)

ANS :-
1. Fixed Partitions:
The computer's memory is divided into fixed-sized blocks called partitions.

Each partition can hold one program.

The size of the partitions is fixed and doesn't change, even if a program is smaller or larger than the partition.

2. Running Multiple Programs:
Multiprogramming means running multiple programs at the same time.

The OS loads several programs into these fixed partitions and runs them simultaneously. While one program is waiting for input or processing, another can use the CPU.

3. How It Works:

The computer has several partitions in memory, and each one can hold one program. If you have, say, 4 partitions, the OS can run 4 programs at once (as long as there's enough space).

If a program finishes, the OS can load another program into the empty partition, keeping the CPU busy and memory usage efficient.

Feature	Fixed Partitions	Variable Partitions
Partition Size	Fixed and unchangeable	Adjustable based on program size
Memory Efficiency	Can waste memory (internal fragmentation)	More efficient (less wasted space)
Flexibility	Less flexible	More flexible
Complexity	Simpler	More complex (needs dynamic management)
Fragmentation	Internal fragmentation	External fragmentation
Program Loading	Loads programs into fixed slots	Programs need exact partition size

Protection scheme

Protection schemes in an operating system (OS) are methods used to ensure that the computer's resources, such as memory, files, and devices, are used safely and securely by different users and programs.

- Memory Protection: Ensures programs cannot access each other's memory, preventing accidental or malicious interference.
- File Protection: Controls who can read, write, or execute files, using permissions to limit access.
- User Authentication: Verifies user identity (e.g., passwords, biometrics) before granting access.
- Access Control: Decides what actions users or programs can perform on resources, based on permissions.
- User Privileges: Assigns different levels of access to users, such as admin (full control) and regular users (limited control).

QUE :-2 Explain the principles of paging and segmentation as memory management techniques, comparing their suitability for different system architectures? (AKTU 23-24)

ANS:- Paging and Segmentation are both memory management techniques used by operating systems to handle how programs and data are stored in memory.

1. Paging:

What it is:

In paging, the memory is divided into small fixed-size blocks called pages. Similarly, the program's memory is divided into blocks of the same size called page frames.

Pages can be stored anywhere in memory, and the operating system keeps track of which pages are loaded into which frames.

How it works:

When a program needs to run, it is divided into small, equal-sized pages. The OS loads these pages into memory, wherever there's free space.

Advantages:

Efficient memory use: Memory is used in small, fixed blocks, so there's minimal wasted space.

Flexibility: Pages can be loaded into any available memory space, avoiding fragmentation.

Disadvantages:

External fragmentation is avoided, but internal fragmentation can occur if a page is only partially used.

2. Segmentation:

What it is:

In segmentation, the memory is divided into variable-sized segments, where each segment represents a logical part of a program (e.g., code, data, stack).

How it works:

A program is divided into segments based on its logical structure, and each segment is loaded into memory. The OS uses a segment table to map the segments to physical memory.

Advantages:

Logical grouping: Segments are based on the program's logical structure, like code or data, making it easier to manage.

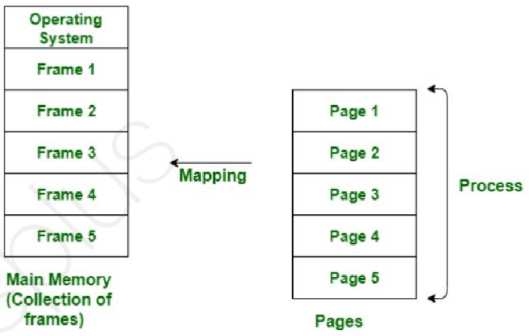
No internal fragmentation: Since segments are of variable size, you avoid wasting space inside segments.

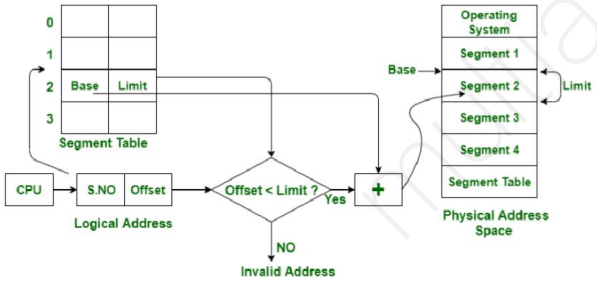
Disadvantages:

External fragmentation: As segments are of different sizes, free memory may become fragmented, making it hard to allocate space for new segments.

Comparison:

Feature	Paging	Segmentation
Memory Division	Divides memory into equal-sized pages	Divides memory into variable-sized segments
Management	Uses a page table to track pages	Uses a segment table to track segments
Fragmentation	Avoids external fragmentation but can have internal fragmentation	Can lead to external fragmentation due to variable sizes
Memory Usage	Efficient, with minimal wasted space	Can have wasted space due to fragmentation
Suitability	Ideal for systems where programs are similar in size and memory use	Better for systems requiring logical grouping, such as large, complex programs





QUE:-3 Discuss virtual memory concepts, including address translation, page tables, and demand paging mechanisms? (AKTU23-24)

ANS:- Virtual Memory is a technique used by operating systems to allow a computer to use more memory than is physically available. It gives the illusion to the user that they have a large, continuous block of memory, even though the actual physical memory (RAM) is smaller. Here's an easy-to-understand breakdown of its key concepts:

1. Virtual Memory Concept:
What it is: Virtual memory allows programs to use more memory than the computer physically has by temporarily transferring data to the hard drive (or swap space) when RAM is full. It makes the system think it has a lot of memory, even if it doesn't.

Why it's needed: Without virtual memory, a program would only be able to use the physical memory available in RAM, which can be limiting. Virtual memory helps programs run without worrying about memory size.

2. Address Translation:
What it is: Virtual memory uses virtual addresses (the memory addresses used by a program) that are different from physical addresses (the actual locations in RAM).

How it works: When a program accesses memory, it uses virtual addresses. The memory management unit (MMU) in the CPU translates these virtual addresses into physical addresses where the data is stored in RAM or disk.

3. Page Tables:
What they are: A page table is a data structure used by the operating system to keep track of how virtual addresses are mapped to physical addresses.

How they work: The memory is divided into small fixed-sized blocks called pages. When a program needs data, the operating system uses the page table to figure out which page (or part of the program) is stored in which physical location.

