# Operating System (OS)

**What it is:**
An **Operating System** is a **system software** that acts as a **bridge between the user and computer hardware**. It controls and coordinates all parts of the computer.

OS is the heart of computer system.

Application programs requires an operating system to function and that OS can be windows, macOS, linux, android, etc.

All applications program send requests to OS for accessing hardware or resources.

**Operating System (OS)** is a very important **system software.**

**Why it is needed:**
Without an OS, hardware cannot understand user commands. The OS makes the computer **usable**, **efficient**, and **secure** by managing all resources.

**How it works:**
The OS runs continuously after the computer starts. It **manages CPU, memory, files, and devices**, allowing multiple programs to run smoothly and users to interact easily.

**When it works:**
It starts **as soon as the computer is powered on** (booting process) and runs until the computer is turned off.

**Main Purpose / Functions:**

1. **Process Management** – Handles execution of programs.
2. **Memory Management** – Allocates and frees memory.
3. **File Management** – Organizes and controls data storage.
4. **Device Management** – Controls input/output devices.
5. **User Interface** – Provides command-line or graphical interaction.
6. **Security** – Protects system and data from unauthorized access.

**Examples:**
Windows, Linux, macOS, Android, iOS.

**In short:**

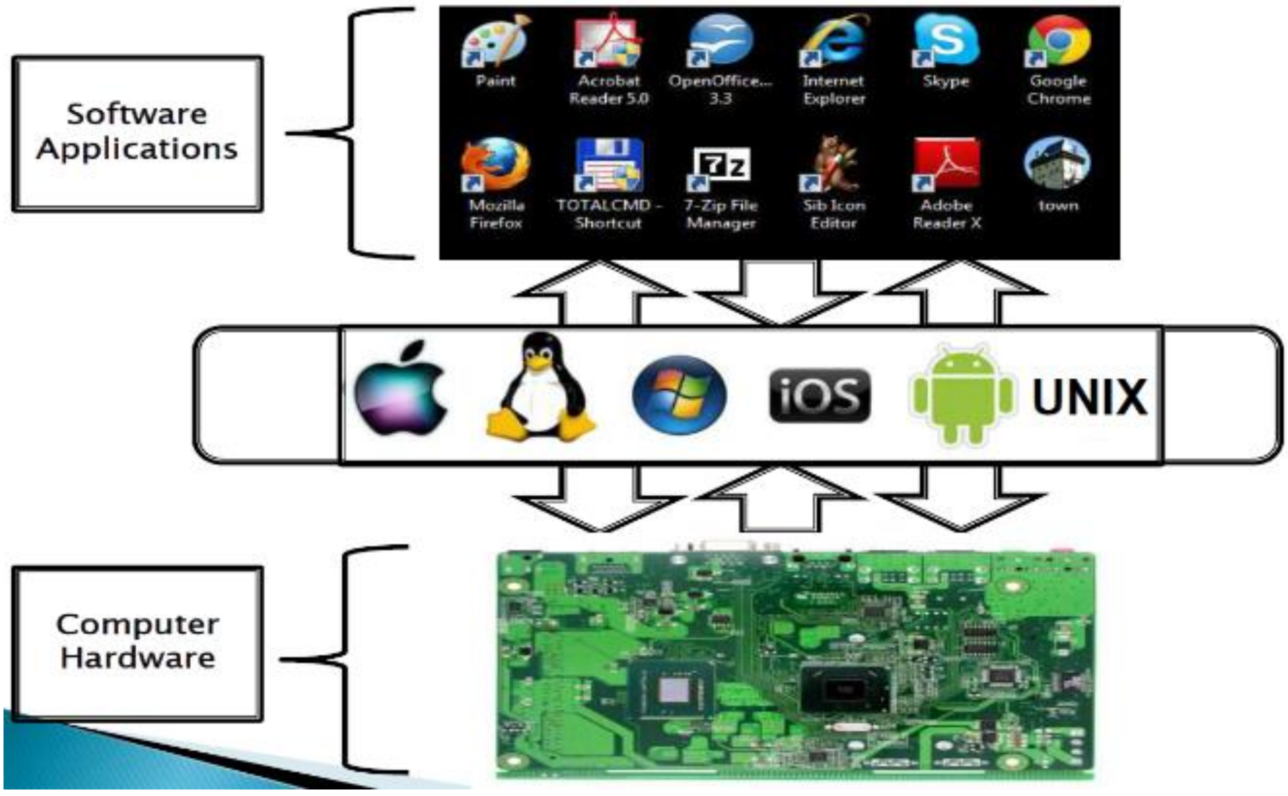
The Operating System is the **core software** that makes hardware work, allows users to run programs, and ensures smooth, safe, and efficient operation of the entire computer system.

**Main Parts / Layers of a Computer System: (Computer system structure):**

1. **Hardware:**
   - The physical components (CPU, memory, I/O devices).
   - Performs the actual computing and data processing.
2. **Operating System (System Software):**
   a. Acts as a **bridge between hardware and users/programs**.
   b. Manages all system resources (CPU, memory, files, devices).
3. **System Programs:**

a. Provide **basic utilities** and **support functions** (like file management tools, compilers, etc.).

4. **Application Programs:**
   a. Software designed for specific user tasks (MS Word, browsers, games, etc.).
5. **Users:**
   a. People or other systems that use the computer to perform tasks.



