A1.3.1 — The Role of Operating Systems

1. Introduction

An **Operating System (OS)** is the **core software** that manages computer hardware and software resources. It provides a **bridge between the user and the hardware**, ensuring that all components of the computer work together efficiently, securely, and reliably.

Without an operating system, users would have to interact directly with the hardware using complex machine code instructions — something impractical and inefficient.
The OS hides this complexity by providing a **user-friendly interface** and **standardized services** for applications.

2. The OS as an Intermediary

User	↔	**Operating System**	↔	**Hardware**
Issues commands (e.g. open file, print document)		Translates user requests into low-level hardware operations		Executes actual machine instructions

The OS **abstracts hardware details** and presents a **simplified environment** for users and software developers.
For example, when you save a file, the OS handles the actual disk operations — you only see the saved document in a folder.

3. Key Roles of the Operating System

a. CPU Management

- The OS decides **which process uses the CPU and for how long**.
- It manages **process scheduling**, ensuring fair and efficient CPU time distribution among running applications.
- It handles **context switching**, allowing multiple programs to appear as if they run simultaneously (multitasking).

Example:
When you listen to music while browsing the web, the OS allocates short time slices to each process so both run smoothly.

b. Memory Management

- The OS **allocates memory** to programs when they start and **frees it** when they end.
- Prevents one process from accessing another's memory (isolation and protection).
- Manages **virtual memory**, allowing the system to use part of the hard drive as temporary RAM when physical memory is full.

Example:
When several browser tabs are open, the OS manages their memory separately, preventing one tab from crashing others.

c. Storage Management

- The OS organizes data using a **file system** (e.g. NTFS, HFS+, ext4).
- Handles **file creation, deletion, reading, and writing**.
- Keeps **directory structures** and metadata (like file size, creation date).
- Ensures reliable **data retrieval and long-term storage**.

Example:
When you save a Word document, the OS decides where on the disk to store it and tracks its location for later access.

d. Device Management

- The OS manages all **input and output devices** using **device drivers**.

- It coordinates communication between hardware (printers, keyboards, USBs) and applications.
- Uses techniques like **spooling**, **buffering**, and **interrupt handling** to manage data flow.

Example:
When you print a file, the OS spools the print job — queuing it until the printer is ready.

4. Simplifying User Interaction

Operating systems provide **user interfaces** that allow humans to interact with computers without needing to understand binary or machine-level code.

Interface Type	Description	Example
Command-Line Interface (CLI)	User types commands	Linux terminal, Windows PowerShell
Graphical User Interface (GUI)	Uses icons, menus, and windows	Windows Desktop, macOS Finder
Touch Interface	Input via gestures and taps	Android, iOS

Example:
Instead of typing commands to move files, users can drag and drop them using GUI tools.

---

5. Abstraction of Hardware Complexity

Concept:
Hardware components like CPUs, memory chips, and I/O controllers are complex.

The OS provides **high-level APIs (Application Programming Interfaces)** that abstract this complexity.

For example:

- Developers can use functions like `readFile()` without knowing the hardware-level disk operations.
- The OS translates these calls into low-level hardware commands automatically.

This **abstraction layer**:

- Makes systems easier to program and use.
- Ensures compatibility across hardware types.

---

6. Examples of Modern Operating Systems

Device Type	Popular OS Examples	Notes
Personal Computers	Microsoft Windows, Apple macOS, Linux	Multi-purpose, GUI-based
Mobile Devices	Android, iOS	Touch-optimized, power-efficient
Servers	Linux (Ubuntu Server, Red Hat), Windows Server	Designed for performance and scalability
Embedded Systems	RTOS (Real-Time Operating System), VxWorks	Used in IoT devices, cars, medical instruments

Example:
Android and iOS manage sensors, memory, and background apps efficiently to balance battery life and performance.

7. Summary Table

Component Managed by OS	Role	Example Task
CPU	Scheduling and process control	Running multiple apps
Memory	Allocation and protection	Running many browser tabs
Storage	File system organization	Saving a video file
Devices	Communication via drivers	Printing or scanning
Interface	User interaction layer	Using touch gestures

8. Importance of the Operating System

1. **Resource Management:** Coordinates CPU, memory, and devices efficiently.
2. **User Convenience:** Offers intuitive interfaces (GUI, CLI).
3. **Security:** Protects data and ensures authorized access only.
4. **Stability:** Handles crashes and manages recovery automatically.
5. **Portability:** Provides hardware abstraction, allowing software to run across devices.

