CBCS SCHEME

USN | | | | | | | | | | |

BCS303

Third Semester B.E/B.Tech. 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	C
1	a.	Explain the various types of system calls with an example for each.	8	L2	CO1
	b.	Explain cloud computing its types and their services it offers.	8	L2	CO1
	c.	What is dual mode operation and what is the need of it?	4	L2	CO1
		OR			
2	a.	Explain the different architecture of OS starting from Simple Structure, Layered Structure, Micro Kernels, Modules and Hybrid Systems.	8	L4	CO1
	b.	Discuss the essential properties of the following types of systems : i. Time sharing system ii. Multi-programmed batch systems.	6 6	L2	CO1
		Module – 2			
3	a.	Explain Inter Process Communication.	6	L2	CO2
	b.	Discuss Multilevel Queue Scheduling Algorithm.	6	L4	CO2
	c.	Consider the set of 3 processes whose arrival time and burst time are given below.	8	L3	CO2

Process Id	Arrival Time	Burst Time
P1	0	9
P2	1	4
P3	2	9

If the CPU scheduling policy is SRTF, calculate the average waiting time and average turnaround time.

OR

4	a.	Compare User Level Threads and Kernel Level threads.	4	L4	CO2
	b.	Illustrate with a neat sketch, Process States and Process Control Block (PCB).	8	L2	CO2
	c.	Consider the set of 6 processes whose arrival time and burst time are given below.	8	L3	CO2

Process ID	Arrival Time	Burst Time
P_1	5	5
P_2	4	6
P_3	3	7
P_4	1	9
P_5	2	2
P_6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 3, calculate Average Waiting time and Turnaround time.

Module – 3

5	a.	Discuss in detail the critical section problem and write the algorithm for producer consumer problem.	10	L2	CO3
	b.	Consider the following system using data structures in the Bankers Algorithm with resource type ABC Maximum instance present in the system A = 10, B = 5, C = 7.	10	L3	CO3

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

i. Calculate Need Matrix
ii. Check whether system is safe or not.

OR

6	a.	Outline the solutions of Dinning –Philosopher problem.	5	L2	CO3
	b.	Describe a resource allocation graph with an example.	5	L4	CO3
	c	Using Bankers algorithm, solve the following problem :	10	L2	CO3

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

i. Calculate the Need Matrix
ii. Check whether system is safe or not.

		Module – 4			
7	a.	Discuss the given memory management technique with diagram i Paging ii Translation Look-Aside Buffer.	6 6	L2	CO4
	b.	Discuss about Contiguous Memory Allocation with a neat diagram.	8	L2	CO4
		OR			
8	a.	Consider the reference string : 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 For a memory with three frames and calculate number of page faults by using i. LRU replacement ii FIFO replacement.	10	L3	CO4
	b.	Describe the process of demand paging in OS.	10	L2	CO4
		Module – 5			
9	a.	Explain in detail about directory and disc structure.	6	L2	CO5
	b.	Analyze the file system implementation.	6	L4	CO5
	c.	The requested tracks, in the order received are {176, 79, 34, 60, 92, 11, 41, 114} Apply the following disk scheduling algorithms starting track at 50. i) FCFS ii) SSTF Calculate the total seek time.	8	L3	CO5
		OR			
10	a.	Explain Free Space Management with an example.	6	L2	CO2
	b.	Explain the Access Matrix method of system protection with the domain as objects and its implementation.	6	L2	CO2
	c.	The requested tracks, the order received are {176, 79, 34, 60, 92, 11, 41, 114} Apply the following disk scheduling algorithms starting track at 50. i) Look ii) C – Look Calculate the total seek time.	8	L3	CO3

* * * * *

1 a. A **system call** is a mechanism through which a **user program requests services from the operating system kernel**. System calls provide an interface between **user-level processes and the operating system**.

System calls are generally grouped into the following **types**:

---

1. Process Control System Calls

These system calls are used to **create, manage, and terminate processes**.

Functions

- Create or terminate processes
- Load and execute programs
- Wait for process completion
- Allocate and free memory

Examples

- **fork()** – Creates a new process
- **exec()** – Loads and runs a program
- **exit()** – Terminates a process
- **wait()** – Waits for a child process to finish

Example (C)

```
#include <stdio.h>
#include <unistd.h>

int main() {
   fork();
   printf("Process created\n");
   return 0;
}
```

---

2. File Management System Calls

These calls are used to **create, read, write, and delete files**.

Functions

- Create/delete files
- Open/close files
- Read/write data
- Set file attributes

Examples

- **open()** – Opens a file
- **read()** – Reads data from a file
- **write()** – Writes data to a file
- **close()** – Closes a file

Example

```
int fd = open("data.txt", O_RDONLY);
read(fd, buffer, 100);
close(fd);
```

---

3. Device Management System Calls

These calls allow programs to **interact with hardware devices**.

Functions

- Request and release devices
- Read/write device data
- Get or set device attributes

Examples

- **read()**
- **write()**
- **ioctl()** – Control device parameters

Example: Reading from a printer, disk, or keyboard.

---

4. Information Maintenance System Calls

These calls are used to **get or set system information**.

Functions

- Get system time or date
- Get process or system data
- Set system parameters

Examples

- **getpid()** – Get process ID
- **time()** – Get system time
- **uname()** – Get system information

Example

```
#include <unistd.h>
printf("Process ID: %d", getpid());
```

---

5. Communication System Calls

These calls are used for **communication between processes (IPC – Inter Process Communication)**.

Functions

- Send and receive messages
- Share data between processes

Examples

- **pipe()** – Communication between related processes
- **msgsnd() / msgrcv()** – Message passing
- **socket()** – Network communication

Example:

```
pipe(fd);
write(fd[1], "Hello", 5);
read(fd[0], buffer, 5);
```

---

6. Protection System Calls

These calls control **access permissions and security** of system resources.

Functions

- Control file permissions
- Manage user access

Examples

- **chmod()** – Change file permissions
- **chown()** – Change file ownership
- **setuid()** – Set user ID

Example:

chmod("file.txt", 0777);

✅ **Summary**

Type	Purpose	Examples
Process Control	Manage processes	fork(), exec(), exit()
File Management	Handle files	open(), read(), write(), close()
Device Management	Manage I/O devices	ioctl(), read(), write()
Information Maintenance	Get/set system info	getpid(), time()
Communication	Process communication	pipe(), socket()
Protection	Security and access control	chmod(), chown()

1b. **Definition**

Cloud computing is a technology that delivers computing services such as servers, storage, databases, networking, software, and analytics over the Internet (“the cloud”) instead of using local computers or personal data centers.

Users can access these services anytime, anywhere, on a pay-as-you-use basis.

🔷 **Key Characteristics of Cloud Computing**

1. **On-demand self-service** – Users can provision resources automatically.
2. **Broad network access** – Accessible via internet from various devices.

3. **Resource pooling** – Shared resources for multiple users.
4. **Rapid elasticity** – Scale up or down quickly.
5. **Measured service** – Pay only for what you use.

◈ Cloud Service Models

Cloud computing mainly provides three types of services:

Definition:
Provides virtualized computing resources over the internet.

Services Provided:

- Virtual machines
- Storage
- Networking
- Operating systems

Examples:

- Amazon Web Services (EC2)
- Microsoft Azure
- Google Cloud Platform

Used By: System administrators, IT infrastructure teams.

Definition:
Provides a platform that allows developers to build, test, and deploy applications without managing hardware.

Services Provided:

- Development tools
- Database management
- Middleware
- Operating system

Examples:

- Heroku
- Google App Engine

- Microsoft Azure

Used By: Application developers.

1c. **Dual mode operation** is a security feature of an Operating System that allows the system to run in two different modes:

1. **User Mode**
2. **Kernel Mode (Supervisor Mode)**

This mechanism protects the system from unauthorized access and accidental damage.

🔹 1. User Mode

- Used for running **application programs** (e.g., browser, editor, games).
- Has **limited access** to system resources.
- Cannot directly access hardware or execute privileged instructions.
- If a program crashes, it does **not affect the entire system**.

👉 Example: When you open MS Word or Chrome, it runs in **user mode**.

🔹 2. Kernel Mode

- Used by the **Operating System core**.
- Has **full access** to hardware and system resources.
- Can execute **privileged instructions** (like I/O operations, memory management, process control).

👉 Example: Handling system calls, managing memory, controlling devices.