## Goals of an Operating System:

### 1. Primary Goals:

a. Convenience (User Goal):

- Make the computer system **easy, simple, and user-friendly** to use.
- Users should not worry about hardware details.
  *(Example: GUI interfaces, file management, etc.)*

b. Efficiency (System Goal):

- Use **hardware resources (CPU, memory, I/O)** efficiently so that performance is maximized.
- Ensures fast and smooth system operation.

c. Ability to Evolve (Development Goal):

- OS should be **flexible and upgradable** for future technologies or new hardware.

2. Secondary (Technical) Goals:

a. Resource Management:

- Manage and allocate CPU, memory, and devices among programs.

b. Security and Protection::

- Protect data and system resources from unauthorized access.

c. Multiprogramming & Multitasking:

- Allow multiple programs to run **simultaneously** without interference.

d. Error Detection & Recovery:

- Detect hardware/software errors and take corrective actions.

Like Linux is more efficient in terms of memory, security, viruses and but not convenient to use that's why windows has more users than Linux/uniux..

## Types of OS

| Type | Explanation (Concise) | Example |
|---|---|---|
| **1. Batch Operating System** | Executes a **batch (group)** of jobs automatically without user interaction. | IBM OS, early mainframe systems |
| **2. Time-Sharing (Multitasking) OS** | Allows **multiple users/programs** to share CPU time **simultaneously**. | UNIX, Windows |
| **3. Multiprogramming OS** | Runs **many programs at once** by keeping them in memory and switching CPU among them. | Linux, UNIX |
| **4. Real-Time OS (RTOS)** | Gives **immediate response** to inputs; used where timing is critical. | VxWorks, QNX, Embedded systems |
| **5. Distributed OS** | Manages **multiple computers** and makes them work as a **single system**. | LOCUS, Amoeba |
| **6. Network OS** | Manages and controls **networked computers**; allows resource sharing. | Novell NetWare, Windows Server |
| **7. Mobile OS** | Designed for **smartphones and tablets** with touch interface and limited resources. | Android, iOS |

### 1. Batch Operating System

**Concept:**
Executes a batch (group) of jobs automatically.

Jobs are collected and processed in batches without user interaction.

Examples: IBM OS, early mainframe systems

**Advantages:**

- No manual intervention once jobs start.
- Efficient for repetitive, large tasks.
- Reduces idle CPU time.

**Disadvantages:**

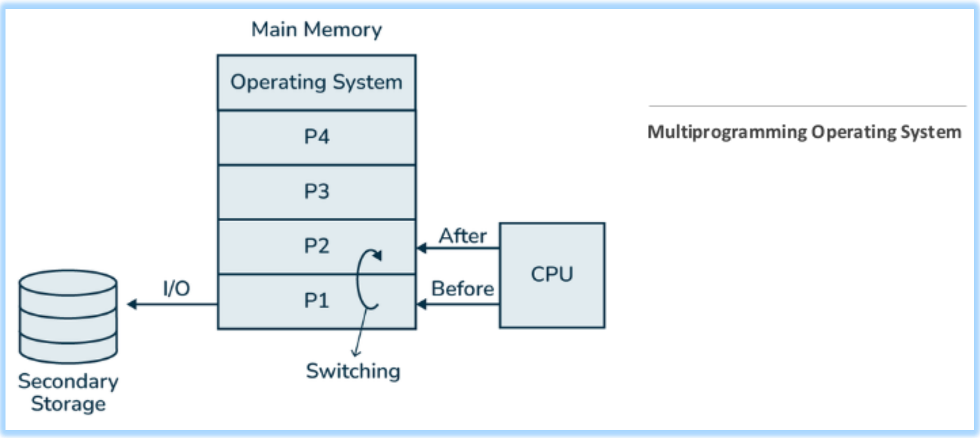
- No interaction between user and system.
- Debugging is difficult.
- CPU may stay idle due to slow I/O devices.

## 2. Multiprogramming Operating System

**Concept:**
Runs **many programs at once** by keeping them in memory and switching CPU among them.

Multiple jobs are kept in memory; CPU switches among them for efficient use.

Examples: Linux ,UNIX



**Advantages:**

- High CPU utilization.
- Reduces idle time.
- Efficient use of memory and resources.

**Disadvantages:**

- Complex CPU scheduling.
- Needs proper memory management.
- Long waiting time for some jobs.

## 3. Multitasking (Time-Sharing) Operating System

**Concept:**

Allows **multiple users/programs** to share CPU time **simultaneously**.
Each process gets CPU for a fixed time (time quantum) to enable multi-user interaction.
Examples: UNIX, Windows

**Advantages:**

- Fast response to users.
- Better CPU sharing and utilization.
- Interactive and user-friendly.

**Disadvantages:**

- Requires powerful CPU and more memory.
- May cause delay if many users share system.
- Complex to manage.

---

## 4. Multiprocessing Operating System

**Concept:**
Uses two or more CPUs within one system to perform tasks simultaneously.

Examples: linux, windows, etc.

**Advantages:**

- True parallel processing.
- High speed and performance.
- Reliable — if one CPU fails, others work.

**Disadvantages:**

- Very expensive hardware.
- Complex system design.
- Synchronization is difficult.

