Operating System (OS)

Concept of Operating System (OS)

An **Operating System** is system software that acts as an **interface between user and computer hardware**.
It manages all hardware components (CPU, memory, keyboard, mouse, disk) and allows applications (like browsers, games, MS Word) to run smoothly.

Lets take an example:

Whenever you want to print something, you are trying to access the printer.
But you cannot directly access the printer because your high-level actions (like clicking Print, the file data (text, images, PDF)) are not understood by the printer.
So the OS works in between — it takes your print command, converts it into low-level instructions that hardware can understand and then sends it to the printer.
That's why the OS acts as an interface between you and the printer.

How it act as an interface between us and printer:

Whenever you want to print a file, you click on **Print**.
The OS takes your file and sends it to the **printer driver**, which converts it into a format the printer understands.
Then the OS sends this converted data to the **printer** through USB or Wi-Fi.
The printer reads that data and starts printing your pages.
So the OS works between you and the printer to make the printing process happen smoothly.

Another example:

When you click open file, the OS understands your click and asks the **storage driver** to read that file.
The storage driver converts this request into low-level commands that the hard disk can understand.
The hard disk then reads the data and sends it back to the OS then OS moves the file to RAM → CPU processes it → application displays it.

When you click, type, or run a program, the OS converts these high-level actions into low-level instructions that hardware can understand.

The user never deals with binary instructions — the OS does it.

Simple Example

You click **Play Music**:

1. User → Clicks icon
2. OS → Loads music software into RAM
3. OS → Sends instructions to CPU
4. OS → Fetches music file from disk
5. OS → Sends audio signals to sound card

6. Sound card → Sends sound to speakers

So OS sits **in the middle**, connecting everything.

Functions of Operating System (Explained with Examples)

1. Process Management

Process management handles the **creation**, **execution**, **scheduling**, and **termination** of processes.

What it includes:

- Creating a process (loading program into memory)
- Scheduling processes to use CPU
- Suspending/resuming processes
- Terminating processes after completion

Example:

- When you open **Chrome**, the OS creates a process for it.
- OS schedules Chrome and other apps using CPU time.
- When you close Chrome, the OS terminates its process.
- OS uses algorithms like **FCFS, Round Robin** for scheduling.

---

2. Memory Management

Memory management deals with **how memory (RAM) is allocated and used** by processes.

What it includes:

- Allocating memory to each process
- Keeping track of memory usage
- Avoiding memory overflow
- Swapping processes between RAM and hard disk

Example:

- When you open multiple apps like **WhatsApp + YouTube + Word**, OS divides RAM among them.
- If RAM is full, OS may perform **swapping** (move inactive programs to disk).
- OS prevents two apps from using the same memory location (protection).

---

3. File Management

File management organizes and controls how data is **stored, retrieved, named, and protected** on storage devices.

What it includes:

- Creating files and directories
- Reading/writing files
- Managing storage space
- Ensuring file security

Example:

- When you save a file in **Documents**, OS decides:
    - where to store it
    - how to name it
    - who can access it
- File permissions like **Read/Write/Execute** are set by OS.
- The OS maintains a **File Allocation Table (FAT)** or **inode table**.

---

4. Device Management

OS manages input/output devices through **device drivers**.

What Are Devices?

Devices are the **hardware components** connected to a computer that perform **input**, **output**, or **storage** functions.

Types of Devices

1. Input Devices

These are used to give data/commands to the computer.

Examples: Keyboard, Mouse,Scanner , Microphone, WebcamBarcode , reader

2. Output Devices

These display or produce the result from the computer.

Examples: Monitor,Printer,Speakers,Projector,Headphones

3. Storage Devices

Used to store data permanently or temporarily.

Examples: Hard Disk (HDD),SSD,Pen ,DriveMemory Card,CD/DVD

4. Communication Devices

Used for network and data transfer.

Examples: Wi-Fi Adapter,Modem,Network Interface Card (NIC),Bluetooth Adapter

5. Input/Output (I/O) Devices

Devices that perform both input and output.

Examples: Touchscreen,Network devices,USB devices

What Are Device Drivers?