🔄 Mode Switching

- The system uses a **mode bit** to distinguish between the two modes:
  - 0 → Kernel Mode
  - 1 → User Mode
- When a program requests a service (like file access), it makes a **system call**, and the CPU switches from **user mode to kernel mode**.

- After completing the task, control returns to **user mode**.

🎯 **Why Dual Mode is Important?**

- Provides **security**
- Prevents **unauthorized access**
- Protects **OS and hardware**
- Ensures **system stability**

✅ **In One Line:**

Dual mode operation is a protection mechanism that separates user applications from core OS functions using User Mode and Kernel Mode.

If you want, I can also provide a small diagram explanation for exams.

Generate only text please

Operating System Dual Mode (Brief Explanation)

Dual mode operation is a protection mechanism in an operating system that allows the system to operate in two different modes:

1. **User Mode**
2. **Kernel Mode (Supervisor Mode)**

This mechanism ensures system security and stability.

User Mode:

- Used to run application programs.
- Has limited access to system resources.
- Cannot execute privileged instructions.
- Cannot directly access hardware.

Kernel Mode:

- Used by the operating system.
- Has full access to hardware and memory.
- Can execute privileged instructions.

- Manages processes, memory, and I/O devices.

A **mode bit** is used to distinguish between the two modes:

- 0 → Kernel Mode
- 1 → User Mode

When a program needs an OS service, it makes a system call, and the CPU switches from user mode to kernel mode. After the task is completed, control returns to user mode.

In short, dual mode operation protects the operating system from user programs and ensures secure and controlled execution.

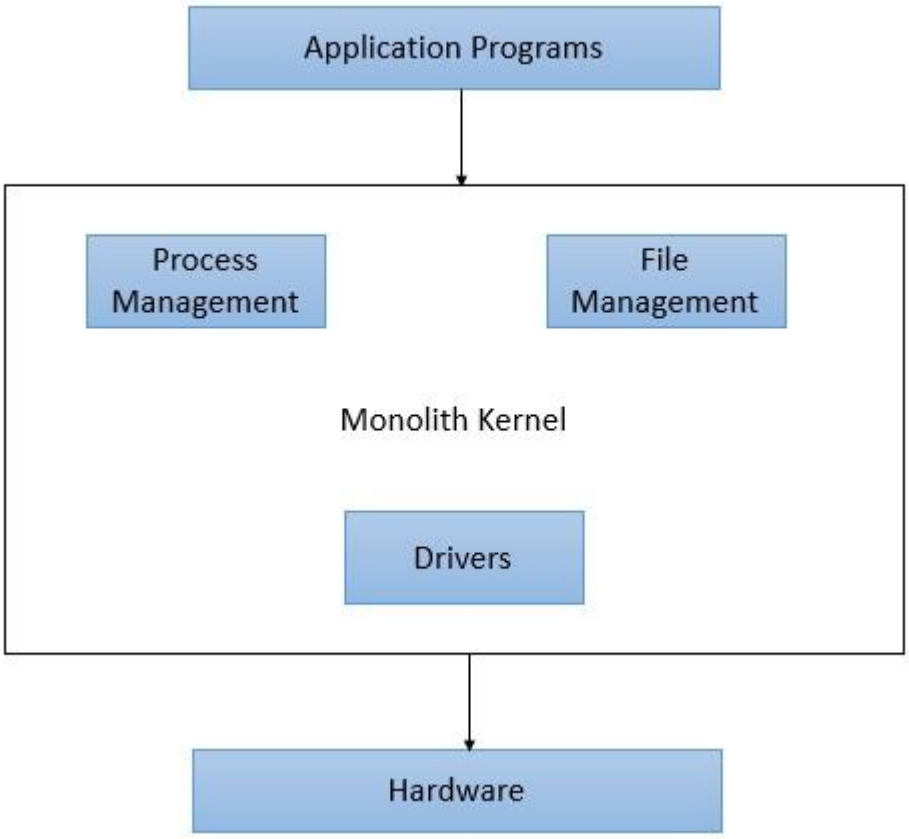
2a.

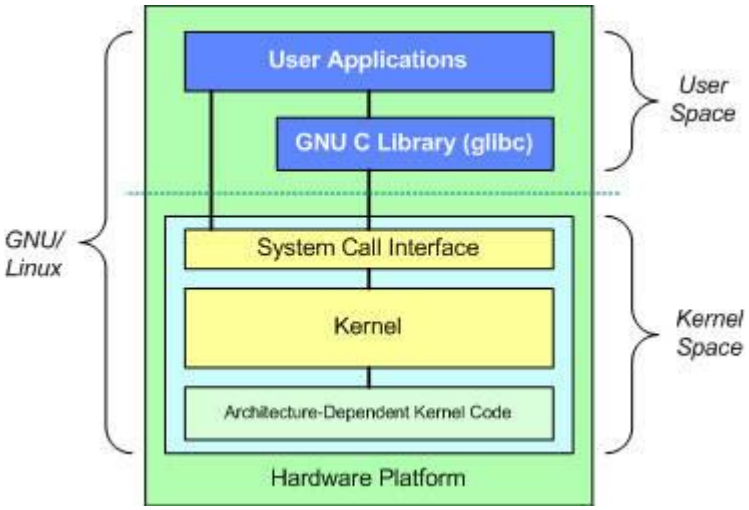
Different Structures of Operating System

Operating System (OS) structure defines how the OS components are organized and how they interact with hardware and users. The main OS structures are:

---

1. Simple / Monolithic Structure





Definition:
All OS services (process management, memory management, file system, device drivers, etc.) are implemented in a single large kernel running in kernel mode.

Features:

- Entire OS works as one program.
- All components can directly communicate.
- High performance but less secure.

Advantages:

- Fast execution.
- Simple design.

Disadvantages:

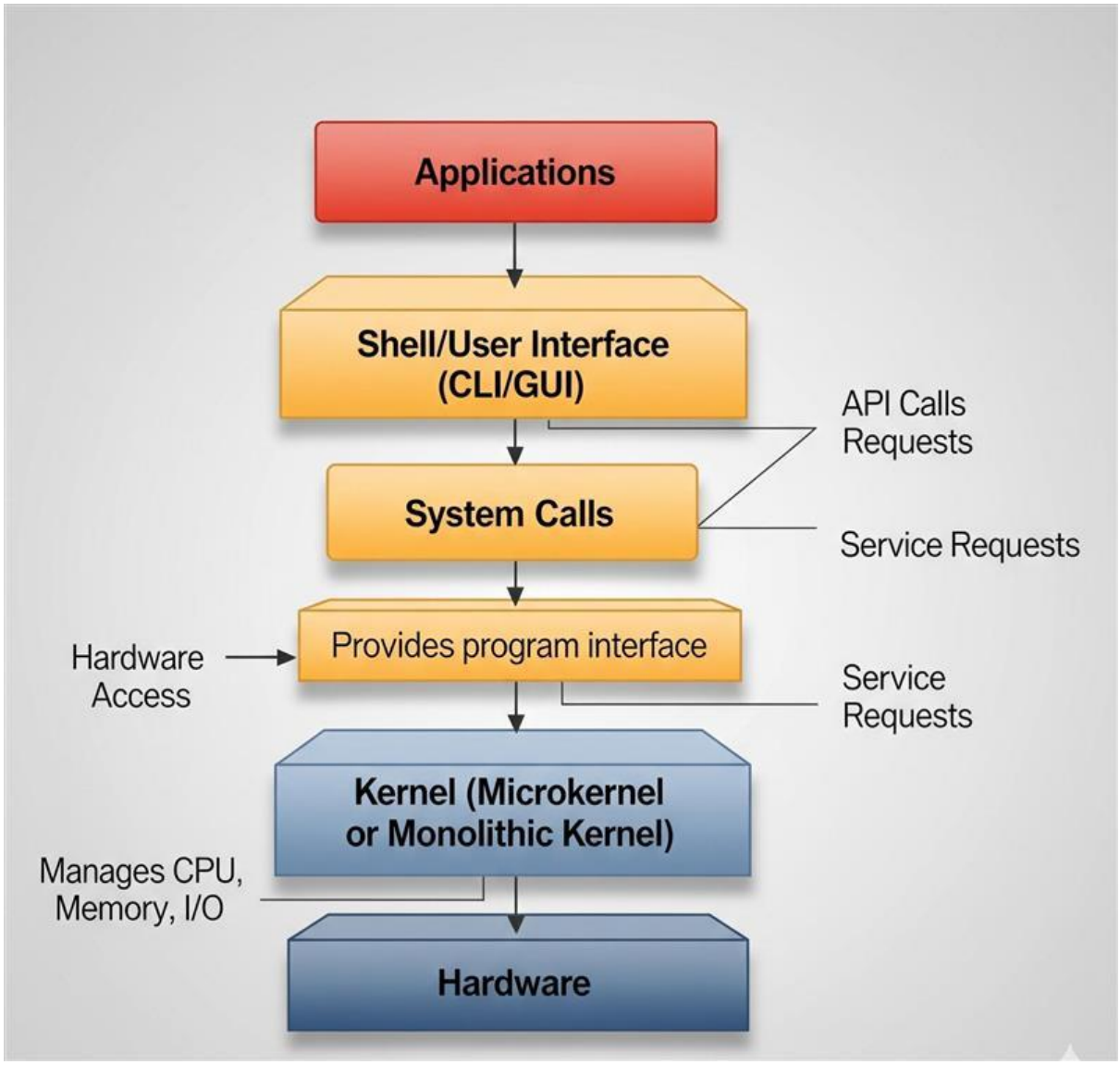
- Difficult to maintain.
- A bug in one part can crash the whole system.