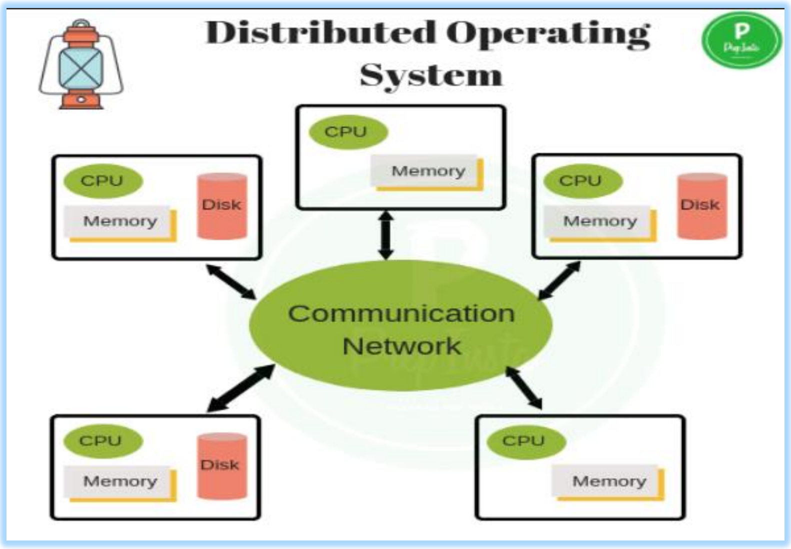
---

## 5. Distributed Operating System

**Concept:**
Manages **multiple computers** and makes them work as a **single system**.

Multiple computers (nodes) connected via a network work as one system.

Examples: LOCUS, amoeba.



**Advantages:**

- High reliability (one system's failure doesn't affect others).
- Resource and data sharing.
- Scalable and faster communication.

**Disadvantages:**

- Expensive setup and maintenance.
- Dependent on network reliability.
- Complex software and communication management.

---

## 6. Real-Time Operating System (RTOS)

**Concept:**

Gives immediate response to inputs; used where timing is critical.
Used in time-critical systems requiring immediate and predictable responses.
The time interval required to process and respond to inputs is very small. This time interval is called **response time**.
**Examples**: VxWorks, QNX, Embedded systems

**Types:**

- **Hard RTOS:** Strict deadlines (missile, aircraft systems).
- **Soft RTOS:** Flexible deadlines (multimedia, automation).

**Advantages:**

- Provides accurate and quick response.
- Highly reliable and stable.

- Ideal for real-time applications.

**Disadvantages:**

- Expensive and complex design.
- Limited multitasking ability.
- Difficult to upgrade.

---

## 1. Hard Real-Time System

**Definition:**
A system where **missing even a single deadline can cause system failure or disaster.**

**Example:**

- Aircraft control systems
- Missile guidance systems
- Medical life-support systems

**Key Points:**

- Deadlines are *strict and non-negotiable*.
- Must give response **within a fixed time**.
- Used where **accuracy and safety** are critical.

**Advantages:**

- High reliability and precision.
- Suitable for mission-critical tasks.

**Disadvantages:**

- Very expensive and complex.
- Limited flexibility.

---

## 2. Soft Real-Time System

**Definition:**
A system where **missing a deadline is tolerable** but **performance may degrade** slightly.

**Example:**

- Multimedia systems
- Online transaction systems
- Video streaming or gaming

**Key Points:**

- Deadlines are *important but not rigid*.
- Focus on **smooth performance** rather than strict timing.
- Common in general-purpose real-time applications.

**Advantages:**

- Flexible and easier to design.
- Less costly than hard RTOS.

**Disadvantages:**

- May cause delays or data loss in heavy load.
- Not suitable for safety-critical systems.

---

**User Operating System Interfaces:**

User OS Interface is the **bridge** between the **user** and the **computer system**, enabling interaction through **commands, visuals, or speech**.