4. Demand Paging:
What it is: Demand paging is a technique where pages of memory are loaded into physical memory only when they are needed, not in advance.

How it works: When a program tries to access a page that is not currently in RAM (because it was swapped out to the disk), the OS brings that page into RAM from the disk. This process is called a page fault.

5. How Demand Paging Helps:
It allows programs to use more memory than physically available by swapping data in and out of RAM.

Programs only use memory for the parts they are actively working with, improving efficiency.

6. Page Faults:
A page fault occurs when the program tries to access a page that's not in RAM. The OS will handle this by fetching the page from the disk (this can be slower than accessing RAM).

After the page is fetched from the disk, the program can continue its work.

In Summary:
Virtual Memory gives the illusion of more RAM by swapping data between physical memory and the disk.

Address Translation: Virtual addresses used by programs are mapped to physical addresses using the MMU and page tables.

Page Tables: Keep track of the mapping between virtual and physical memory.

Demand Paging: Only loads the pages into memory when they are needed, which optimizes memory use.

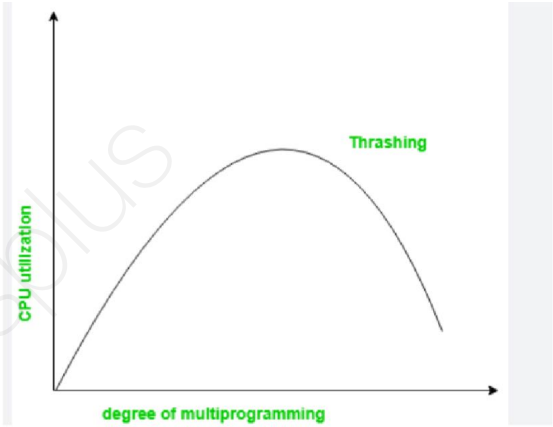
Page Faults: Happen when a program tries to use data not currently in RAM, causing the OS to load the data from the disk.

1. Thrashing:

What it is: Thrashing occurs when the operating system spends most of its time swapping data between the hard disk and RAM, instead of running programs. This makes the system very slow and inefficient.

Why it happens: It usually happens when there is not enough physical memory (RAM) for the programs running. As a result, the OS continuously swaps data in and out of memory, causing a lot of delays.

Example: Imagine you have too many programs open at once, and the computer doesn't have enough RAM. The system constantly has to move parts of programs to the disk, making everything slow and unresponsive. That's thrashing.



2. Cache Memory:
What it is: Cache memory is a small, fast type of memory located close to the CPU. It stores frequently accessed data or instructions, so the CPU can quickly access them instead of going to the slower main memory (RAM).

Why it's important: Cache memory speeds up the system by reducing the time it takes for the CPU to fetch data. It acts as a temporary storage that holds the most commonly used data or instructions.

Example: If you're repeatedly using a program's specific function, the CPU will store the result of that function in the cache so it doesn't have to repeat the entire calculation each time you use it.

3. Locality of Reference:
What it is: Locality of reference refers to the tendency of a program to access a small set of memory locations repeatedly over a short period of time.

Spatial locality: This refers to accessing memory locations that are close to each other (e.g., accessing nearby elements in an array).

Temporal locality: This refers to accessing the same memory locations repeatedly within a short time (e.g., using the same variable multiple times in quick succession).

Why it's important: Understanding locality helps the operating system manage memory more efficiently. It ensures that frequently used data is kept in the cache or in memory, reducing the time it takes to fetch the data.

QUE:-5 Multilevel feedback Queue Scheduling(AKTU 22-23)

ANS:- **Multilevel Feedback Queue (MFQ) Scheduling i**s a CPU scheduling technique that uses multiple queues with different priorities to manage processes.

Multiple Queues: Processes are placed in different queues based on their CPU usage, with higher-priority queues for short tasks and lower-priority queues for long tasks.

Feedback: If a process uses too much CPU time, it moves to a lower-priority queue. If it waits too long, it may be promoted to a higher-priority queue.

Aging: Prevents starvation by moving processes from lower-priority queues to higher-priority ones over time.

UNIT-5

I/O devices in an operating system refer to the hardware components that allow the computer to interact with the outside world. "I/O" stands for Input/Output, meaning these devices are used to input data into the computer and output data from the computer.

1. Input Devices:
What they are: These devices send data to the computer for processing.

Examples:

Keyboard: Used to type text into the computer.
Mouse: Used to interact with the computer's graphical interface by moving a pointer on the screen.
Scanner: Used to convert physical documents or images into digital form.
Microphone: Used to record sound or voice input.

2. Output Devices:
What they are: These devices receive data from the computer and display or output it in a form that people can understand or use.

Examples:

Monitor: Displays the visual output of the computer, such as text, images, or videos.
Printer: Prints documents or images onto paper.
Speakers: Output sound from the computer, like music or alerts.

I/O Subsystem

The I/O Subsystem in an operating system (OS) refers to the part of the system responsible for handling input and output (I/O) operations. It manages the communication between the computer and external devices like keyboards, mice, printers, hard drives, and more.

What is the I/O Subsystem?
I/O (Input/Output): Input refers to data received by the computer from external devices (like typing on a keyboard or clicking the mouse). Output refers to data sent from the computer to external devices (like displaying text on the screen or printing a document).

The I/O subsystem makes sure these operations are handled properly, ensuring that data is transferred smoothly between the computer and external devices.

I/O Buffering (AKTU21-22)

I/O Buffering is a technique used in computers to improve how data is transferred between devices and memory. It temporarily stores data in a special area of memory called a buffer. This helps the system manage the difference in speed between fast and slow devices, making data transfer more efficient.

Why I/O Buffering is Used:

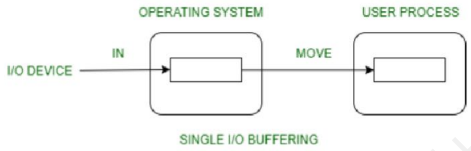
Speed Mismatch: Sometimes, data is produced by one device (like a modem) faster than another device (like a hard disk) can handle it. Buffering helps by holding the data temporarily until the slow device is ready to process it.