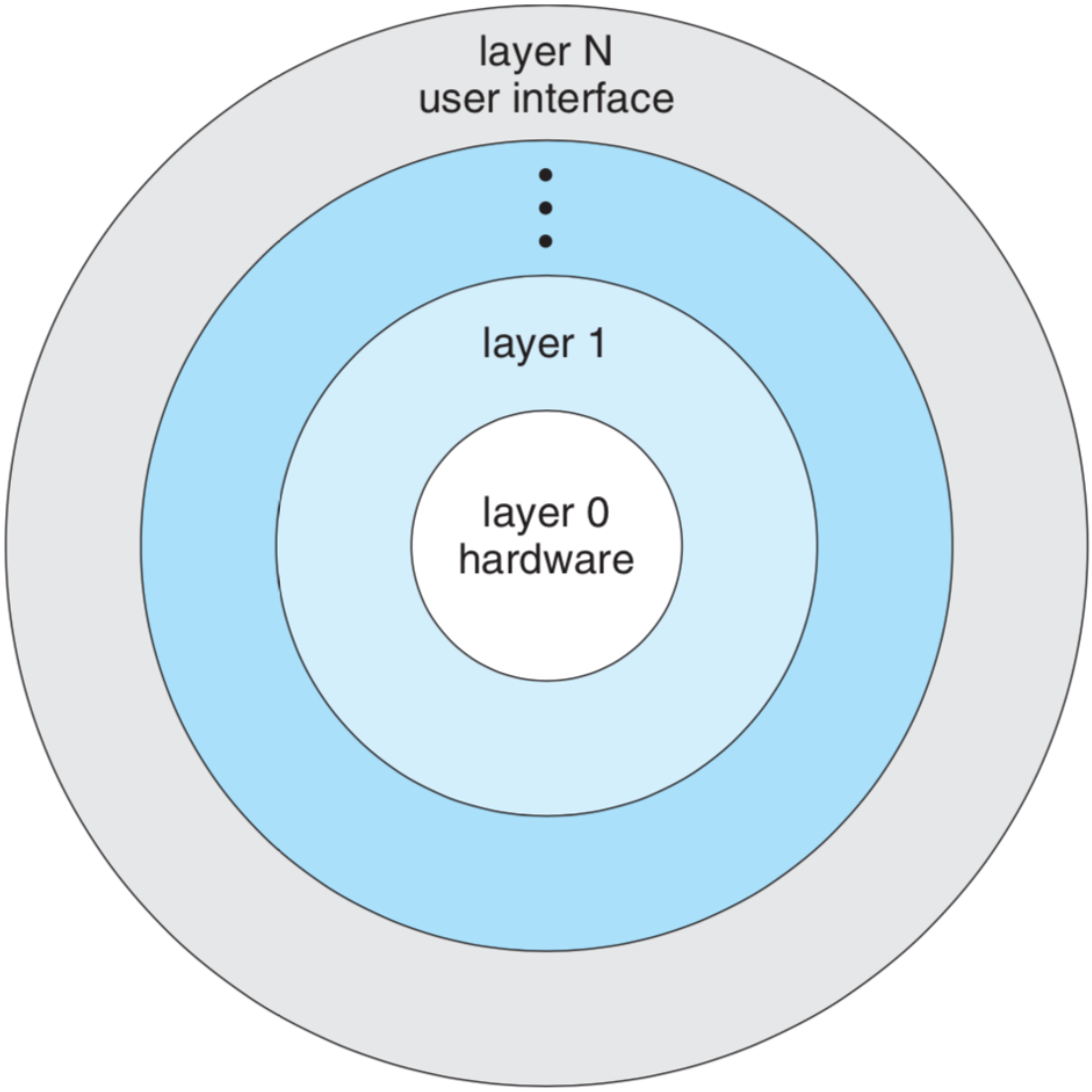
Examples:

- UNIX
- Linux

---

2. Layered Structure


Applications
Shell/User Interface (CLI/GUI)
API Calls Requests
System Calls
Service Requests
Hardware Access
Provides program interface
Service Requests
Kernel (Microkernel or Monolithic Kernel)
Manages CPU, Memory, I/O
Hardware



Definition:
OS is divided into different layers, where each layer performs specific functions and communicates only with adjacent layers.

Features:

- Bottom layer → Hardware
- Top layer → User Interface
- Each layer depends on the lower layer.

Advantages:

- Easy to design and debug.

- Better organization.

Disadvantages:

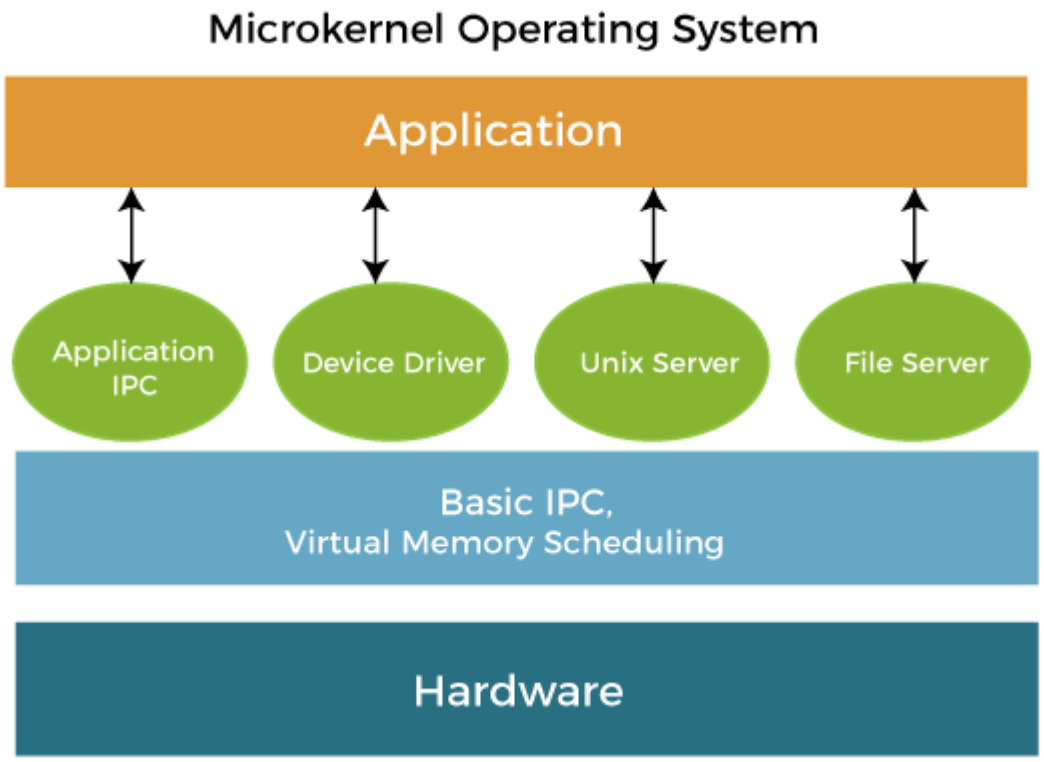
- Slower due to layer communication.
- Difficult to properly define layers.