Questions for Revision

1. **Define** the term "operating system."
2. **Explain** how the OS acts as an intermediary between users and hardware.
3. **Describe** two reasons why abstraction is important in an operating system.
4. **Outline** the four main resource management roles of the OS.
5. **Explain** the function of CPU management in multitasking.
6. **Describe** how virtual memory extends system capability.
7. **Identify** the main file system responsibilities of an OS.
8. **Distinguish** between device drivers and device controllers.
9. **State** one advantage of GUI over CLI in modern OS design.
10. **Compare** mobile and desktop operating systems in terms of design priorities.

A1.3.2 Functions of an Operating System

The **Operating System (OS)** is the core software that manages the hardware and software resources of a computer system. It acts as a bridge between the user and the computer hardware, ensuring that programs run smoothly, resources are used efficiently, and data is secure.

---

1. Memory Management

• **Definition:** Memory management involves controlling and coordinating how memory (RAM) is allocated and used by different programs.
→ The OS ensures each process gets enough memory to run without interfering with others.

• **Process:**

1. The OS loads the program from secondary storage (like an SSD or HDD) into RAM.
2. It divides memory into sections — assigning space for code, data, and stack.
3. As programs run, the OS reallocates and frees memory dynamically.

• **Example:**
When you open Google Chrome, the OS allocates memory to run it. If you open multiple tabs, each tab is given a separate memory space to prevent one from crashing the others.

• **Virtual Memory:**
When RAM is full, the OS uses part of the hard drive as **virtual memory** (a “swap file”).
→ This keeps the computer running, though slower, as storage access is slower than RAM.

Memory Type	Location	Speed	Purpose
RAM	Main memory	Very fast	Active programs and data
Virtual memory	Hard drive (swap file)	Slow	Extends usable memory when RAM is full

Analogy: Think of memory as a bookshelf — each book (process) has its own space. The OS ensures books don’t overlap or mix up.

---

2. File Management

• **Definition:** The OS organizes, stores, retrieves, and manipulates data on storage devices using a **file system**.

• **Functions:**

1. Creates, reads, writes, renames, copies, and deletes files.
2. Maintains a **hierarchical structure** (folders and subfolders).
3. Ensures unique file naming and manages extensions like `.docx`, `.jpg`, `.exe`.
4. Controls access permissions (who can read, write, or execute files).
5. Performs **defragmentation** to reorganize scattered data on disks for better performance.

Operation	**Example in OS**
Create/Delete	Creating a new Word file or deleting old photos
Rename	Changing “draft.txt” → “final.txt”
Move/Copy	Moving a file to another folder
Defragmentation	Reorganizing fragmented data on HDD for faster access

• **Example:**
In Windows, File Explorer allows you to view and organize files in folders; macOS uses Finder to do the same.

---

3. Device Management

• **Definition:** The OS manages all connected hardware devices printers, disks, keyboards, and monitors through **device drivers**.

• **Device Drivers:** Specialized software that allows the OS to communicate with hardware devices.
→ Each hardware type requires its own driver.

• **Key Processes:**

1. **Buffering** – temporarily storing data before sending or receiving to handle speed differences.
2. **Caching** – storing frequently used data for quick access.
3. **Spooling** – queuing tasks (like print jobs) for devices that can handle one task at a time.
4. **Plug and Play (PnP)** – automatically detects and configures new devices.

Technique	Purpose	Example
Buffering	Smooth data flow between devices	Watching YouTube — video data buffered
Caching	Faster access to frequent data	Browser cache storing webpage data
Spooling	Managing queued tasks	Printing multiple documents
Plug and Play	Auto hardware setup	Connecting a new USB mouse

• **Example:**
When you connect a printer, Windows automatically installs its driver. The OS manages print requests through **spooling**, ensuring jobs are printed in order.

---

4. Scheduling

• **Definition:** Scheduling determines **which process runs at a given time** on the CPU.
→ The goal is to maximize CPU efficiency and ensure fairness among processes.

• **How it works:**

1. The OS keeps a **ready queue** of processes waiting for CPU time.
2. It allocates CPU time using algorithms (like First Come First Served, Round Robin, etc.).
3. Balances performance, responsiveness, and fairness.

• **Example:**
When you stream a video while downloading a file, the OS schedules tasks so both can run smoothly — video playback doesn't freeze even though downloads are in progress.

---

5. Security Management

• **Definition:** The OS safeguards data and system integrity from unauthorized access or malicious activity.

• **Mechanisms:**

1. **Authentication:** Verifies user identity (passwords, biometrics, tokens).
2. **Authorization:** Determines which resources a user can access.
3. **Encryption:** Protects stored or transmitted data using cryptography.
4. **Auditing:** Logs activities for review and monitoring.
5. **Malware Protection:** Uses antivirus, firewalls, and intrusion detection systems (IDS).