Efficient Data Transfer: The system doesn't have to constantly wait for data to be transferred. Instead, it can work with one set of data while another set is being transferred, making things quicker and smoother.

Types of I/O Buffering Techniques:

1.Single Buffer:

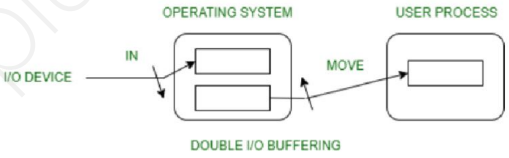
- The system uses one buffer to temporarily store data.
- Example: When you download a file, the system might use a single buffer to hold the data before it moves it to storage.
- This is useful for block-oriented devices (like hard drives) or stream-oriented devices (like keyboards or printers).



Double Buffer:

- In this method, the system uses two buffers. One buffer holds the data being processed, and the other is filled with new data.
- This helps to keep things moving without waiting for data to finish processing before starting the next task.

Disadvantage: It can get more complex, and might not be as efficient if data is transferred very quickly.

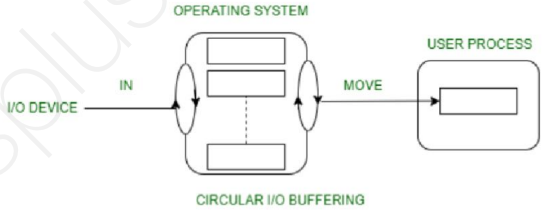


Circular Buffer:

A circular buffer is a bit more advanced. It uses multiple buffers arranged in a loop.

Once one buffer is full, the system starts writing data to the next buffer, and so on. After the last buffer is filled, it loops back to the first one.

This allows continuous data transfer without gaps, especially for devices that constantly send or receive data, like network devices.



QUE:-1 Compare and contrast different disk storage technologies, such as HDDs and SSD?(AKTU 23-24)

ANS:-**Comparing HDDs (Hard Disk Drives) and SSDs (Solid State Drives):**
Both HDDs and SSDs are used to store data in computers, but they work in very different ways. Let's compare them in simple terms:

1. Technology:
HDD (Hard Disk Drive):

Uses spinning disks (platters) to read and write data.

Heads move over the spinning disks to access data, much like a record player's needle reading grooves.

SSD (Solid State Drive):

Uses flash memory chips to store data. There are no moving parts.

Data is stored on memory cells, and the drive reads or writes data electronically, which makes it much faster than HDDs.

2. Speed:
HDD:

Slower because data has to be read from or written to spinning disks, and the head must physically move to the correct location.

Example: Imagine looking through a photo album where you have to flip through each page to find a specific photo. It's slow, right?

SSD:

Much faster because it has no moving parts and uses electronic circuits to access data instantly.

Example: It's like instantly finding a photo on your phone with a simple search.

3. Durability:
HDD:

More fragile because it has moving parts (spinning disks and heads). If the drive is dropped or bumped too much, it could damage the disks and cause data loss.

SSD:

More durable because there are no moving parts. It's less likely to be damaged if dropped or shaken.

4. Capacity:
HDD:

Usually offers larger storage at a lower price. You can get HDDs with several terabytes (TB) of space for a reasonable cost.

SSD:

Typically offers less storage for a higher price compared to HDDs. But the price of SSDs is dropping over time, and they are becoming available in larger sizes.

5. Price:
HDD:

Cheaper for large storage, making them a good choice for those who need a lot of space on a budget (e.g., for storing movies, large files, or backups).

SSD:

More expensive per gigabyte of storage, but the prices have been decreasing steadily.

6. Noise and Heat:
HDD:

Noisy due to the spinning disks and moving heads.

Can generate more heat because of the moving parts.

SSD:

Silent, as there are no moving parts.

Generates less heat, which can be a benefit for laptops and other portable devices.

Feature	HDD (Hard Disk Drive)	SSD (Solid State Drive)
Technology	Spinning disks and moving parts	Flash memory with no moving parts
Speed	Slower (data access is mechanical)	Faster (instant data access)
Durability	Fragile (prone to damage from shock)	Durable (no moving parts, shock-resistant)
Capacity	Large capacity at a lower price	Smaller capacity at a higher price
Price	Cheaper for large storage	More expensive per GB
Noise	Noisy (due to spinning disks)	Silent (no moving parts)
Heat	Generates more heat	Generates less heat
Power Consumption	More power-consuming	Less power-consuming
Lifespan	Long-lasting, but mechanical wear occurs	Limited write cycles, but lasts many years

QUE:-2 Discuss the organization of files within file systems, including contiguous, linked, and indexed allocation methods? (AKTU 23-24)

ANS:- **File Organization in File Systems**
In a computer's file system, data is stored in files, and these files need to be organized in a way that makes it easy to read, write, and manage. The way the file system organizes data on the disk is called file allocation. There are three main methods for organizing files: contiguous allocation, linked allocation, and indexed allocation.

1. Contiguous Allocation:
What it is: In contiguous allocation, a file is stored in one continuous block of space on the disk. The file occupies a sequence of adjacent disk blocks.

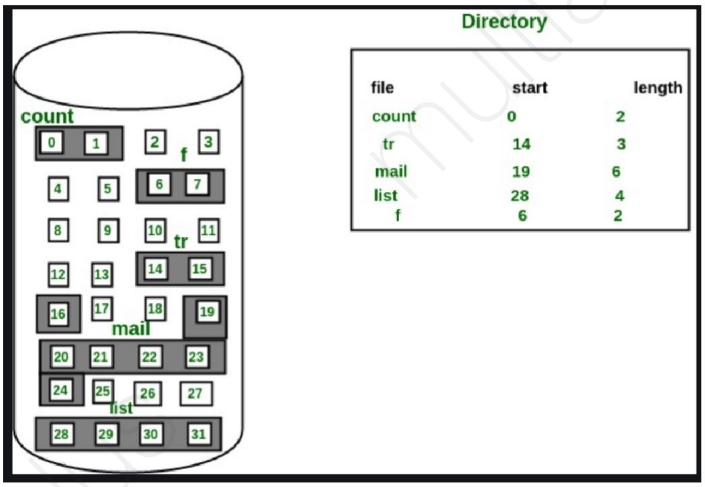
How it works: When a file is created, the system looks for a large enough space (contiguous blocks) to store it. All the data of the file is stored next to each other in a continuous area.