A **Device Driver** is a **small software program** that allows the OS to communicate with hardware devices.

Why drivers are needed?

Because OS cannot directly understand every hardware.
The **driver acts as a translator** between the device and the OS.

Examples of Device Drivers

Here are some **real-life examples**:

1. Keyboard Driver

- Allows OS to detect key presses.
- Example: Standard PS/2 Keyboard Driver in Windows.

2. Mouse Driver

- Converts mouse movement and clicks into actions on screen.

3. Printer Driver

- Translates document data into a format that printer understands.
- Example: HP LaserJet driver, Canon printer driver.

4. USB Driver

- Helps OS detect USB flash drives, keyboards, cameras.
- Example: USB 2.0 / USB 3.0 drivers.

5. Graphics Driver (GPU Driver)

- Controls display, images, videos.
- Example: NVIDIA Driver, AMD Radeon Driver, Intel HD Graphics Driver.

6. Audio Driver

- Manages sound output and microphones.
- Example: Realtek HD Audio Driver.

7. Wi-Fi / Network Driver

- Allows device to connect to Internet.

- Example: Qualcomm Atheros Wi-Fi driver, Intel Wi-Fi driver.

8. Bluetooth Driver

- Enables Bluetooth communication.
- Example: Broadcom Bluetooth Driver.

9. Touchpad Driver (Laptop)

- Controls touchpad gestures.
- Example: Synaptics Touchpad Driver.

What it includes:

- Communicating with devices (keyboard, mouse, printer, USB)
- Allocating device access
- Handling interrupts
- Loading appropriate drivers

Example:

- When you plug in a **pendrive**, OS automatically detects it using drivers.
- When you print a file, OS sends it to **printer buffer**.
- If multiple apps need the printer, OS decides the order.

---

5. Security & Protection

Security ensures protection from **unauthorized access**, while protection controls **access rights** to system resources.

What it includes:

- User authentication (password, PIN, fingerprint)
- Permission control (read/write/execute)
- Encryption
- Preventing malware and unauthorized usage

Example:

- Windows login screen asking for **password or face unlock**.
- Permission settings:
    - File can be **Read-only**
    - Some apps require **admin rights**
- Antivirus and firewall features are part of OS security.

---

6. Error Detection & Handling

OS detects and handles errors that occur during operation.

Types of errors handled:

- Hardware errors (disk failure, memory issues)
- Software errors (invalid instructions)
- I/O errors (printer jam, device not responding)

Example:

- If a program stops responding, Windows shows:
  "The application is not responding."
- OS may automatically restart the application or ask to “Close the program”.
- Error messages like **“USB device not recognized”**.

---

7. User Interface

User Interface allows users to interact with the computer.

Types:

CLI (Command Line Interface)

User types commands.
Example:
Windows CMD, Linux Terminal
Command:
cd Desktop

GUI (Graphical User Interface)

Uses windows, icons, menus, buttons.
Example:
Windows, macOS, Android
You click a **folder icon** to open it.

Shell

Shell is an interface between user and OS (CLI or GUI shell).
Example:

- **Bash Shell** in Linux
- **PowerShell** in Windows
- **Finder shell** in macOS

Real-Life Applications of Operating Systems

Operating Systems are used everywhere in daily life. Some real-life applications include:

---

1. Smartphones

- Android and iOS are operating systems.
- They help run apps like WhatsApp, Instagram, banking apps, and camera.

---

2. Computers & Laptops

Windows, macOS, and Linux run all software like MS Office, browsers, games, etc.

Computer Operating Systems

- **Windows** (Windows 10, Windows 11)
- **Linux** (Ubuntu, Fedora, Kali)
- **macOS** (MacBook, iMac)

Used in schools, offices, shops, and homes.

---

3. ATMs (Automated Teller Machines)

- ATMs use operating systems (usually Linux or Windows Embedded) to manage money withdrawal and account access safely.

---

4. Cars (Automotive Systems)

- Modern cars use OS in infotainment systems, navigation (GPS), sensors, airbags, and automatic braking systems.