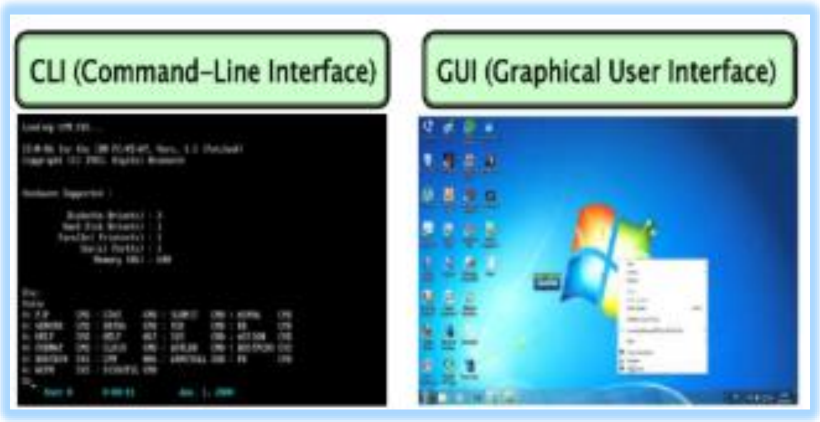
**Types of User Interfaces:**

1. **Command Line Interface (CLI):**

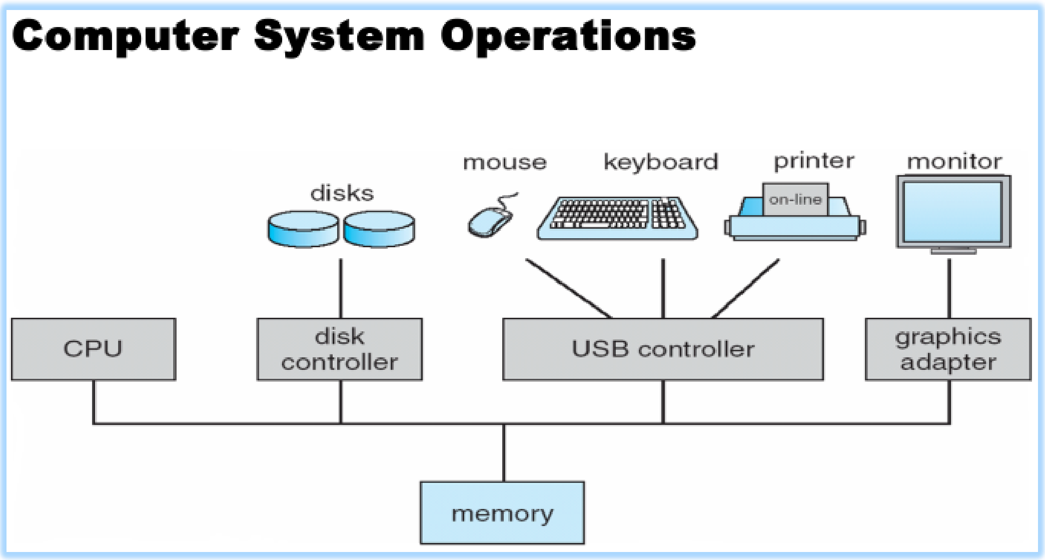
   Command Line Interface (CLI) **allows the user to interact with the system using commands.**

2. **Graphical User Interface (GUI):**

   Graphical User Interface (GUI) allows the user to **interact with the system using graphical** elements such as windows icons, menus.



---



- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer (local buffer ➔ temporary **storage area** in a computer's memory (usually **RAM**) that is used to **hold data temporarily** while it is being **transferred or processed** by a specific program, device, or function.).
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

**How the OS manages Users and Tasks:**

"The OS manages users by handling authentication, access, and privileges, and manages tasks by scheduling, executing, and coordinating multiple processes efficiently."

| Feature | Single User, Single Tasking | Single User, Multitasking |
|---|---|---|
| **Number of Users** | One | One |
| **Number of Tasks** | One at a time | Multiple at a time |
| **Examples** | MS-DOS, early phones | Windows, macOS, Android |
| **Performance** | Simple, limited | Efficient, flexible |
| **Memory Use** | Low | High |
| **CPU Utilization** | Low | High |

"A **Single User, Single Tasking** OS allows one program at a time, while a **Single User, Multitasking** OS allows one user to perform several tasks simultaneously using CPU scheduling."

---

**Threads:**

A **thread** is the **smallest unit of CPU execution** within a process.
Each thread shares the **same memory and resources** of its parent process but runs **independently**.

- **multiple parts of the same program** — called **threads**

**Example (Simple):**

When you use a web browser:

- One thread loads a webpage,
- Another thread downloads a file,
- Another thread plays a video —
  ➡ All running *simultaneously* under the same browser process (multi-threading).

## Multithreading:

Multithreading is an OS feature that allows multiple threads (multiple parts of same program) of a single process to run concurrently (**simultaneously**), improving CPU utilization and system performance.

## OS Architecture:

| Component | Description |
|---|---|
| 1. User Interface (UI) | Provides interaction between user and system (CLI, GUI, etc.) |
| 2. System Calls / API | Act as a bridge between user programs and OS kernel. |
| 3. Kernel | Core part of OS — manages CPU, memory, I/O, and processes. |
| 4. Device Drivers | Help the OS communicate with hardware devices. |
| 5. File System | Manages how data is stored and retrieved. |
| 6. Hardware Layer | Physical components like CPU, memory, and I/O devices. |

**Hardware =** physical element of computer system.

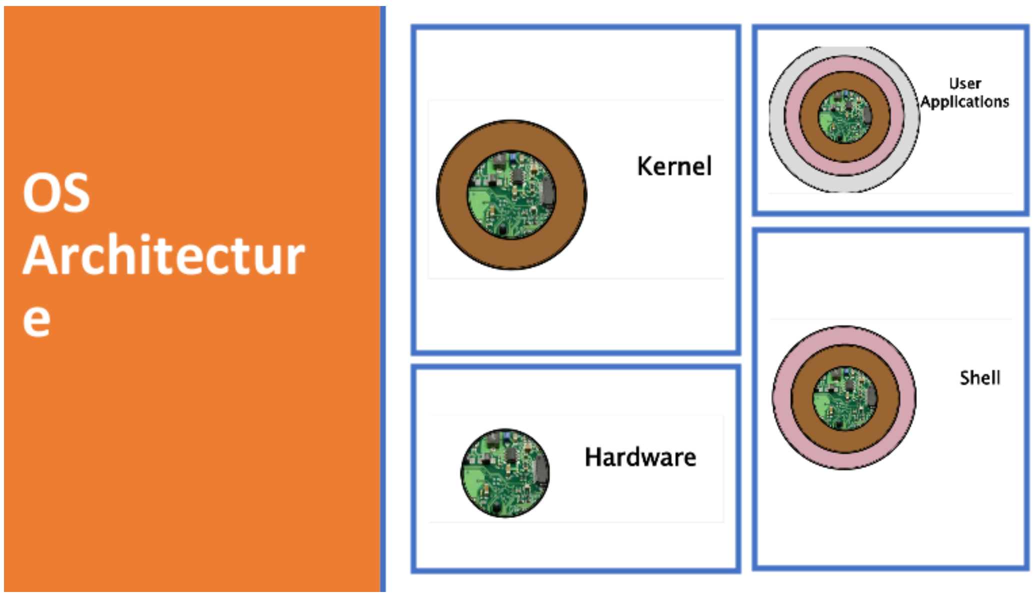
**Kernel =** core part of an OS that directly communicates with the hardware, manages system resources i.e. CPU, memory, I/O, and processes,,, and provide a **bridge between user applications and the hardware.**

**(OS → bridge b/w User and computer system , hardware)**

**(Kernel → bridge b/w user applications and computer hardware)**

**Shell =** operating system shells use either a **command-line interface (CLI) or graphical user interface** (GUI).

**User Applications =** are **computers program** designed to perform group of coordinated functions, tasks, or activities for the benefit of the user.