Example: Think of a row of seats in a theater. If a file is small, it will fit easily in a few seats (blocks). If the file is larger, the system will try to find a long enough row of empty seats for it.

Advantages:
Fast access: The system can read or write the entire file without needing to move around to different parts of the disk. It's like reading a book where all the pages are in order and next to each other.

Disadvantages:
Fragmentation: Over time, as files are created and deleted, free space becomes scattered, and there may not be enough large, continuous spaces for new files. This is called external fragmentation.

Wasted Space: If a file doesn't perfectly fit into a space, the remaining part of the block is wasted.



2. Linked Allocation:
What it is: In linked allocation, each file is broken into smaller blocks (called blocks or clusters), but these blocks are not stored next to each other. Instead, each block points to the next block of the file. So, the blocks can be scattered anywhere on the disk, but each block knows where the next one is.

How it works: When the file system creates a file, it assigns each block of the file a pointer to the next block in the sequence. The last block of the file has a pointer that points to null, indicating the end of the file.

Example: Imagine a train where each car (block) is connected by a link (pointer). The first car knows where the second car is, the second car knows where the third is, and so on. But the cars (blocks) can be placed anywhere on the track (disk).

Advantages:

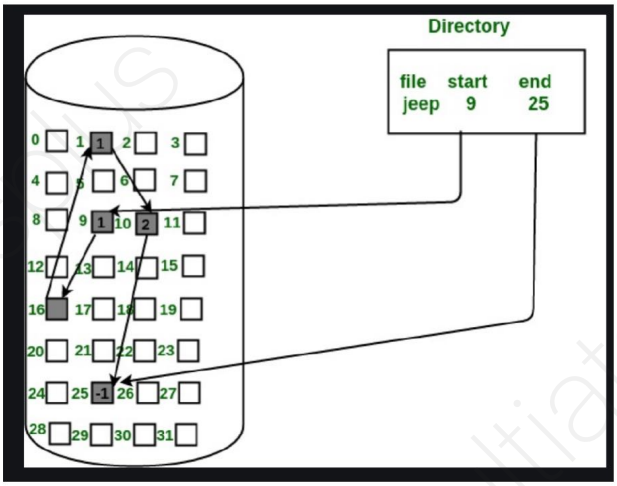
No external fragmentation: The system doesn't need to find large, continuous spaces for files, so it's more flexible.

Efficient use of space: Files don't need to occupy contiguous blocks, so space is used more efficiently.

Disadvantages:

Slow access: To read the file, the system must follow the pointers from one block to the next. It's like looking up a word in a dictionary where you have to flip through every page until you find the one you need.

Overhead: Each block requires additional space to store the pointer, which can lead to more storage overhead.



3. Indexed Allocation:

What it is: In indexed allocation, each file has its own index block that stores pointers to all the blocks of the file. Instead of having each block point to the next, the index block contains a list of all the block addresses.

How it works: When a file is created, the system builds an index block that lists the locations of all the file's blocks. The file system can directly access any block by looking it up in the index block.

Example: Imagine a library where each book has a catalog card (index block) listing the location of every page (block). To find a page, you just check the catalog card instead of searching through the entire library.

Advantages:

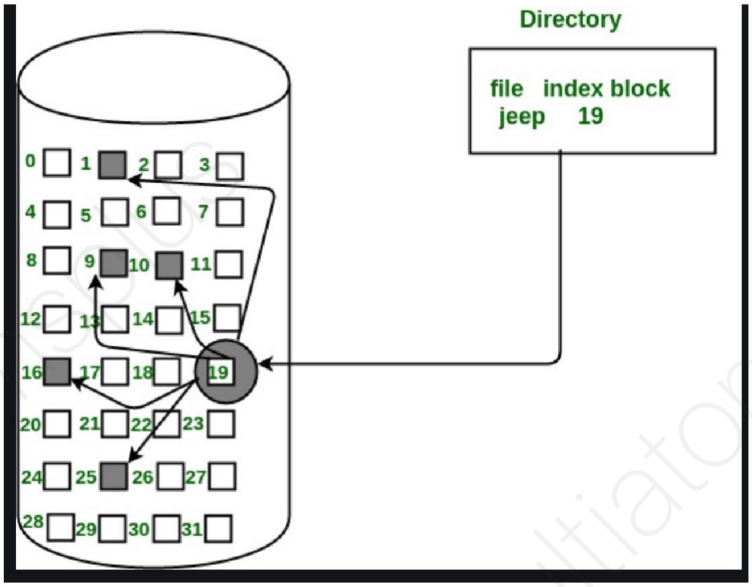
Fast access: The system can go directly to any block of the file, just like looking up a page number in a book. There's no need to follow pointers like in linked allocation.

No fragmentation: Since the file's blocks can be scattered across the disk, but still be managed via the index, it avoids the problem of fragmentation.

Disadvantages:

Space overhead: The index block itself uses space. If the file is large, the index block can become large as well.

Complexity: Managing the index block adds some complexity to the file system.



QUE:-3 Explain the structure and management of file directories, including hierarchical and flat directory structures? (AKTU 23-24)

ANS:-**File Directory Structure and Management**
A file directory is like a storage container for files in a computer. It helps organize, store, and manage files so that users can easily find and access them. Just like a physical filing cabinet with folders, a file directory stores files in an organized manner, and it's managed by the operating system.

There are two main ways to organize directories: flat directory structure and hierarchical directory structure.

1. Flat Directory Structure:
What it is: In a flat directory structure, all files are stored in a single list or folder. There are no subfolders or organization beyond the top level.

How it works:

Imagine a simple list of files without any subcategories. All the files are stored in one big "folder" and are identified by their names.

For example, if you have three files named "file1.txt," "file2.txt," and "file3.txt," all of them are in the same directory with no separation or categories.

Advantages:

Simple to understand and manage when there are only a few files.

No need to worry about complex folder structures.

Disadvantages:

Hard to manage when there are a lot of files, because everything is in the same place.

No way to organize files by categories or types.

2. Hierarchical Directory Structure:
What it is: In a hierarchical directory structure, files are organized into folders and subfolders (also called directories and subdirectories), creating a tree-like structure.