---

5. Smart Home Devices

- Devices like smart TVs, Alexa, Google Home, washing machines, and microwaves use embedded operating systems.

---

6. Hospitals & Medical Equipment

- OS runs machines like MRI, X-ray, ECG, ventilators, and hospital management systems.

---

7. Airlines & Railways

- Ticket booking systems, displays, check-in machines, and flight control systems run on OS.

---

8. Banking Systems

- Servers and computers in banks run on OS to manage transactions, accounts, and security.

---

9. Industry & Manufacturing

- Robots, assembly machines, sensors, and automation systems run on embedded OS.

Generations of Operating Systems

Computers did not always look like today's laptops. As technology improved, operating systems also evolved. Their development is divided into **5 generations**.

1st Generation (1940–1950): No Operating System

What was happening?

- Computers were huge machines using **vacuum tubes**.
- There were **no monitors**, **no keyboards**, **no mouse**, **no OS**.
- Everything had to be done manually by the computer operator.

How did people use computers?

- Programmers wrote programs in **machine language (0s and 1s)**.
- They punched the code onto **cards or paper tape**.
- Operators loaded the cards into the machine.
- Computer executed instructions directly from hardware.

Main characteristics:

- **No OS** → computer did not manage anything.
- **Single program** ran at a time.
- Very slow and difficult to operate.
- Used mostly for scientific calculations.

Example:

If you wanted to add two numbers, you had to write binary instructions and put them on punched cards.

2nd Generation (1950–1960): Batch Operating Systems

What changed?

- Vacuum tubes were replaced by **transistors** → computers became faster.
- For the first time, simple **Operating Systems started**.

How batch systems worked?

- Programs were submitted in **batches** (groups).
- Users did **not** interact with computer directly.
- OS processed one batch after another.

Main characteristics:

- No real-time user interaction.
- Used **job scheduling** (queue).
- Faster than 1st gen.
- OS handled:
    - Loading programs
    - Running them
    - Giving output

Example:

100 students submit their programs → OS executes one by one automatically.

3rd Generation (1960–1980): Multiprogramming & Time-Sharing OS

What improved?

- Use of **ICs (Integrated Circuits)** → computers became smaller and powerful.
- Need for multitasking increased.

Two major types of OS came:

a) **Multiprogramming OS**
Multiprogramming means keeping multiple programs in memory at the same time so the CPU can switch between them and stay busy.

Multiple programs loaded in memory.

CPU switches between programs → increases utilization.

No CPU idle time.

Simple Example

Imagine you have 3 programs:

Program A

Program B

Program C

In multiprogramming:

- A runs → waits for input
- CPU switches to B
- B runs → waits for output
- CPU switches to C
- C runs → waits
- CPU switches back to A

➞ All programs **progress together**
➞ But still, **only one instruction runs at a time**, CPU switches very fast

b) Time-Sharing OS

- Many users can work **at the same time**.
- CPU time is divided into **time slices**.

Main characteristics:

- **Multitasking** started.
- **Interactive computing** (users used terminals).
- Faster processing.
- Concepts like:
    - Memory management
    - File systems
    - I/O management
    - Scheduling algorithms (FCFS, RR, SJF)

Example:

Typing on a terminal while someone else is compiling a program—both work together.

4th Generation (1980–Present): Personal Computers (PCs)

What changed?

- Microprocessors were invented.
- Computers became **small and affordable**.

Operating systems for personal use were developed:

- MS-DOS

- Windows
- macOS
- Linux
- UNIX

Main characteristics:

- **GUI (Graphical User Interface)**: Windows, icons, mouse.
- **Multitasking** became common.
- Introduced:
    - Networking
    - Security
    - User accounts
    - Device drivers
    - File management

Example:

Laptops, desktops running Windows 10/11, Linux Ubuntu, macOS.

5th Generation (Present & Future): AI-Based Operating Systems

What is new?

- Focus on **Artificial Intelligence**, **cloud computing**, **virtualization**, and **mobile OS**.