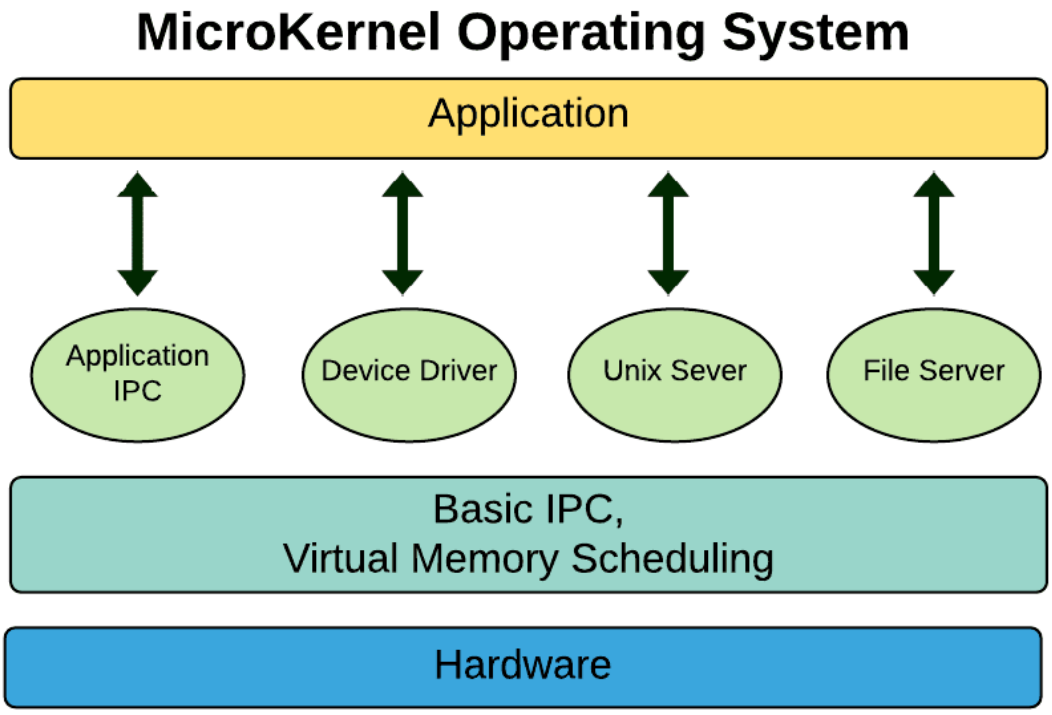
Example:

- THE operating system

---

3. Microkernel Structure





Definition:
Only essential services (like communication and basic memory management) run in kernel mode. Other services run in user mode.

Features:

- Minimal kernel.
- Uses message passing.

Advantages:

- More secure and reliable.
- Easy to extend.

Disadvantages:

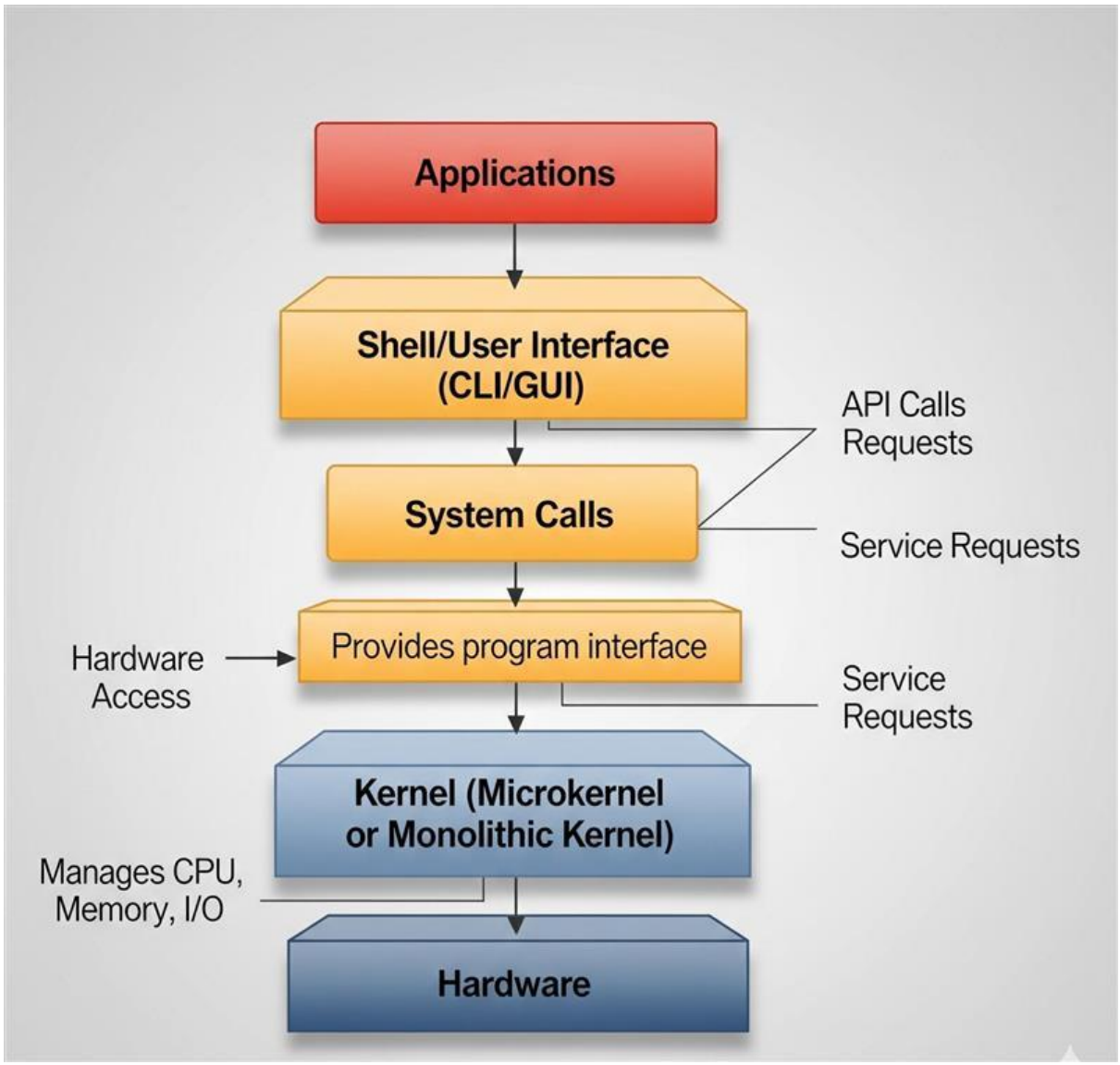
- Performance overhead due to message passing.

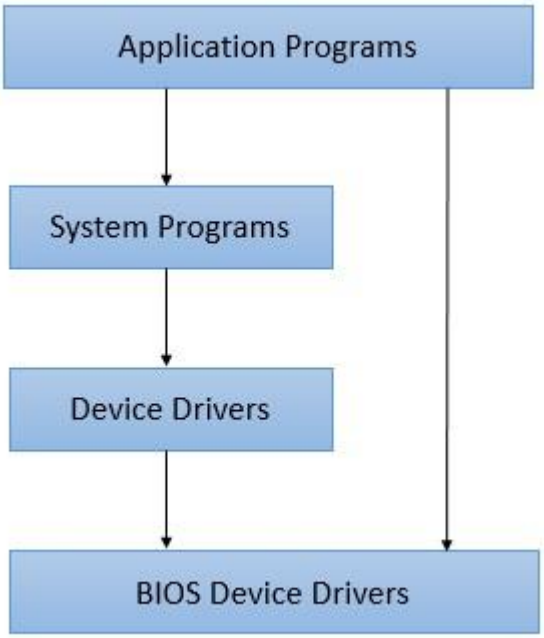
Examples:

- MINIX
- QNX

---

4. Modular Structure


Applications
Shell/User Interface (CLI/GUI)
API Calls Requests
System Calls
Service Requests
Hardware Access
Provides program interface
Service Requests
Kernel (Microkernel or Monolithic Kernel)
Manages CPU, Memory, I/O
Hardware



Definition:
OS uses modules that can be dynamically loaded and unloaded into the kernel.

Features:

- Core kernel + loadable modules.
- Flexible and efficient.