How it works:

The top-level directory is often called the root directory, and under it, you can have several subdirectories (folders).

Each subdirectory can contain other subdirectories and files. This way, you can organize files into categories and subcategories.

Files are stored inside folders, and each folder can contain more folders or files.

Example:

Suppose you have a folder called Documents. Inside the Documents folder, you can have subfolders like Work, School, and Personal. Each of these subfolders can contain files like "project.docx" or "notes.txt."

This is like a filing cabinet with drawers (subdirectories) for different categories, and each drawer contains folders with files inside.

Advantages:

Easy to organize files based on categories (e.g., documents, images, videos).
Efficient file management, as files are grouped together in their respective folders.
Faster searching for files because you know where to look based on their category.

Disadvantages:

Can become complex with too many subdirectories. If there are too many nested folders, finding files can still take time.
Requires a bit more effort to set up and manage compared to a flat directory.

4. File Directory Management:

The operating system is responsible for managing the file directory structure. It performs tasks like:

Creating directories: The OS allows users to create new directories and subdirectories to organize files.

Naming files: Files are given names to help users identify them. The OS also ensures that no two files in the same folder have the same name.

Access control: The OS manages who can access files and directories. Permissions can be set to allow reading, writing, or executing files.

File retrieval: The OS keeps track of where each file is stored in the directory structure and provides a way to access them.

File deletion: The OS allows users to delete files and directories when they are no longer needed.

QUE:-4 Explain the concept of file system management. Also, explain various file allocation and file access mechanisms in details. (AKTU 22-23)

ANS:- **File System Management:**

File system management refers to how an operating system (OS) organizes and controls access to files stored on a computer. It makes sure that files are stored, retrieved, and managed in an efficient and secure way.

File Access Mechanisms:

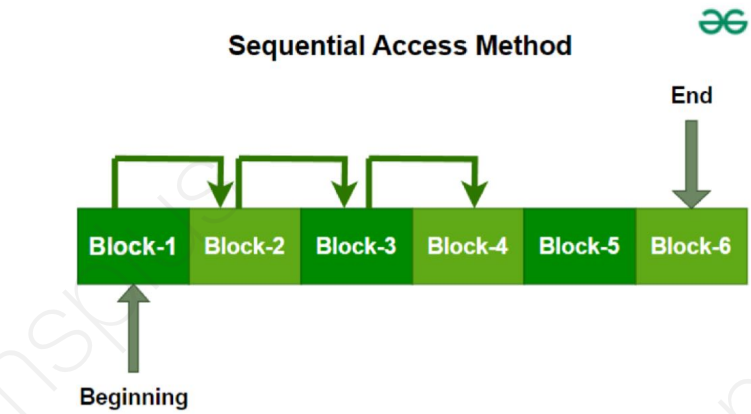
File access mechanisms define how data can be read from or written to files. These mechanisms are ways to access the content inside a file. There are different methods of accessing files depending on how the file is organized and what operations are needed.

1. Sequential Access:
How it works: In sequential access, data is read or written in order, one piece after another, starting from the beginning of the file.

Example: Reading a book page by page, from start to finish. You cannot skip ahead to a random page, you need to go in order.

Advantages: Simple and fast for files where you need to process everything in order.

Disadvantages: Slow if you need to access a specific part of the file, because you have to read through all the data before reaching the desired part.

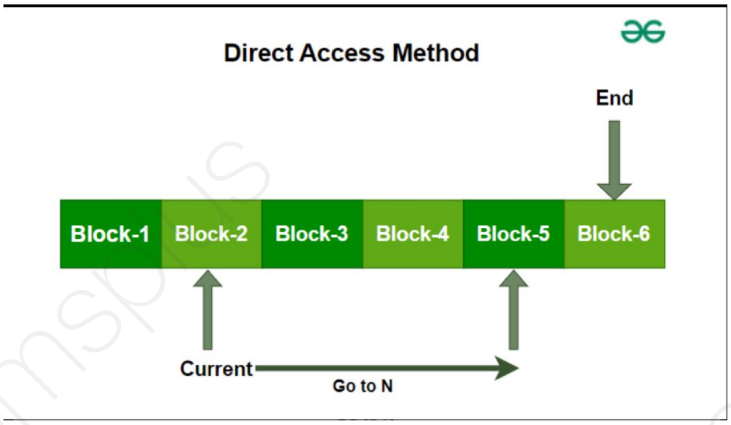


2. Direct Access (Random Access):
How it works: In direct access, you can jump directly to any part of the file, without reading through all the data.

Example: Imagine a DVD player where you can skip directly to any scene, rather than having to watch the movie from the beginning.

Advantages: Faster access to any part of the file, especially if you only need a specific piece of data.

Disadvantages: More complex to manage and requires indexing or pointers to find data quickly.

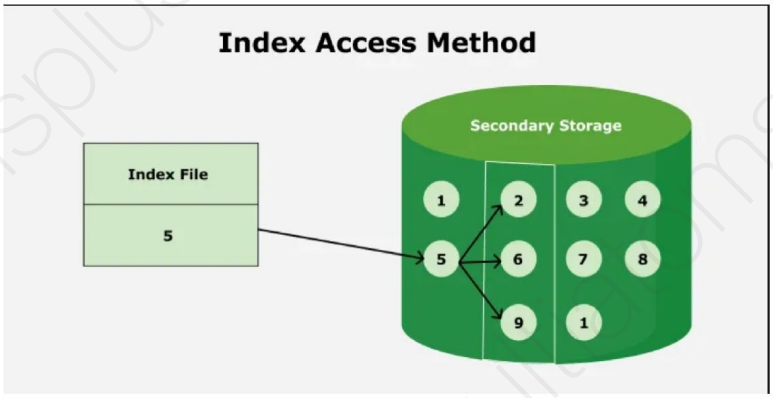


3. Indexed Access:
How it works: Indexed access is similar to random access, but the system uses an index to find the location of data.

Example: This is like using an index in a textbook to find the exact page with the information you need.

Advantages: Very efficient for large files with a lot of data.

Disadvantages: Needs extra memory to store the index, and managing the index can be complex.



File System Protection & Security

File System Protection and Security ensure files are safe from unauthorized access and damage. Here's how they work:

File Protection: Uses permissions (read, write, execute) to control who can access and modify files.