Examples:

- **Android**, **iOS** – smartphones.
- **Cloud OS** like Chrome OS.
- **AI assistants** integrated with OS (Cortana, Siri).

Main characteristics:

- Speech recognition.
- Natural language control.
- Cloud integration.
- High security.
- Virtual machines and containers (VMware, Docker).

Generations of Operating Systems(summary)

Generation	Years	Hardware Used	Monitor & Keyboard?	How Programs Were Written	OS Features	Examples
1st Generation	1940–1950	Vacuum Tubes	❌ **No monitor** ❌ **No keyboard**	Only **punched cards**, machine language	No OS, manual operation	No OS
2nd Generation	1950–1965	Transistors	❌ **No monitor** ❌ **No keyboard**	Still **punched cards**, batch of jobs	Batch OS, faster processing	IBM Batch OS
3rd Generation	1965–1980	ICs (Integrated Circuits)	✓ **Monitor introduced**✓ **Keyboard introduced**	Interactive typing on monitor, no punched cards	Multiprogramming, Time-sharing	UNIX
4th Generation	1980–Present	Microprocessors	✓ Monitor & Keyboard✓ GUI & mouse	GUI-based programs, files, apps	Network OS, multitasking	Windows, macOS, Linux
5th Generation	Present & Future	AI processors, advanced chips	✓ Smart screens, voice, touch	Voice, touch, AI interaction	AI-based, cloud OS	Android, iOS, AI OS

Types of Operating Systems

---

1. Batch Operating System

Concept

- Users do not interact directly with the computer.
- Tasks (jobs) are collected, grouped into batches, and executed one by one.
- A batch monitor controls job flow.

Why it was used?

In old days, computers were very slow and expensive.
So users submitted tasks on punched cards → the operator made batches → computer executed them.

Features

- No direct interaction
- Jobs executed sequentially
- Good for long ofline processing

Examples

- Early IBM Mainframe Batch OS

---

2. Time-Sharing Operating System

Concept

- Many users use the computer at the same time.
- CPU switches between users very quickly.
- This gives a feeling that each user has their own computer.

How it works?

- Each user gets a tiny time slice of CPU
- OS switches so fast → multitasking feels smooth

Features

- Interactive
- Good response time
- Supports multiple users

Examples

- UNIX

- Linux multi-user environments

---

3. Multiprogramming Operating System

Concept

- Multiple programs are loaded into memory.
- If one program waits (like for I/O), CPU switches to the next program.
- This increases CPU utilization (no idle time).

How OS manages?

- Keeps a job queue
- Scheduler selects which program to run next

Features

- Increases efficiency
- Better resource utilization
- CPU always busy

Examples

- Early IBM OS/360
- Modern systems also use multiprogramming inside

---

4. Multiprocessing Operating System

Concept

- System has multiple CPUs/cores.
- OS divides tasks among these CPUs.
- True parallel execution happens.

How it works?

- Shared memory
- Multiple CPUs run different processes at the same time
- Load balancing used by OS

Types

- SMP (Symmetric multiprocessing): All CPUs equal
- AMP (Asymmetric multiprocessing): One CPU master, others slaves

Examples

- Windows
- Linux
- macOS
- Android (multi-core)

---

5. Distributed Operating System

Concept

- Many computers connected in a network and behave like one big system.
- Tasks are distributed among machines.

Goal

- Share resources
- Increase reliability
- Faster computing

Example

If one machine is idle → OS sends the work to that machine.

Examples

- LOCUS
- Amoeba OS
- Modern cloud systems conceptually work like this

---

6. Real-Time Operating System (RTOS)

Definition

A **Real-Time OS** is an operating system that gives an output **within a fixed, predictable time**, not too fast or too slow.
Its main goal is **correct timing + correct results**.

Why "Real-Time"?

Because the system must respond **within a deadline**.
If it misses the deadline → the system can fail.