(Concept and explanation :

## 1. User Mode

**Definition:**
User Mode is the **less-privileged mode** in which **application programs** (like Word, PowerPoint, browsers, etc.) are executed.

**Examples:**

- MS Word
- PowerPoint
- Reading a PDF
- Browsing the Internet

**Key Points:**

- Has **limited access** to hardware resources.
- Runs **application-level programs** only.
- If a program in user mode needs hardware (e.g., printer, memory), it **requests the kernel** for service.
- Also called **Restricted Mode**, **Slave Mode**, or **Less-Privileged Mode**.

**Purpose:**
➡ To **protect the system** from accidental or malicious access to hardware by applications.

---

## 2. Kernel Mode

**Definition:**
Kernel Mode is the **privileged mode** where the **operating system kernel** runs and has **full access to all hardware components**.

**Key Points:**

- Kernel acts as a **bridge (middleware)** between hardware and application software.
- Used for **low-level, trusted functions** like memory management, process control, and I/O operations.
- Only the OS kernel and device drivers operate here.
- Errors in kernel mode can **crash the whole system** because it has full control.

**Relationship Between Both Modes:**

| Aspect | User Mode | Kernel Mode |
|---|---|---|
| **Privilege (authority )Level** | Low (Restricted) / Restricted / Slave mode. | High (Full Access) privileged / Master mode. |
| **Who Runs Here (which program are controlled by them)** | Applications | Operating System Kernel & Drivers |
| **Hardware Access** | Indirect (via Kernel) (Uses **system calls** to request kernel services.) | Direct |
| **Purpose** | Protect system from faulty programs | Manage hardware and system resources |
| **Examples** | MS Word, Browser, PDF Reader | Process Scheduler, Memory Manager |
| **Failure Impact** | Affects only the running program | Can crash the whole system |
| **Purpose** | To provide **protection** and **stability** by isolating applications from hardware. | To perform **core system functions** and manage all resources. |

"Who runs here" = *Which type of program or code executes in that mode.*

## Example:

When you open a Word file — it runs in **User Mode**.
When you **save** that file, Word requests the **Kernel Mode** (via a system call) to access the **disk hardware** safely.

## In Summary:

**User Mode = Safe zone for applications.**
**Kernel Mode = Power zone for OS.**

---

## Computer Startup Process (or booting process):

The **computer startup** (or **booting process**) is the sequence of steps a computer performs when it is powered on to load the **Operating System (OS)** into memory.

## Steps:

1. **Power On** → CPU gets control.
2. **BIOS/UEFI runs** → checks hardware (POST – Power-On Self Test).
3. **Bootstrap program** (in ROM) is loaded → it finds the OS.
4. **OS Kernel is loaded** from storage into main memory (RAM).
5. **System is ready** for user operations.

**In short:**

Booting = Loading the OS into memory to start the computer.

## Bootstrap Program:

A **bootstrap program** is a small program stored in **ROM** (Read Only Memory) that runs when the computer is powered on.
It's the **first code** executed by the CPU.

### Function:

- Initializes hardware components.
- Loads the **Operating System kernel** from disk to RAM.
- Transfers control to the OS.

**Think of it as:**

"The program that wakes up the computer and calls the OS."

## Interrupt:

An **interrupt** is a **signal sent to the CPU** by hardware or software to get its attention when an immediate action is required.

### Purpose:

To **pause the current execution**, handle the event, then **resume** the previous process.

### Types:

| Type | Source | Example |
|---|---|---|
| **Hardware Interrupt** | Sent by hardware devices. | Keyboard input, mouse click, I/O completion. |
| **Software Interrupt** | Generated by a program or OS service. | System call, divide-by-zero error. |

**In short:**

Interrupts help the CPU **respond quickly** to urgent events.

## Interrupt Handling:

The process by which the **OS responds to an interrupt** is called **Interrupt Handling**.

### Steps:

1. Interrupt signal is generated.
2. CPU **pauses** current execution.
3. Control is transferred to **Interrupt Handler (Service Routine)**.
4. Interrupt is serviced (e.g., read input).
5. CPU **returns** to the previous task.

**In short:**

Interrupt Handling = Detect → Pause → Handle → Resume.

---

## System Calls:

A **system call** is a **request made by a user program to the operating system** to perform a service that requires **kernel-level access** (like I/O, file, or process control).

### Purpose:

User programs **can't access hardware directly**, so they use system calls to communicate safely with the kernel.

**In short:**

System Call = Bridge between **User Mode** and **Kernel Mode**.