User Authentication: Verifies identity through username and password (or biometrics) before accessing files.

Access Control: Manages who can access or modify files using models like Discretionary Access Control (DAC) or Mandatory Access Control (MAC).

Encryption: Converts data into unreadable code, ensuring only authorized users can access it using a decryption key.

Backup and Recovery: Regularly copies files to a secure location to recover them in case of loss or damage.

NUMERICALS

Scheduling Times :-

1. **Arrival Time** :- Time at which process arrives.
2. **Burst/Service Time** :- Amount of time a process runs on CPU.
3. **Waiting Time** :- Amount of time for which process waits in ready state.
4. **Completion Time** :- Time at which process completes.
5. **TAT**:- Amount of time process spends from arrival to completion
   TAT=CT-AT
   WT=TAT-BT

Scheduling Times :-

.**Response Time** :- Amount of time from arrival till first time execution of process.
.**Throughput** :- No of process executed per unit time.
Throughput = no of processes executed / scheduling length

Non-Preemptive :- Once a process scheduled to run then it can not be forcefully taken out of CPU .

Preemptive :- A running process can be taken out of CPU Forcefully

Scheduling Algorithm :-

1. FCFS

—> Scheduling Criteria whichever process has min arrival time .

—> Tie-Breaker :- Arrival Time

—> Non-Preemptive

(AKTU-18-19)

(a) Consider the following process:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

What is the average waiting and turn around time for these process with

(i) FCFS Scheduling

Convoy Effect :- If a heavy process is scheduled first it delay all other process execution significantly hence performance of the system slow down .

Only FCFS suffers from convoy effect.

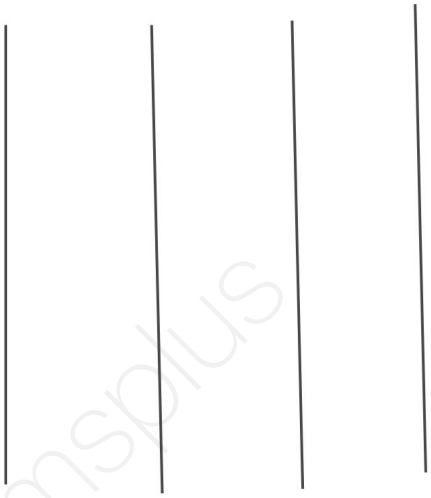
SJF(Shortest Job First)

—> **Scheduling Criteria** :- Schedule process with smallest burst time .
—>**Tie-Breaker** :- FCFS
—>Non-Preemptive

PROCESS	AT	BT

SRTF (Shortest Remaining Time First)

—> **Scheduling Criteria** :- Smallest BT first :- Tie-Breaker = FCFS
—>Preemptive



Priority Based Algorithm

—>Highest Priority process first

—>Tie-Breaker = Given in Que

—>Preemptive / Non-Preemptive

(AKTU 22-23)

(b) For the following processes, draw Gantt chart to illustrate the execution using,
(i) Preemptive priority scheduling
(ii) Non-Preemptive priority scheduling.
Also, Calculate average waiting time and average turnaround time.
(Assumption: A larger priority number has higher priority.)

Process	Arrival Time	Burst Time	Priority
A	0	5	4
B	2	4	2
C	2	2	6
D	4	4	3

ROUND-ROBIN

—>Scheduling Criteria :- Arrival Time + Quantum Q

—>Tie-breaker :- FCFS

—>Maximum amount of time for which a process runs on CPU at a time.

PROCESS	AT	BT

(AKTU 22-23)

(b) Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 49. Calculate the net head movement using:

(i) SSTF
(ii) SCAN
(iii) CSCAN
(iv) LOOK

(a) Explain the concept of demand paging. Consider the given references to the following pages by a program:
0,9,0,1,8,1,8,7,8,7,1,2,8,2,7,8,2,3,8,3
How many pages faults will occur if the program has three-page frames available to it and uses:
(i) FIFO replacement
(ii) LRU replacement
(iii) Optimal replacement

Peterson Algorithm

```
Boolean Flag[2];
Int turn;
Flag[0] = Flag[1] = false
```

Process Pi	Process Pj
{	do{
Flag[i] = true;	Flag[j] = true;
turn = j; (5th)	turn = i; (6th)
while (Flag[j] && turn == j);	while (Flag[i] && turn == i);
C.S	C.S
Flag[i] = false;	Flag[j] = false;
R.S	R S
} while(1);	} while(1);

$x = 2$

$x + 2 \rightarrow 3$

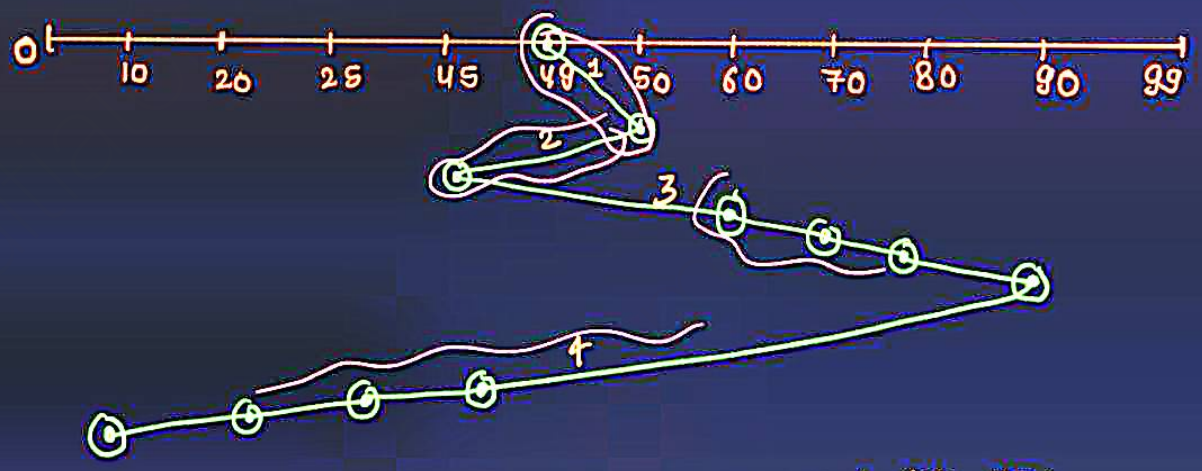
$x + 2 \rightarrow 4$

$x + 1$

(b) Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 49. Calculate the net head movement using:

(i) SSTF
(ii) SCAN
(iii) CSCAN
(iv) LOOK

Soln (i) SSTF → Fulfill Req^n of nearest Cy^m..