Examples of RTOS

- Automotive airbag system
- Pacemaker in medical equipment
- Missile guidance
- Industrial robots

- Drones & flight control systems
- Anti-lock braking system (ABS)

Types of Real-Time OS

Hard Real-Time OS

- Missing the deadline = **system failure**
- Examples: Airbag system, pacemaker

Soft Real-Time OS

- Missing the deadline reduces performance but does not fail
- Examples: Video streaming, online gaming

Features of RTOS

- Fast context switching
- Minimum interrupt latency
- Deterministic timing
- Priority-based scheduling
- Small size, low memory usage

Simple Example

If an airbag system takes **5 seconds** to respond, it is useless.
So RTOS ensures it responds within **5 milliseconds** every time

7. Network Operating System (NOS)

A **Network Operating System (NOS)** is an operating system designed to **manage, control, and support computers connected over a network**.

It allows multiple computers to **share files, printers, applications, and communicate** with each other. It Manages and provides services to computers connected in a network.

Main Features of a Network OS

- File sharing across computers
- Printer and device sharing
- User account management
- Security and permissions
- Remote login and remote access
- Backup and centralized control
- Communication services (email, messaging)

Examples of Network Operating Systems

- **Windows Server** (most common)
- **Linux Server / Ubuntu Server**
- **Novell NetWare**
- **UNIX**
- **MacOS Server**

Where is NOS used?

- Colleges and school labs
- Office networks
- Banks and corporate organizations
- Data centers
- Cloud servers

8. Mobile Operating System

Concept

- Designed for smartphones, tablets.
- Optimized for touchscreen, battery, sensors, apps.

Examples

- Android
- iOS

System Calls

1. What Are System Calls?

System Calls are interfaces that allow user programs (applications) to request services from the Operating System (OS).

Example:
• When you save a file → your program cannot directly touch the hard disk.
• It asks the OS using a system call like write().
The OS then performs the action safely.

---

2. Why Do We Need System Calls?

User programs cannot directly access hardware because it would be unsafe and can damage the system.

System calls help in:

- Protection
- Controlled hardware access
- Resource management
- Communication between user program and OS

---

3. How System Calls Work (Simple Explanation)

User Program → System Call → OS Kernel → Hardware

Example:
open("data.txt")

1. Program requests OS to open file
2. OS checks if file exists
3. OS gives file handle/descriptor
4. Program uses that file handle to read/write

---

4. Types of System Calls

A. Process Control System Calls

Used to create, end, or manage processes.

Examples:

- fork() → create a process
- exec() → run a new program
- exit() → end a process
- wait() → wait for a child process

Student Example:
When you open Chrome → OS uses fork() and exec() to start it.

B. File Management System Calls

Used to read, write, open, close files.

Examples:

- open()
- read()
- write()
- close()
- create()
- delete()

Example:
Saving a Word document uses write().

C. Device Management System Calls

Used to access hardware devices (printer, keyboard, etc.)

Examples:

- ioctl() → control device
- read() / write() → device I/O
- open() / close() device files

Example:
When you print a PDF, the OS uses device system calls to send data to the printer.

D. Information Maintenance System Calls

Used for getting system info, time, date, memory, etc.

Examples:

- getpid() → get process ID
- gettimeofday() → get time
- alarm() → set timer

Example:
Your alarm app uses a system call to schedule a wake-up timer.

E. Communication System Calls (IPC – Inter Process Communication)

Used for processes to send/receive data.

Examples:

- pipe()
- send() / recv()
- shmget() → shared memory
- mmap() → map file to memory

Example:
WhatsApp running in background talking to notifications uses IPC calls.

Real Life Example for Students

Consider a simple program:

printf("Hello");

printf() is not system call — but it uses one internally:

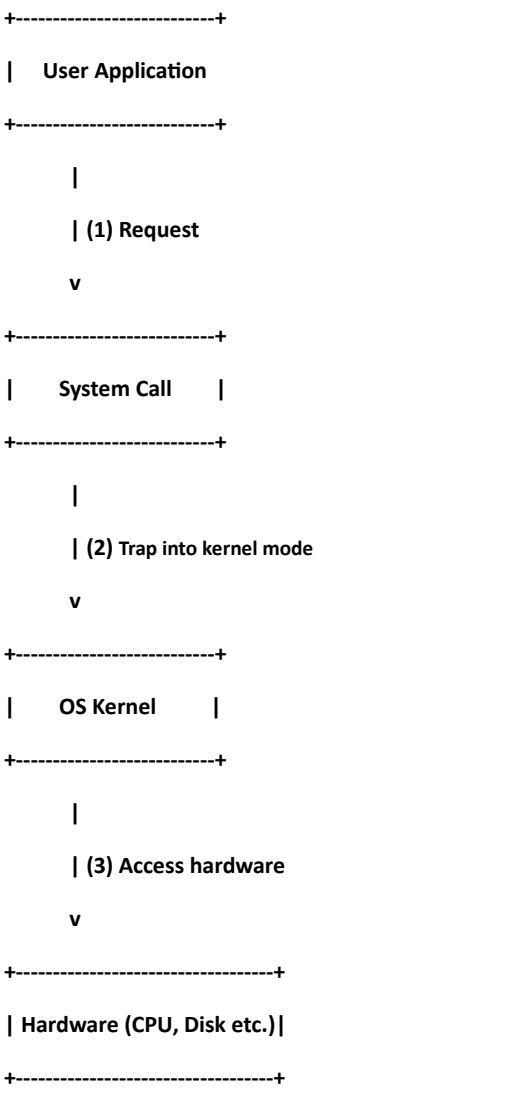
➡️ It calls write() system call
➡️ OS writes “Hello” to the screen

So the flow is:

Your Program → write() → OS Kernel → Display Hardware

---

6. Diagram: System Call Working (Text Version)



STRUCTURE OF OPERATING SYSTEM

Operating systems can be designed in different ways to manage hardware and software.
The three main OS structures are:

1. **Monolithic Structure**
2. **Layered Structure**
3. **Microkernel Structure**

Let's understand them deeply one by one.

---

1. Monolithic Operating System

✓ Definition

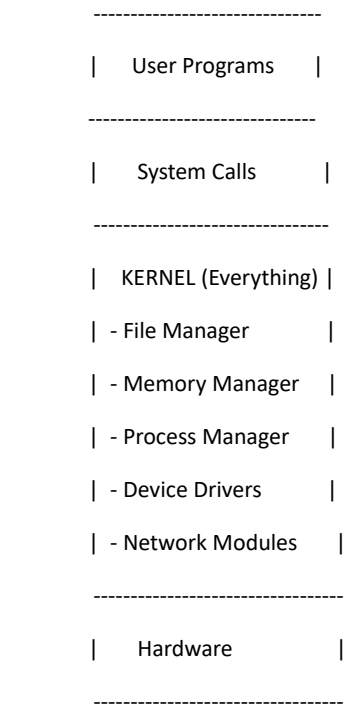
A monolithic OS is an operating system where **all OS services run inside the kernel** (the main part of OS).

The kernel contains:

- File system
- Memory management
- Device drivers
- CPU scheduling
- System calls
- Network drivers

Everything runs together in **one big unit**.

Simple Diagram (Monolithic OS)



Advantages

- Very fast (because everything runs in kernel mode)
- Less overhead
- Good performance

Disadvantages

- Difficult to maintain
- One bug in a module can crash the whole system
- Not very secure

Examples

- **Linux**
- **UNIX**
- **MS-DOS**

Students can remember:
➞ *Linux and UNIX are monolithic.*

---

2. Layered Operating System

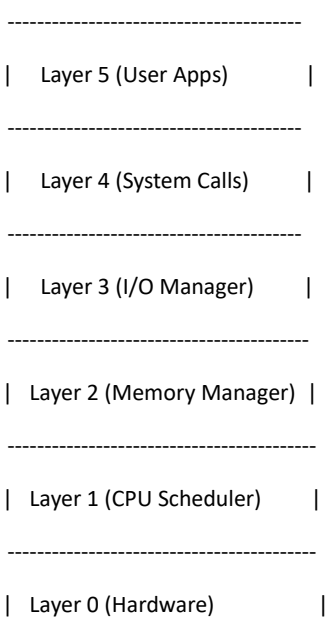
Definition

In this structure, the OS is divided into **layers**, and each layer performs a specific function.