Advantages:

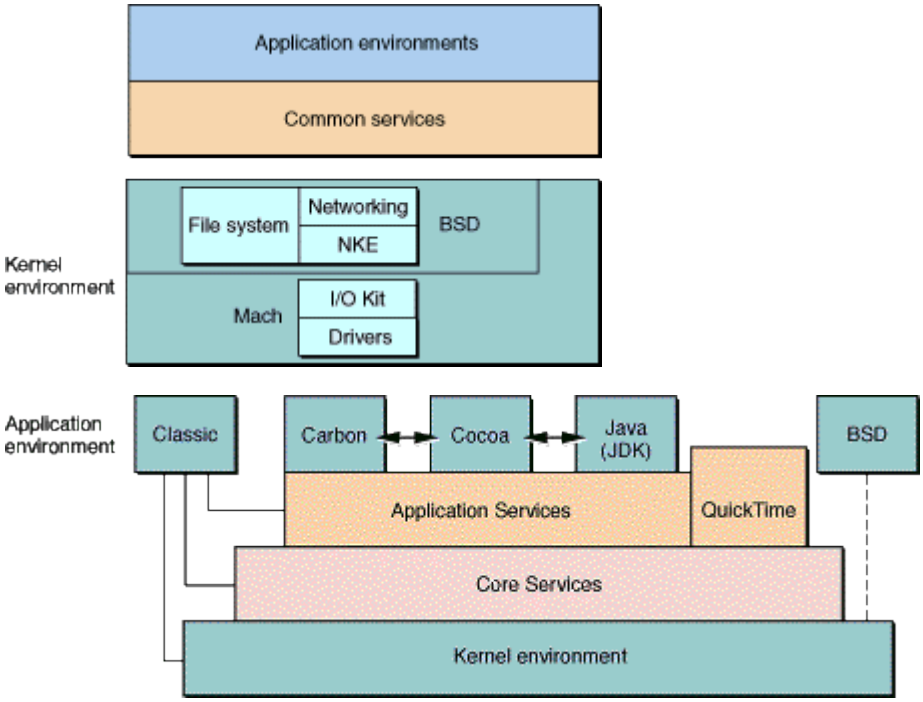
- Easy to add new features.
- Better performance than microkernel.

Example:

- Linux (supports loadable modules)

---

5. Hybrid Structure



Definition:
Combination of monolithic and microkernel structures.

Features:

- Some services in kernel mode.
- Some services in user mode.

Advantages:

- Balanced performance and security.
- More flexible.

Examples:

- Microsoft Windows
- macOS

Summary Table

Structure	Speed	Security	Complexity	Example
Monolithic	High	Low	Simple	UNIX
Layered	Medium	Medium	Moderate	THE OS

Structure	Speed	Security	Complexity	Example
Microkernel	Medium	High	Moderate	MINIX
Modular	High	Medium	Moderate	Linux
Hybrid	High	High	Complex	Windows

2 b. **Definition:**
A Time-Shared Operating System allows multiple users to use the system simultaneously by dividing CPU time into small units called **time slices**.

Working:

- Each process/user gets a small portion of CPU time.
- CPU switches rapidly between processes.
- Provides fast response to users.

Main Goal: Quick response time.

Example: UNIX, Linux.

Definition:
A Multiprogramming Operating System keeps multiple programs in memory and executes them by switching the CPU when one process waits for I/O.

Working:

- Several programs are loaded in memory.
- CPU executes another program when one is waiting.
- Increases CPU utilization.

Main Goal: Maximum CPU utilization.

Example: Batch Operating Systems.

Inter Process Communication (IPC)

Inter Process Communication (IPC) is a mechanism that allows processes to communicate and synchronize with each other in an operating system.

When multiple processes are running, they may need to:

- Share data
- Exchange information
- Coordinate their activities

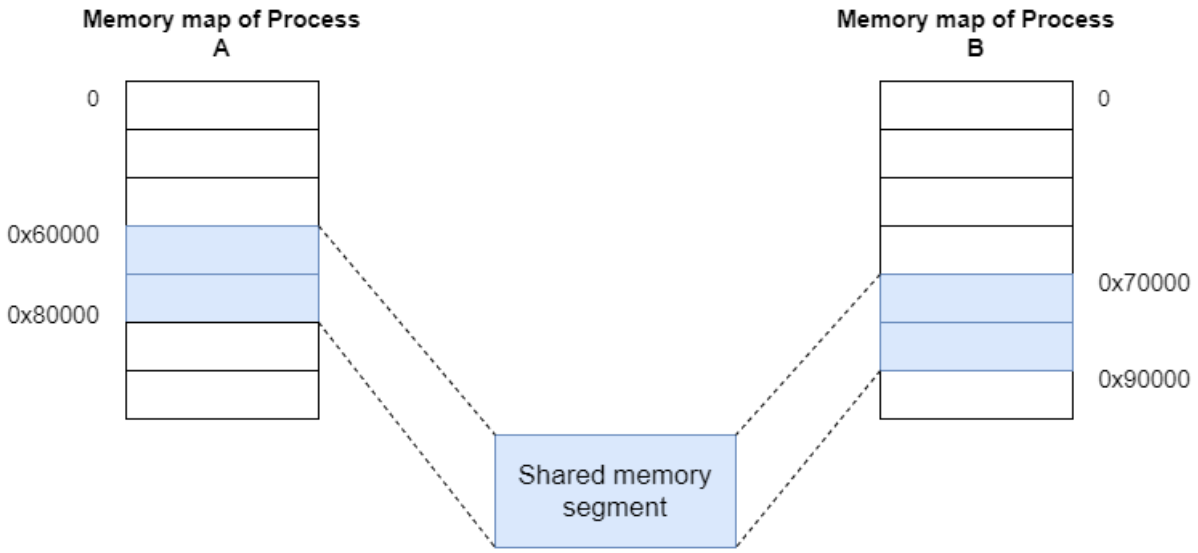
IPC provides methods for this communication.

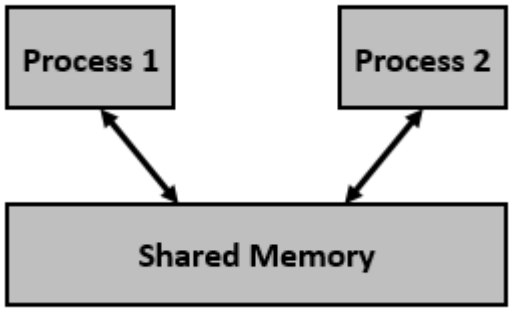
📌 Need for IPC

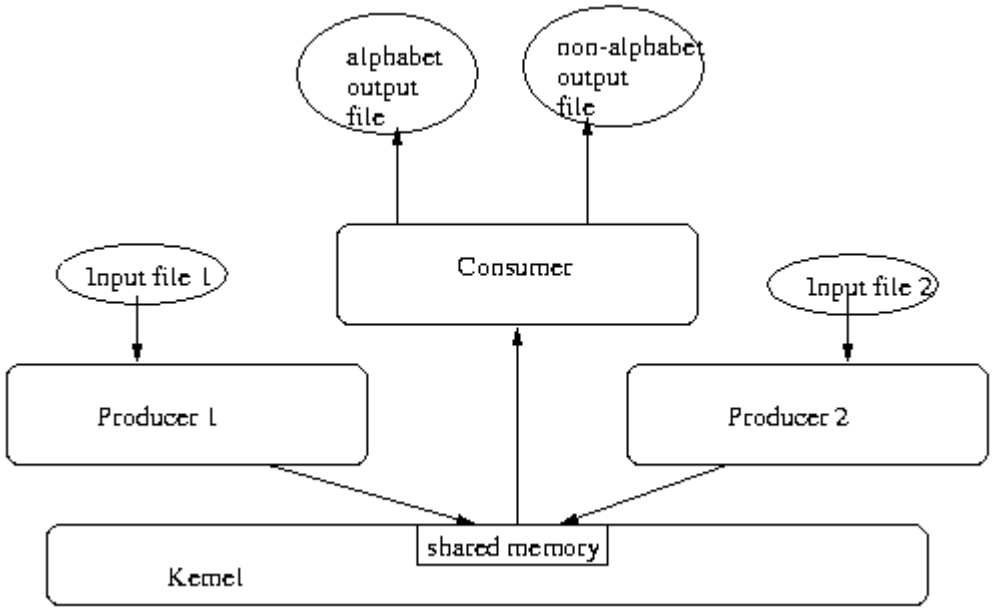
1. Information sharing
2. Speed up computation (parallel processing)
3. Modularity
4. Convenience in system design