Security Feature	Purpose	Example
Authentication	Confirms identity	Logging into your PC with a fingerprint

Authorization	Controls access	Admin vs guest account privileges
Encryption	Secures data	Encrypted Wi-Fi network (WPA3)
Auditing	Tracks actions	Windows Event Viewer logs
Malware Protection	Prevents attacks	Windows Defender blocking a virus

• **Example:**
When you log into macOS using Face ID, it authenticates you and restricts access to your files unless you are verified.

---

6. Accounting

• **Definition:** The OS keeps track of how system resources are used by users and processes.

• **Purpose:**

1. Helps administrators monitor performance and cost allocation.
2. Identifies bottlenecks and inefficiencies.
3. Enables **usage-based billing** in cloud systems.

Metric Tracked	**Description**
CPU usage	Time each process uses the CPU

Memory usage	Amount of RAM allocated per user
Disk usage	Space occupied by files
Network usage	Data sent and received

• **Example:**
Cloud providers like AWS and Azure use OS-level accounting to charge customers based on CPU hours and storage used.

---

7. Graphical User Interface (GUI)

• **Definition:** The GUI is the visual interface of the OS that allows users to interact with the system using **icons, menus, windows, and pointers.**

• **Functions:**

1. Simplifies interaction — no need for command-line knowledge.
2. Provides feedback through notifications and progress bars.
3. Enables multitasking via windows and taskbars.
4. Supports accessibility features like screen readers and magnifiers.

• **Example:**
Windows desktop icons, macOS Finder, and Ubuntu GNOME all use GUI elements to help users navigate and manage files.

---

8. Virtualization and Networking

• **Virtualization:**
Allows multiple **virtual machines (VMs)** to run on one physical system.
→ Managed by a **hypervisor** that allocates CPU, memory, and storage to each VM.

Example:
Running Windows and Linux simultaneously on a Mac using software like Parallels or VMware.

• **Networking:**
The OS manages network connections, assigning IP addresses, controlling firewalls, and enabling secure communication through protocols (TCP/IP, HTTPS, VPNs).

Example:
When you connect to Wi-Fi, the OS handles DHCP for IP allocation and firewall rules for safe internet access.

Questions for Revision

1. **Define** memory management in an operating system.
2. **Explain** how virtual memory supports system performance.
3. **Describe** how the operating system manages file creation and deletion.
4. **Identify** one purpose of defragmentation and **explain** its importance.
5. **Outline** how the OS uses device drivers to communicate with hardware.
6. **Explain** the differences between buffering, caching, and spooling with examples.
7. **Discuss** how scheduling ensures fairness in multitasking environments.
8. **Identify** two key security mechanisms used by the OS and **explain** how they protect users.
9. **Analyse** how OS accounting functions support performance monitoring in multi-user systems.
10. **Evaluate** how GUI design improves accessibility for diverse users.

A1.3.3 Scheduling Approaches

1. Definition

Scheduling is the process by which the operating system decides **which task should use the CPU and for how long**.
It ensures fair allocation of resources, reduces waiting time, and increases CPU efficiency.

When many processes are running at once, the OS uses a **scheduling algorithm** to determine the order of execution.

---

2. Types of Scheduling

Category	Description	Example
Pre-emptive	The OS can pause a process to start another more important one	Windows multitasking
Non-pre-emptive	Once a process starts, it runs until finished or blocked	Simple embedded systems

Example:
If you are downloading a file while typing an essay, the OS switches between both tasks rapidly so that neither freezes.

---

3. Common Scheduling Algorithms

a. First Come, First Served (FCFS)

• Processes are executed in the order they arrive.
• Simple to implement but can cause long waiting times if one process takes too long.

Example:
Three print jobs arrive. Job 1 is huge, so smaller jobs must wait even if ready.

Order	Process	Burst Time (ms)	Waiting Time (ms)

1	P1	10	0
2	P2	5	10
3	P3	2	15

---

b. Shortest Job Next (SJN) or Shortest Job First (SJF)

• The process with the shortest estimated run time executes first.
• Reduces average waiting time but may cause starvation of longer jobs.

Example:
If several small background tasks exist, large software updates may be delayed.

---

c. Round Robin (RR)

• Each process is assigned a fixed time slice (quantum).
• After its time expires, the next process gets CPU time.
• Ideal for **time-sharing systems**.

Example:
In an online meeting app, voice, video, and chat services all get regular CPU turns to keep everything running smoothly.

Time Quantum	**Result**
10 ms	Fair responsiveness for all tasks
Too small	Frequent context switching, high overhead

Too large	Longer wait times, less responsiveness

---

d. Priority Scheduling

• Each process has a priority value.
• Higher priority tasks run before lower ones.
• Can be pre-emptive or non-pre-emptive.