### Types of System Calls:

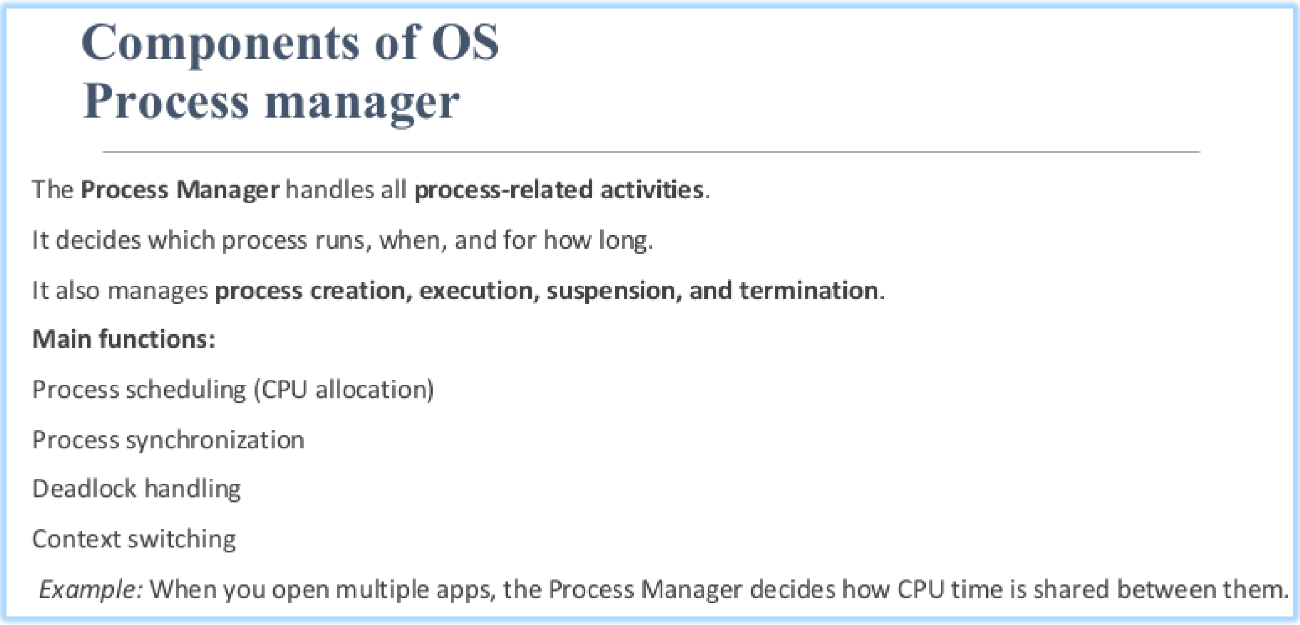
| Type | Purpose / Example |
|---|---|
| **Process Control** | Create, execute, or terminate processes. e.g., `fork()`, `exit()`, `wait()` |
| **File Management** | Create, read, write, delete, open, close files. e.g., `open()`, `read()`, `write()` |
| **Device Management** | Request or release devices. e.g., `ioctl()`, `read()`, `write()` |
| **Information Maintenance** | Get or set system data. e.g., `getpid()`, `settime()` |
| **Communication** | Exchange information between processes. e.g., `pipe()`, `send()`, `recv()` |

## Summary Table:

| Concept | Purpose | Example |
|---|---|---|
| Computer Startup | Load OS into memory to start system. | Windows/Linux boot sequence. |
| Bootstrap Program | First program in ROM that loads OS kernel. | BIOS, UEFI, GRUB. |
| Interrupt | Signal to CPU for urgent attention. | Keyboard input, divide-by-zero. |
| Interrupt Handling | Process of responding to interrupts. | Printer or disk interrupt. |
| System Call | User program requests OS service. | `read()`, `fork()`, `exit()`. |

| | | |
|---|---|---|
| Types of System Calls | File, Process, Device, Info, Communication. | `open()`, `write()`, `pipe()`. |

**Only one interrupt is generated per block, rather than the one interrupt per byte.**



## Components of OS
## Process manager

The **Process Manager** handles all **process-related activities**.

It decides which process runs, when, and for how long.

It also manages **process creation, execution, suspension, and termination**.

**Main functions:**

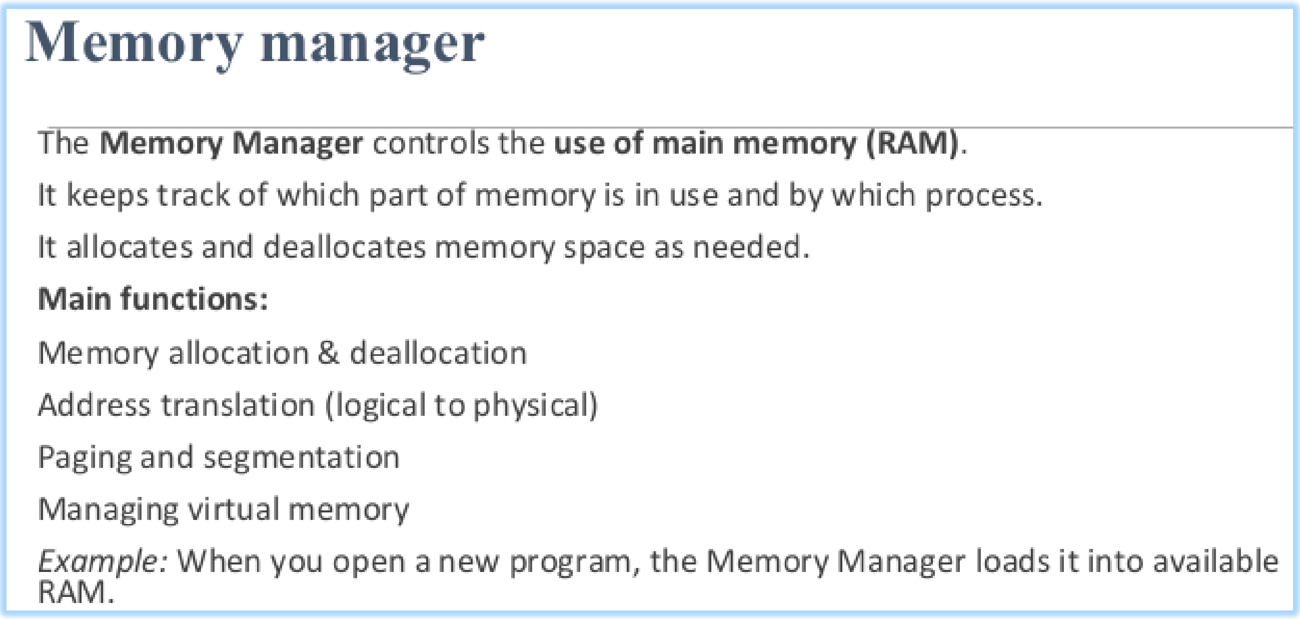
Process scheduling (CPU allocation)