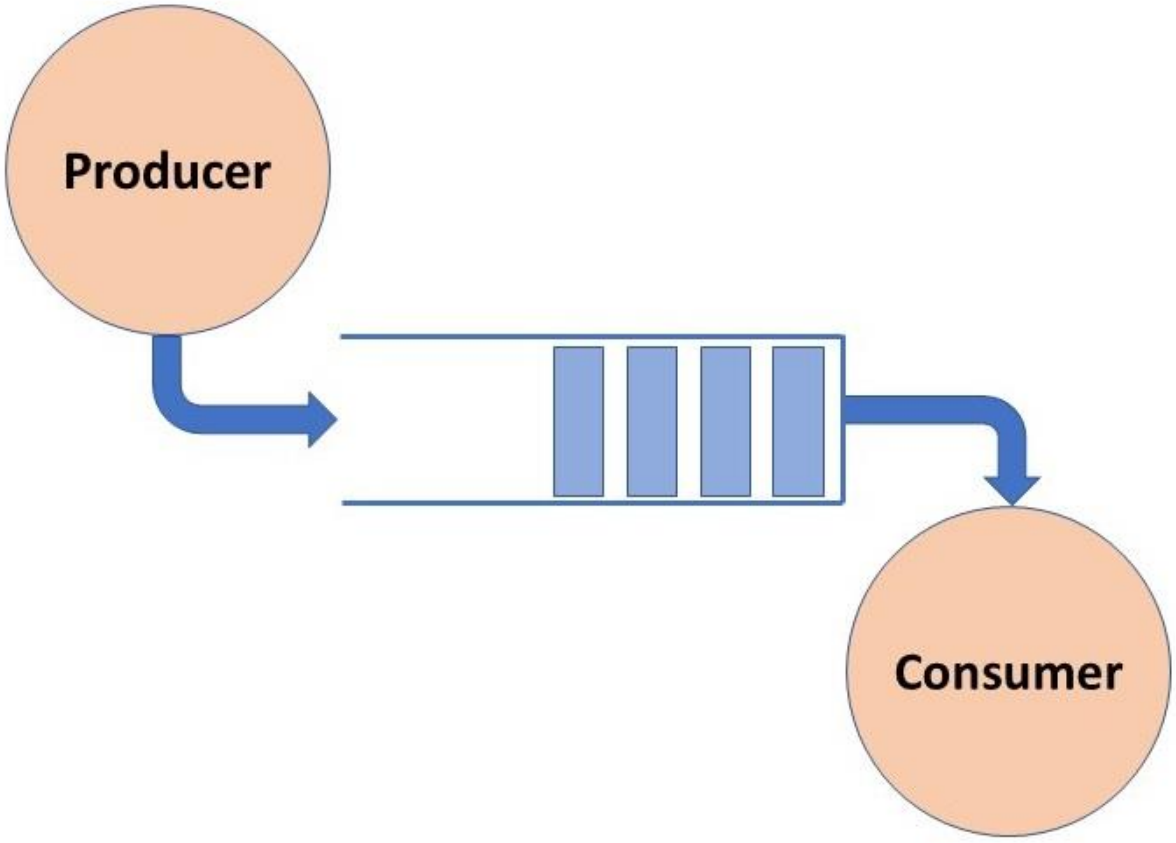
🔧 Types of IPC

1️⃣ Shared Memory







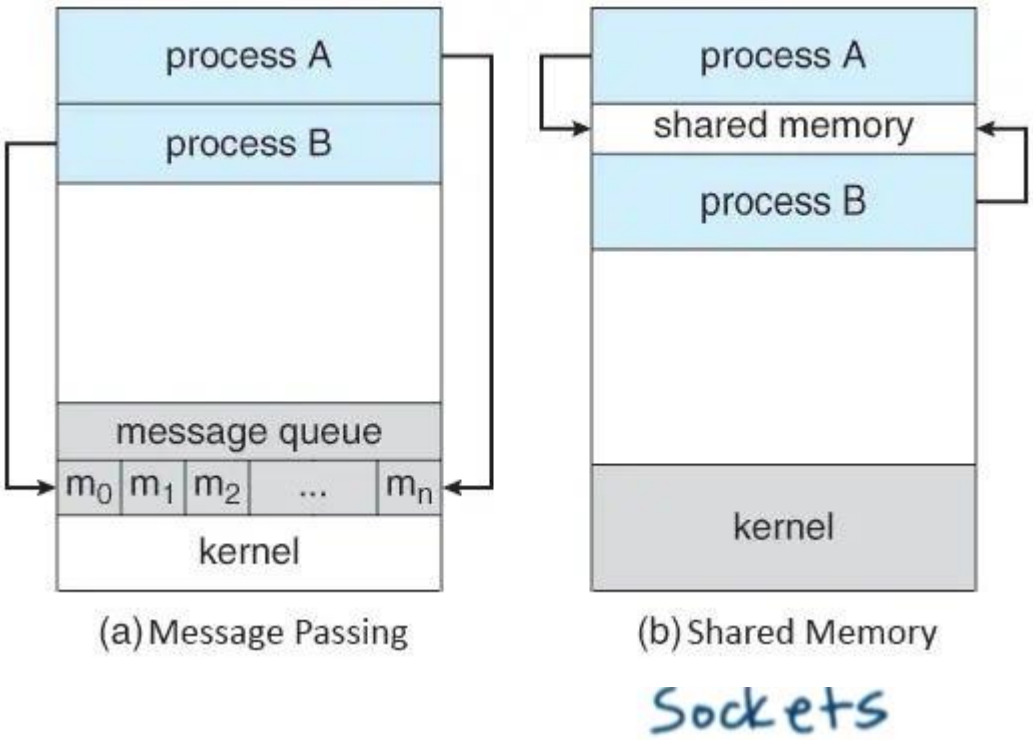


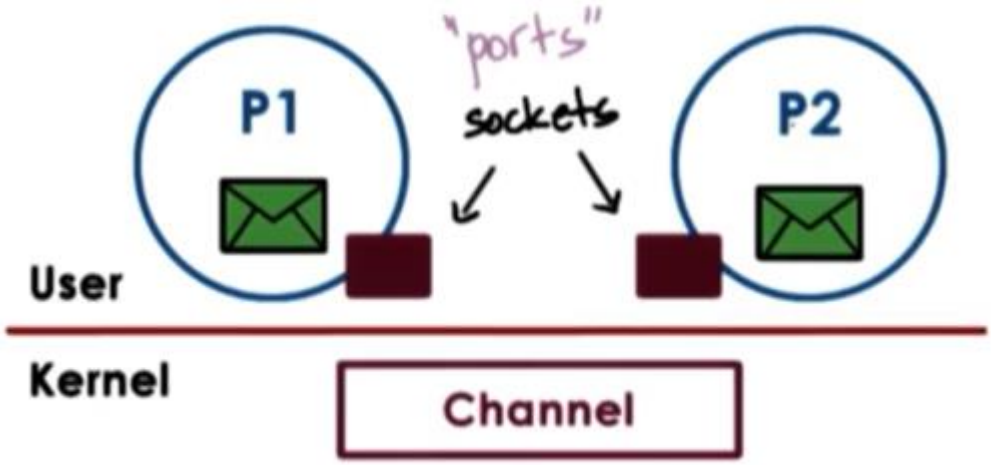
- Two or more processes share a common memory area.
- Processes can read and write to this shared region.
- Fast communication method.

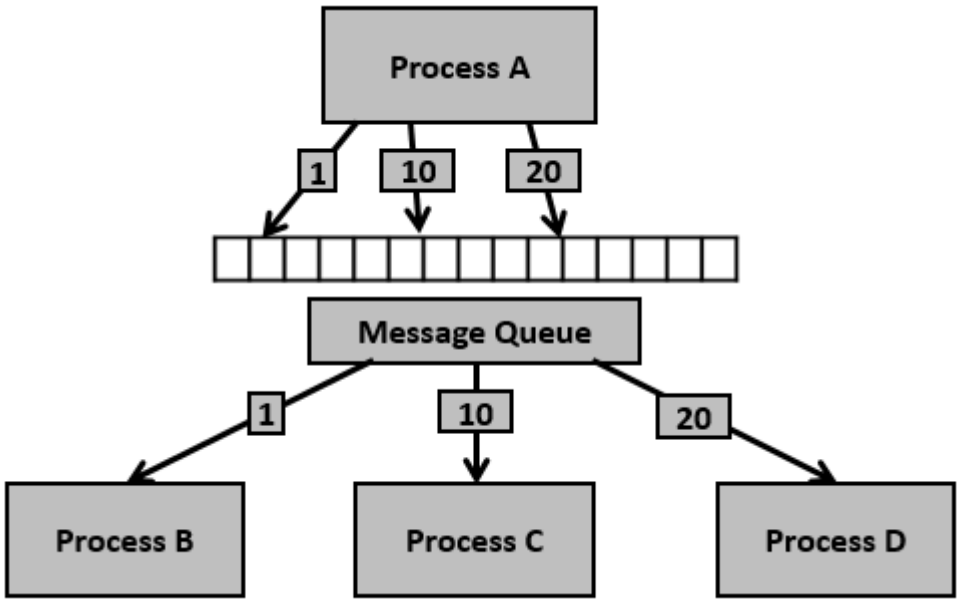
- Requires synchronization (e.g., semaphores) to avoid data inconsistency.