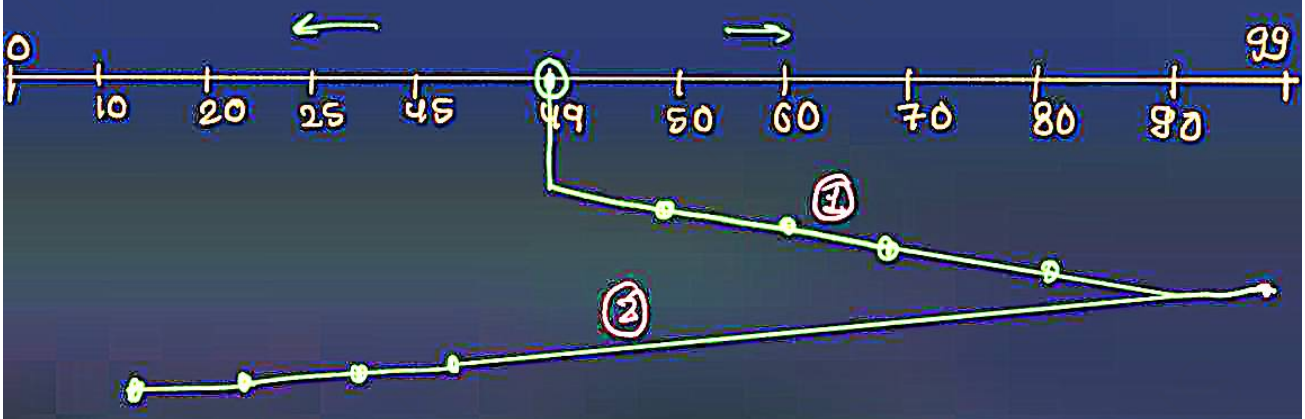


Net Head Mov^n = (50-49) + (90-45) + (90-10) + (50-45)
(1) (3) (4) (2)

= 1 + 80 + 45 + 5

Net = 131 tracks Ans

(ii) SCAN :- arm^m → Chalana → toward large Value
→ toward Smallest Value

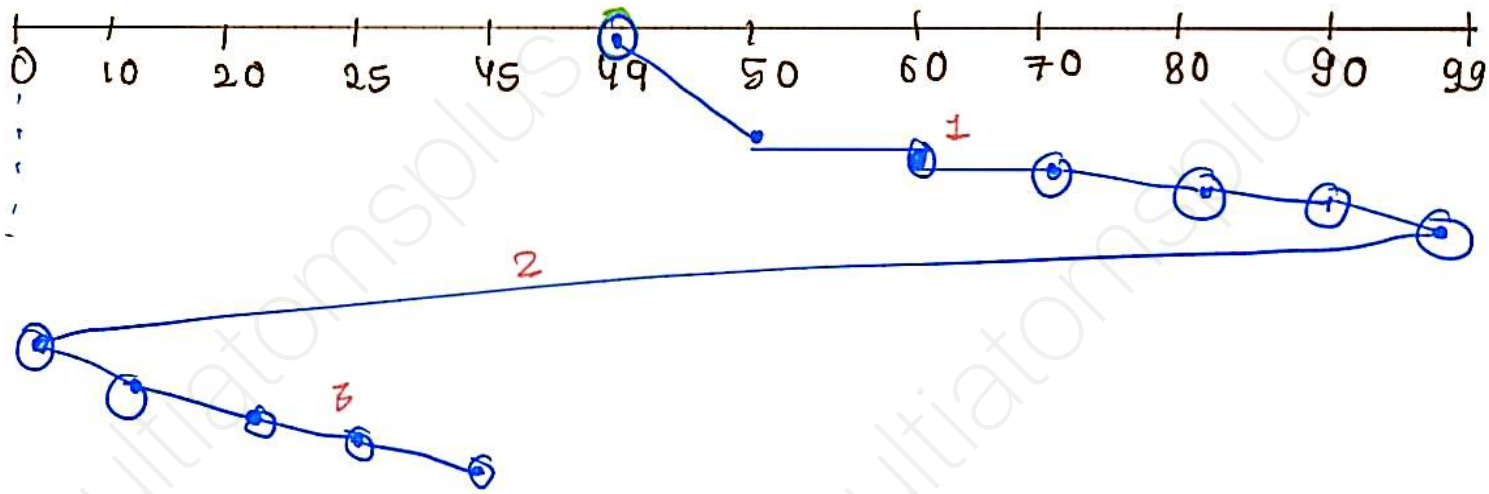


Net Head Mov^m = (99-49) + (99-10)

= 50+89 = 139 tracks Ans

[AKTU 22-23]

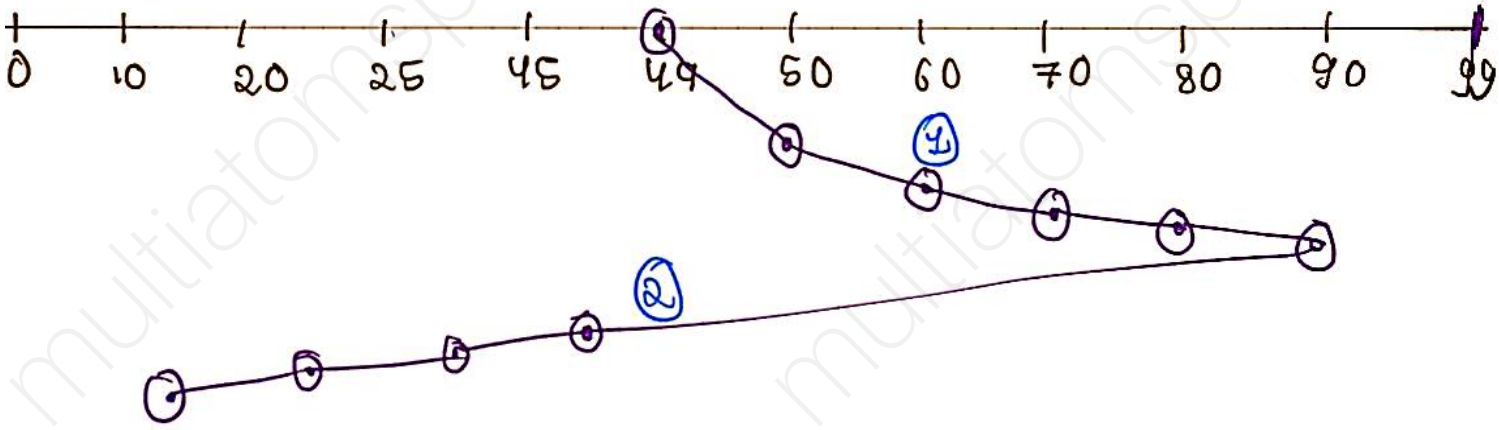
(iii) C-Scan :- Should move toward large