Process synchronization

Deadlock handling

Context switching

*Example:* When you open multiple apps, the Process Manager decides how CPU time is shared between them.



## Memory manager

The **Memory Manager** controls the **use of main memory (RAM)**.

It keeps track of which part of memory is in use and by which process.

It allocates and deallocates memory space as needed.

**Main functions:**

Memory allocation & deallocation

Address translation (logical to physical)

Paging and segmentation

Managing virtual memory

*Example:* When you open a new program, the Memory Manager loads it into available RAM.

## File manager

The **File Manager** manages all **file operations** like creating, reading, writing, and deleting files.

It also maintains information about files (metadata) and access permissions.

**Main functions:**

File creation, deletion, and manipulation

Directory management

Access control

File storage organization

*Example:* When you save a Word document, the File Manager stores it properly on disk.

## Device manager

The **Device Manager** handles all **input/output (I/O) devices** such as printers, keyboards, and disks.

It acts as a bridge between hardware devices and user applications.

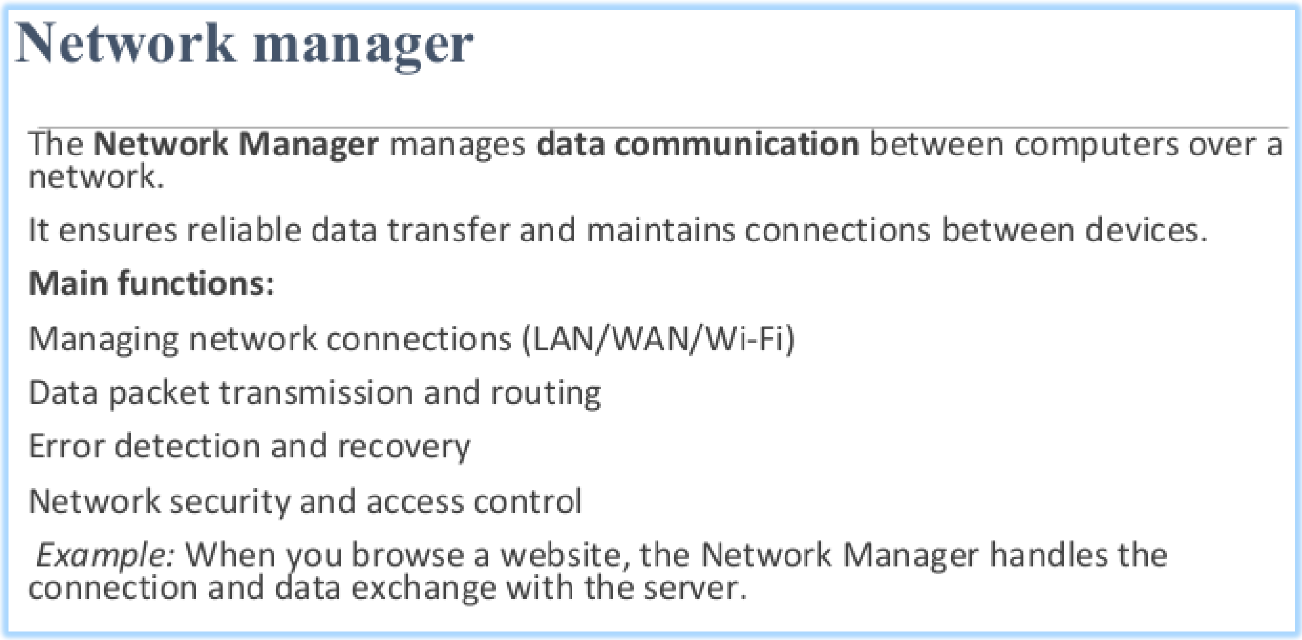
**Main functions:**

Device communication and control

Device scheduling

Error handling

*Example:* When you print a file, the Device Manager manages communication between the OS and the printer.



## Network manager

The **Network Manager** manages **data communication** between computers over a network.

It ensures reliable data transfer and maintains connections between devices.

**Main functions:**

Managing network connections (LAN/WAN/Wi-Fi)

Data packet transmission and routing

Error detection and recovery

Network security and access control

*Example:* When you browse a website, the Network Manager handles the connection and data exchange with the server.