Example: Producer-Consumer problem.

2️⃣ Message Passing







- Processes communicate by sending and receiving messages.
- No shared memory is required.
- Can be:
    - Direct communication (process to process)
    - Indirect communication (via mailbox/message queue)

Example: Client-Server communication.

📌 IPC Mechanisms

- Pipes

- Message Queues
- Shared Memory
- Semaphores
- Sockets

3 b. **Multilevel Queue Scheduling in Operating System**

Multilevel Queue Scheduling is a CPU scheduling algorithm where the ready queue is divided into multiple separate queues based on process type or priority.

Each queue has its own scheduling algorithm.
Processes are permanently assigned to one queue (no movement between queues).

Basic Structure

The ready queue is divided as follows:

```
          READY QUEUE
              |
  ----------------------------------
  |           |           |
System Q    Interactive Q   Batch Q
(Highest)     (Medium)      (Lowest)
```

Scheduling Between Queues

1) Fixed Priority Scheduling

CPU is given to the highest priority queue first.

Priority Order:

System Queue → 1st

Interactive Queue → 2nd

Batch Queue → 3rd

If System queue has processes, others must wait.

---

2) Time Slice Scheduling Between Queues

Each queue gets a fixed CPU percentage.

Example:

Interactive Queue → 80% CPU time

Batch Queue → 20% CPU time

Scheduling Within Each Queue

Different algorithms can be used inside each queue:

System Queue → FCFS

Interactive Queue → Round Robin

Batch Queue → FCFS

Advantages

- Simple and easy to implement
- Good for systems with different process categories
- Faster response for high-priority processes

Disadvantages

- Starvation of low-priority queues
- No flexibility (process cannot move to another queue)

Conclusion

Multilevel Queue Scheduling divides processes into separate queues based on priority or type. Each queue follows its own scheduling algorithm, and CPU is allocated either by fixed priority or time sharing between queues.

3 c. **Given:**

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	9
P2	1	4
P3	2	9

Scheduling Algorithm: **Shortest Remaining Time First (SRTF)**
(SRTF is preemptive version of SJF)

Step 1: Gantt Chart Construction

At time 0 → Only **P1** available → Run P1

At time 1 → **P2** arrives (BT=4 < remaining of P1=8) → Preempt P1 → Run P2

At time 2 → **P3** arrives (BT=9, P2 remaining=3) → Continue P2

P2 finishes at time 5.

Now compare:

- P1 remaining = 8
- P3 = 9
  → Run P1

P1 runs from 5 to 13 and finishes.

Then P3 runs from 13 to 22.

Gantt Chart

0 1 5 13 22

| P1 | P2 | P1 | P3 |

Step 2: Completion Time (CT)

Process	CT
P1	13

Process	CT
P2	5
P3	22

Step 3: Turnaround Time (TAT)

[TAT = CT - AT]

Process	TAT
P1	13 − 0 = 13
P2	5 − 1 = 4
P3	22 − 2 = 20

Step 4: Waiting Time (WT)

[WT = TAT - BT]

Process	WT
P1	13 − 9 = 4
P2	4 − 4 = 0
P3	20 − 9 = 11

Step 5: Average Times

Average Waiting Time

[(4 + 0 + 11) / 3]

Average Turnaround Time

[(13 + 4 + 20) / 3]

4a. **User-Level Threads vs Kernel-Level Threads (Brief Comparison)**

Feature	**User-Level Threads**	**Kernel-Level Threads**
Managed by	User thread library	Operating System kernel
Kernel awareness	Kernel not aware	Kernel fully aware
Creation & switching	Fast (no system call)	Slower (system call required)
Blocking call	Blocks entire process	Only that thread blocks
Multi-core support	No true parallelism	Supports true parallelism
Overhead	Low	High

- **User-level threads** are faster but limited in parallel execution.
- **Kernel-level threads** support better concurrency but have higher overhead.

4b. ◈ User-Level Threads vs Kernel-Level Threads (Brief Comparison)

Feature	**User-Level Threads**	**Kernel-Level Threads**
Managed by	**User thread library**	**Operating System kernel**
Kernel awareness	**Kernel not aware**	**Kernel fully aware**
Creation & switching	**Fast (no system call)**	**Slower (system call required)**
Blocking call	**Blocks entire process**	**Only that thread blocks**
Multi-core support	**No true parallelism**	**Supports true parallelism**
Overhead	**Low**	**High**

◈ Conclusion

- **User-level threads are faster but limited in parallel execution.**
- **Kernel-level threads support better concurrency but have higher overhead.**

5a. **Critical Section Problem**

A **critical section** is a part of a program where shared resources (like variables, files, buffers) are accessed.

When multiple processes/threads access shared data simultaneously, it may cause **race condition**.

Structure of a Process

```
do {
  Entry Section
  Critical Section
  Exit Section
  Remainder Section
} while (true);
```

Requirements of Critical Section Problem

1. **Mutual Exclusion**
   Only one process can enter the critical section at a time.
2. **Progress**
   If no process is in the critical section, selection of the next process should not be postponed indefinitely.
3. **Bounded Waiting**
   A process should not wait forever to enter the critical section.

Producer–Consumer Problem (Bounded Buffer Problem)

Two processes:

- **Producer** → Produces items and puts them into buffer
- **Consumer** → Consumes items from buffer

Assume:

- Buffer size = N
- mutex = 1 (binary semaphore)
- empty = N
- full = 0

Producer Algorithm

```
do {
  produce item

  wait(empty)
  wait(mutex)

  add item to buffer

  signal(mutex)
  signal(full)

} while (true);
```

Consumer Algorithm

```
do {
  wait(full)
  wait(mutex)

  remove item from buffer

  signal(mutex)
  signal(empty)

  consume item

} while (true);
```

Explanation of Semaphores

- mutex → Ensures mutual exclusion
- empty → Counts empty buffer slots
- full → Counts filled buffer slots

6a. **Dining Philosophers Problem (Brief)**

The Dining Philosophers Problem is a classical synchronization problem in Operating Systems, proposed by Edsger W. Dijkstra.

Problem Statement:

- Five philosophers are sitting around a circular table.
- There is one fork between each pair of philosophers (total 5 forks).
- Each philosopher alternates between thinking and eating.
- To eat, a philosopher needs two forks (left and right).

Problem:
If all philosophers pick up their left fork at the same time, each will be holding one fork and waiting for the other. This leads to:

1. Deadlock – All philosophers wait forever.
2. Starvation – Some philosophers may never get a chance to eat.
3. Resource sharing issue – Forks are limited shared resources.

Goal:
Design a solution that avoids deadlock and starvation while allowing proper synchronization.

Common Solutions:

- Allow only four philosophers to try eating at a time.
- Use a waiter (monitor) to control access to forks.
- Use semaphores or mutex locks.
- Impose an order for picking forks (odd-even method).

6b. **Resource Allocation Graph (RAG)**

A **Resource Allocation Graph (RAG)** is a directed graph used in Operating Systems to represent the allocation of resources to processes and to detect **deadlocks**.

◈ Components of RAG

1. **Process (Pi)** – Represented by a **circle (○)**
2. **Resource (Rj)** – Represented by a **square (□)**
3. **Request Edge** – Arrow from **Process → Resource**
    - Means the process is requesting the resource.
4. **Allocation Edge** – Arrow from **Resource → Process**
    - Means the resource is allocated to the process.

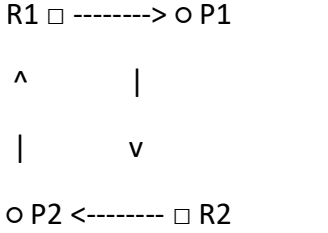
◈ Diagram Example

Consider:

- 2 Processes: P1, P2
- 2 Resources: R1, R2

Situation:

- P1 is holding R1 and requesting R2
- P2 is holding R2 and requesting R1



Explanation:

- R1 → P1 (R1 allocated to P1)
- P1 → R2 (P1 requesting R2)
- R2 → P2 (R2 allocated to P2)
- P2 → R1 (P2 requesting R1)

This forms a **cycle**:
P1 → R2 → P2 → R1 → P1

🔷 **Important Points**

- If there is **no cycle**, there is **no deadlock**.
- If there is a **cycle**:
    - For **single instance resources** → Deadlock exists.
    - For **multiple instance resources** → Deadlock may or may not exist.