Example:
A virus scan may run at low priority while the video conference app runs at high priority.

Process	**Priority**	**Order Executed**
P1	1 (highest)	1
P2	3	3
P3	2	2

Problem:
Starvation can occur if low priority tasks never get CPU time.
Solution:
Use *aging* to gradually increase priority over time.

---

e. Multilevel Queue Scheduling

• Processes are grouped into separate queues such as system processes, interactive tasks, and batch jobs.
• Each queue has its own scheduling algorithm.
• Prevents interference between types of processes.

Example:
Interactive apps get round robin scheduling, while background updates get FCFS.

---

4. Goals of Scheduling

1. Maximize CPU utilization.
2. Minimize waiting and turnaround time.
3. Ensure fairness among users and processes.
4. Maintain responsiveness in real-time systems.

Real-World Example:
A smartphone OS schedules the camera, GPS, and app threads so that photos are processed instantly without lag.

Questions for Revision

1. **Define** scheduling in the context of operating systems.
2. **Identify** two differences between pre-emptive and non-pre-emptive scheduling.
3. **Describe** the purpose of the time quantum in round robin scheduling.
4. **Compare** FCFS and SJF in terms of waiting time and fairness.
5. **Explain** how priority scheduling may cause starvation and how aging solves it.
6. **Outline** one real-world example where multilevel queue scheduling is beneficial.

A1.3.4 Interprocess Communication (IPC)

1. Definition

Interprocess Communication refers to the **methods that allow different processes to exchange data and coordinate their actions**.
It is essential in multitasking systems where many programs must work together.

Example:
When a web browser and a media player communicate to stream a video, they use IPC to share data efficiently.

---

2. Purpose of IPC

1. **Data Sharing:** Processes can exchange information such as messages or files.
2. **Event Notification:** One process can signal another when an action is complete.
3. **Resource Sharing:** Prevents conflicts when multiple processes need the same resource.
4. **Modularity:** Large applications can be split into smaller cooperating processes.

---

3. Methods of Interprocess Communication

Method	**Description**	**Example**
Shared Memory	Two or more processes share a section of RAM for communication	Database cache shared among threads
Message Passing	Processes send messages via the OS kernel	Chat application transferring text
Pipes	Unidirectional data flow from one process to another	Command chaining in Linux (e.g., `cat file

Sockets	Used for communication over a network	Web browser communicating with a web server
Signals	Short notifications to indicate events	CTRL + C sending an interrupt signal
Semaphores	Variables that control access to shared resources	Printer queue management

---

4. Shared Memory

• Fastest method of IPC since processes read and write directly to the same memory space.
• Requires synchronization to prevent data corruption.
• Often used in systems needing high-speed communication.

Example:
Gaming engines use shared memory between graphics and physics engines to keep gameplay smooth.

---

5. Message Passing

• The OS provides message queues for processes to exchange data safely.
• Easier to implement but slower than shared memory because messages pass through the kernel.

Example:
Email servers use message queues to send and receive messages between user sessions.

---

6. Pipes

• One-way channel connecting the output of one process to the input of another.
• Named pipes allow persistent communication between unrelated processes.

Example:
In Linux, `ls | sort` sends directory listings to another process for sorting.

---

7. Sockets

• Enable two processes to communicate across a network.
• Based on Internet protocols like TCP or UDP.
• Essential for distributed systems and client-server applications.

Example:
Online games use sockets to send position and action data between players.

---

8. Semaphores and Synchronization

• A semaphore is a special variable used to manage access to shared resources.
• Prevents two processes from writing to the same data at the same time.
• Commonly used with shared memory.

Example:
In a multi-user printer system, semaphores ensure that only one print job accesses the printer at a time.

---

9. Problems in IPC

Issue	Description	Prevention
Deadlock	Two processes wait forever for each other's resources	Use timeouts or ordering of resource requests
Race Condition	Output depends on timing of concurrent access	Apply semaphores or locks

Starvation	Process never gets access to resource	Use fairness algorithms

---

10. Real-World Examples

1. **Web Browsers:** Each tab runs as a separate process that communicates with the main interface via IPC.
2. **Operating Systems:** Kernel and user processes exchange signals and messages for security and updates.
3. **Databases:** Multiple clients share memory for caching and synchronization.

Questions for Revision

1. **Define** interprocess communication and **state** its importance in multitasking systems.
2. **Identify** two types of IPC and **explain** how they differ.
3. **Describe** the role of semaphores in synchronization.
4. **Outline** an example of message passing in network communication.
5. **Explain** what causes a race condition and how it can be prevented.
6. **Compare** pipes and sockets in terms of purpose and usage.
7. **Analyse** how shared memory improves performance in real-time systems.