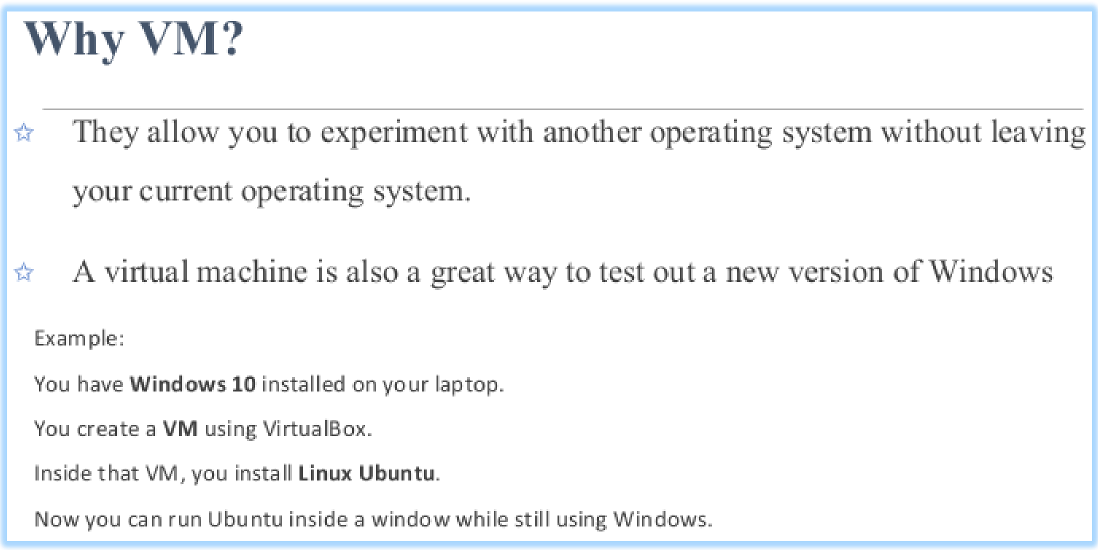
---

## Virtual Machine (VM):

A Virtual Machine is a "computer inside a computer."

A **Virtual Machine (VM)** is a **software-based simulation of a computer system** that behaves like a real physical computer.
It runs an **operating system and applications** just like a real machine, but it is created and managed by another software called a **Hypervisor**.

it has its own CPU, memory, storage, and operating system — but it runs inside another physical machine.



### Why VM?

☆ They allow you to experiment with another operating system without leaving your current operating system.

☆ A virtual machine is also a great way to test out a new version of Windows

Example:

You have **Windows 10** installed on your laptop.

You create a **VM** using VirtualBox.

Inside that VM, you install **Linux Ubuntu**.

Now you can run Ubuntu inside a window while still using Windows.

### How it Works:

- The **Hypervisor** (or Virtual Machine Monitor) divides the physical hardware (CPU, memory, disk, etc.) into multiple **virtual environments**.
- Each VM runs its **own Operating System** (called a *guest OS*) independent of others.
- The **real hardware** has a **host OS** (like Windows, Linux, macOS).

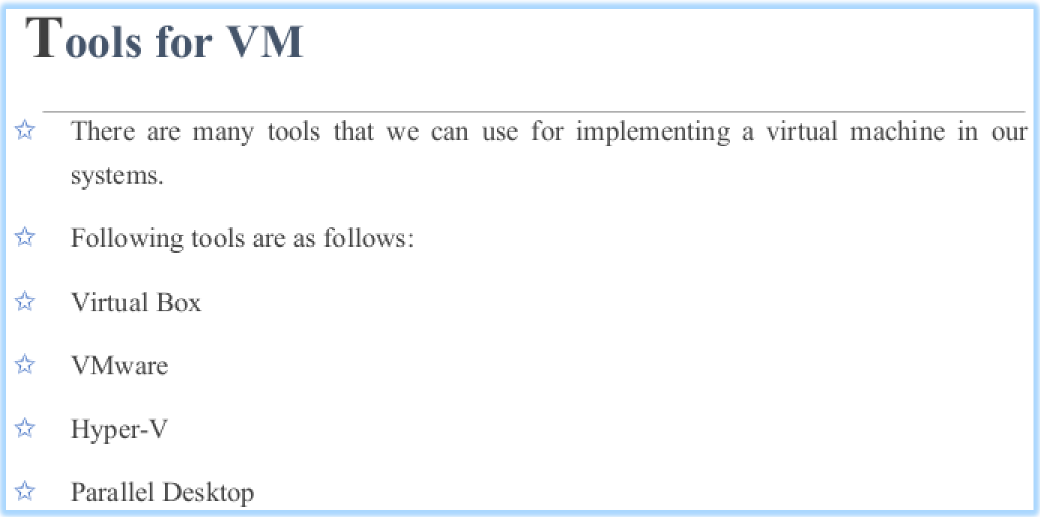
## Advantages:

✅ Efficient hardware utilization (one PC can run many OSs)
✅ Isolation — one VM crash doesn't affect others
✅ Easier software testing (different OSs on one machine)
✅ Supports system backup, recovery, and portability

---

## Disadvantages:

❌ Slower performance than real hardware
❌ Needs more RAM and CPU power
❌ Complex management and setup
❌ Requires large disk space



### Tools for VM

- ☆ There are many tools that we can use for implementing a virtual machine in our systems.
- ☆ Following tools are as follows:
- ☆ Virtual Box
- ☆ VMware
- ☆ Hyper-V
- ☆ Parallel Desktop