7a Explain paging and Look aside Buffer

Ans: **Paging**

Paging is a memory management technique in Operating Systems that eliminates external fragmentation.

🔷 **Concept:**

- Physical memory is divided into fixed-size blocks called **Frames**.
- Logical memory (process memory) is divided into same-size blocks called **Pages**.
- Pages are loaded into any available frame in physical memory.
- The mapping between pages and frames is stored in a **Page Table**.

🔷 **Address Structure:**

Logical Address =
Page Number (p) + Offset (d)

- Page number → used to find frame number in page table
- Offset → location within the frame

🔷 **Advantages:**

- No external fragmentation
- Efficient memory utilization

🔷 **Disadvantage:**

- Page table access increases memory access time

Look Aside Buffer (TLB – Translation Lookaside Buffer)

A **Look Aside Buffer**, commonly called **TLB**, is a special high-speed cache memory used to reduce paging overhead.

🔷 **Need:**

In paging, to access data:

1. Access page table
2. Access actual memory

This requires **two memory accesses**.

🔷 **Solution:**

TLB stores recently used **page number → frame number** mappings.

🔷 **Working:**

- CPU first checks TLB.
    - If entry found → **TLB Hit** → Directly get frame number.
    - If not found → **TLB Miss** → Access page table, then update TLB.

🔷 **Advantage:**

- Reduces effective memory access time.
- Improves system performance.

7b. Discus contiguous memory allocation with a neat diagram.

Ans: Contiguous Memory Allocation

Contiguous Memory Allocation is a memory management technique in which each process is allocated a single continuous block of main memory.

🔷 **Basic Idea**

- **Memory is divided into Operating System area and User processes area.**

- **Each process occupies one contiguous partition.**
- **When a process finishes, its memory is freed and can be reused.**

🔷 Types of Contiguous Allocation

1) Fixed Partition Allocation

- **Memory is divided into fixed-size partitions.**
- **Each partition holds one process.**
- **Simple but leads to internal fragmentation.**

Diagram:

```
+--------------------+
|       OS           |
+--------------------+
|       P1           |
+--------------------+
|       P2           |
+--------------------+
|     Free Space     |
+--------------------+
|       P3           |
+--------------------+
```

---

2) Variable Partition Allocation

- **Partitions are created dynamically based on process size.**
- **Better memory utilization.**
- **Causes external fragmentation.**

Diagram:

```
+--------------------+
|       OS           |
```

+----------------------+

| P1 (20 KB) |

+----------------------+

| P2 (35 KB) |

+----------------------+

| Free (10 KB) |

+----------------------+

| P3 (25 KB) |

+----------------------+

| Free (15 KB) |

+----------------------+

◈ Placement Strategies

In variable partitioning, free space is allocated using:

1. **First Fit – Allocate first sufficient hole.**
2. **Best Fit – Allocate smallest sufficient hole.**
3. **Worst Fit – Allocate largest hole.**

◈ Advantages

- **Simple to implement.**
- **Easy memory protection.**

◈ Disadvantages

- **Internal fragmentation (fixed partition).**
- **External fragmentation (variable partition).**
- **Compaction may be required.**

◈ Conclusion

Contiguous memory allocation is simple and easy to manage, but fragmentation problems limit its efficiency in modern systems.

9a Explain in detail directory and disc structure in detail

Ans: **Directory Structure in Operating System**

A **directory** is a special file that stores information about files and other directories. It helps in organizing files systematically.

🔹 **Functions of a Directory**

- File naming
- File organization
- File searching
- File sharing
- File protection

Each directory entry typically contains:

- File name
- File type
- Location (pointer to disk blocks)
- Size
- Access permissions

Types of Directory Structures

1) Single-Level Directory

- Only one directory for all users.
- All files are stored in the same directory.

Root

├— file1

├— file2

├— file3

Advantages:

- Simple to implement

Disadvantages:

- File name conflicts
- Not suitable for multi-user systems

---

2) Two-Level Directory

- Separate directory for each user.
- Each user has their own files.

Root

├── User1

│ ├── fileA

│ └── fileB

├── User2

├── fileC

└── fileD

Advantage:

- No name conflict between users

Disadvantage:

- No file sharing between users

---

3) Tree-Structured Directory

- Directories can contain subdirectories.
- Forms a hierarchical structure.

Root

├── CSE

│ ├── Notes

│ └── Programs

└── ECE

└── Labs

Advantages:

- Efficient organization
- Easy file grouping

---

4) Acyclic Graph Directory

- Allows sharing of files and subdirectories.
- No cycles allowed.

Advantage:

- File sharing possible

Disadvantage:

- More complex management

---

5) General Graph Directory

- Allows cycles.
- Complex to manage.
- Risk of infinite loops.

Disk Structure in Operating System

A **disk** is a secondary storage device used to store data permanently.

🔷 Physical Structure of Disk

A disk consists of:

1. **Platters** – Circular magnetic disks
2. **Tracks** – Concentric circles on a platter
3. **Sectors** – Subdivisions of tracks
4. **Cylinder** – Collection of tracks at same position on all platters

5. **Read/Write Head** – Reads or writes data

◈ Disk Layout Diagram (Conceptual)

```
 ----------  ← Track
/          \
|  Sector   |  ← Sector
\          /
 ----------
```

Stack of tracks at same radius = Cylinder

◈ Logical Disk Structure

Before using a disk, it must be:

1. **Low-level formatting** – Divides disk into sectors.
2. **Partitioning** – Divides disk into logical disks.
3. **Logical formatting** – Creates file system structures.

◈ Disk Access Time

Disk access time =
Seek Time + Rotational Latency + Transfer Time

- Seek Time → Move head to correct track
- Rotational Latency → Wait for sector to rotate
- Transfer Time → Actual data transfer

Conclusion

- Directory structure organizes files efficiently.
- Disk structure explains how data is physically and logically stored.
- Both are essential for proper file management in Operating Systems.

10 a. Explain free space management in detail.

Free Space Management in Operating System

Free space management is a method used by the file system to keep track of **unused disk blocks**, so that they can be allocated to files when required.

When a file is created, the OS allocates free blocks.
When a file is deleted, those blocks are returned to the free space list.

🔷 Need for Free Space Management

- To know which disk blocks are free.
- To allocate space efficiently.
- To avoid overwriting existing data.
- To improve disk performance.

🔷 Methods of Free Space Management

1) Bit Map (Bit Vector) Method

- Each disk block is represented by **1 bit**.
- Bit value:
    - 0 → Free block
    - 1 → Allocated block

Example:

If disk has 8 blocks:

Block No: 0 1 2 3 4 5 6 7

Bit Map : 1 0 0 1 1 0 1 0

Free blocks → 1, 2, 5, 7

Advantages:

- Easy to find contiguous free blocks.
- Simple implementation.

Disadvantages:

- Requires extra memory for large disks.

---

2) Linked List Method

- All free blocks are linked together.
- Each free block contains a pointer to the next free block.
- OS maintains a pointer to the first free block.

Representation:

Free Block → 5 → 9 → 13 → 20 → NULL

Advantages:

- No extra memory needed for bitmap.
- Simple to manage.

Disadvantages:

- Slow to find contiguous free blocks.
- Must traverse entire list.

---

3) Grouping Method

- Similar to linked list.
- First free block stores addresses of several free blocks.
- Last entry points to another block that contains more free block addresses.

Advantage:

- Faster than simple linked list.

---

4) Counting Method

- Used when many contiguous blocks are free.
- Instead of storing each block number, store:
    - Starting block address
    - Number of free blocks

Example:

(100, 5)

Means blocks 100, 101, 102, 103, 104 are free.

Advantages:

- Efficient when free space is in large contiguous chunks.
- Requires less storage.

Disadvantages:

- Less effective if free blocks are scattered.

🔷 Comparison

Method	Space Efficient	Fast Search	Suitable For
Bit Map	Moderate	Yes	Large disks
Linked List	Good	No	Small systems
Grouping	Better	Faster	Medium systems
Counting	Very Good	Yes	Contiguous free space