$= (99-49) + (99-0) + (45-0)$

$= 194$ tracks

(iv) Look :- toward large



Head = $(90-49) + (90-10)$

$= 41 + 80$

$= 121$ tracks

(a) Explain the concept of demand paging. Consider the given references to the following pages by a program:
0,9,0,1,8,1,8,7,8,7,1,2,8,2,7,8,2,3,8,3
How many pages faults will occur if the program has three-page frames available to it and uses:
(i) FIFO replacement
(ii) LRU replacement
(iii) Optimal replacement

0 9 0 1 8 1 8 7 8 7 1 2 8 2 7 8 2 3 8 3

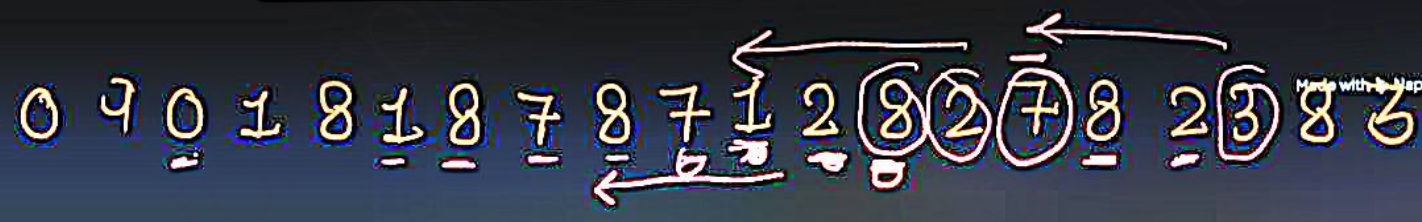
FIFO

	0	9	0	1	8	1	8	7	8	7	1	2	8	2	7	8	2	3	8	3
f1				1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
f2		9	9	9	9	9	9	7	7	7	7	7	7	7	7	7	7	8	8	
f3	0	0	0	0	8	8	8	8	8	8	8	8	8	8	8	8	3	3	3	
	*	*	H	*	*	H	H	*	H	H	H	*	H	H	H	H	H	*	*	

No of Page Fault :- 8 Ans

No of Page Hit :- 12 Ans

(a) Explain the concept of demand paging. Consider the given references to the following pages by a program
0,9,0,1,8,1,8,7,8,7,1,2,8,2,7,8,2,3,8,3
How many pages faults will occur if the program has three-page frames available to it and uses:
(i) FIFO replacement
(ii) LRU replacement
(iii) Optimal replacement



1, 2, 7

1, 2, 8

No of Page Fault :- 9 =

→ Replace the page which is not used in longest dimension of time in future.

(a) Explain the concept of demand paging. Consider the given references to the following pages by a program
0,9,0,1,8,1,8,7,8,7,1,2,8,2,7,8,2,3,8,3
How many pages faults will occur if the program has three-page frames available to it and uses:
(i) FIFO replacement
(ii) LRU replacement
(iii) Optimal replacement

0 9 0 1 8 1 8 7 8 7 1 2 8 2 7 8 2 3 8 3

Made with Napkin

2,7

	0	9	0	1	8	1	8	7	8	7	1	2	8	2	7	8	2	3	8	3
f1				1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
f2		9	9	9	9	9	9	7	7	7	7	7	7	7	7	7	7	3	3	3
f3	0	0	0	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
	*	*	H	*	*	H	H	*	H	H	H	*	H	H	H	H	H	*	H	H

No of Page Fault :- 7

Peterson Algorithm

```
Boolean Flag[2];
Int turn;
Flag[0] = Flag[1] = false
```

Process Pi	Process Pj
{	do{
Flag[i] = true;	Flag[j] = true;
turn = j; (5th)	turn = i; (6th)
while (Flag[j] && turn == j);	while (Flag[i] && turn == i);
C.S	C.S
Flag[i] = false;	Flag[j] = false;
R.S	R S
} while(1);	} while(1);

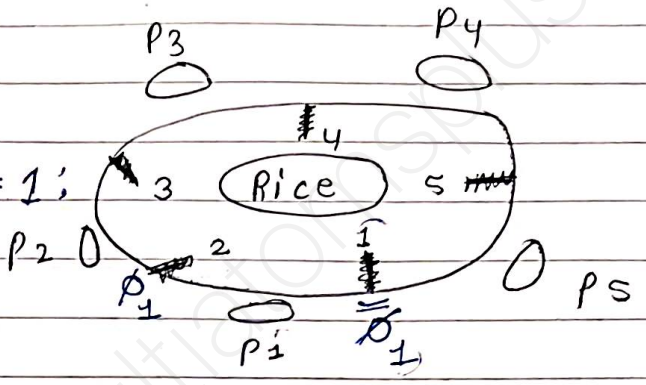
$x = 2$

$x + 2 \rightarrow 3$

$x + 2 \rightarrow 4$

$x + 1$

Dinning Philospher Problem



Semaphore chopstick [5] = 1;

Philosopher i

do {

wait (chopstick [i]);

wait (chopstick [i+1] % 5);

2 % 5 = 2

⋮

Eat

Signal (chopstick [i]);

Signal (chopstick [i+1] % 5);

2 % 5 = 2

⋮

Think

} while (1)

S > 0

wait S = S - 1

Signal S = S + 1;