- Lower layers → close to hardware
- Higher layers → close to the user

Each layer uses the services of the layer **below it**.

Diagram (Layered OS)



Advantages

- Easy to design and debug
- One layer does not affect the others
- More organized

Disadvantages

- Slower than monolithic
- A layer may need to wait for services from lower layers

Examples

- **THE Operating System** (famous academic OS)
- Early versions of Linux also had some layered concepts

Students can remember:
➞ *Layered OS = layers built one on top of another*

---

3. Microkernel Operating System

Definition

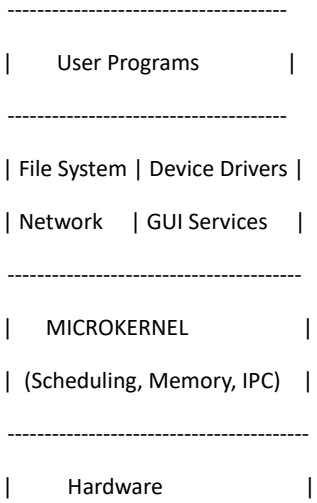
Microkernel OS keeps **only essential services in the kernel**:

- CPU scheduling
- Interprocess communication (IPC)
- Basic memory management

Everything else runs in **user space**, such as:

- File system
- Device drivers
- Network protocols

Diagram (Microkernel OS)



Advantages

- Very secure (drivers run outside kernel)
- More stable
- Easier to update individual modules

Disadvantages

- Slower than monolithic (more communication between modules)
- More complex design

Examples

- **macOS** (XNU uses hybrid microkernel)
- **Minix**
- **QNX**
- **Windows NT (partially microkernel)**

Students can remember:
➞ *Microkernel = small kernel + more security*

Concept of Virtual Machine (VM)

A **Virtual Machine (VM)** is software that creates a **virtual (fake) computer inside a real computer**.

It behaves just like a real physical computer:

- It has its own CPU (virtual CPU)
- Its own RAM (virtual memory)
- Its own storage (virtual disk)
- Its own Operating System (Windows / Linux / Android)

But **all of these are simulated using software**, not physical hardware.

---

1. Why Virtual Machines Are Needed?

Virtual Machines solve many problems:

✓ **Run multiple OS on one hardware**

Example: Run Linux inside a Windows laptop.

✓ **For testing software safely**

A virus inside VM cannot harm your actual computer.

✓ **To run old OS**

Example: Running Windows XP for old software.

✓ **For cloud computing**

Companies like AWS, Azure, Google Cloud run millions of VMs for customers.

✓ **Better utilization of hardware**

Instead of one OS using the whole machine, multiple VMs share it.

---

2. How Does a Virtual Machine Work?

A VM is created using a software called **Hypervisor**.

Hypervisor

A Hypervisor is a software that sits between **hardware** and **multiple virtual machines**.
It divides hardware resources and assigns them to each VM.

Types of Hypervisors

Type 1 Hypervisor (Bare-metal)

Runs directly on the hardware (no OS in between).

Examples:

- VMware ESXi
- Microsoft Hyper-V
- KVM
- Xen

Used in **servers, cloud computing, data centers**.

Type 2 Hypervisor (Hosted)

Runs on top of a host operating system (Windows/Linux).

Examples:

- VirtualBox
- VMware Workstation
- Parallels (Mac)

Used in **student laptops, programming labs, testing**.

---

3. Components of a Virtual Machine

✓ **Virtual CPU (vCPU)**

Software-based version of real CPU.

✓ **Virtual Memory**

A portion of RAM allocated for the VM.

✓ Virtual Disk

A file on the host system that acts like a hard disk.
Example: vm-disk.vdi or vm-disk.vmdk

✓ Virtual Network Interface

VM gets its own IP address and connects like a real computer.

✓ Virtual Devices

Keyboard, mouse, USB, display — all simulated.

---

4. How Many VMs Can Run on One System?

It depends on:

- CPU cores
- RAM
- Disk space

Example:
If your laptop has:

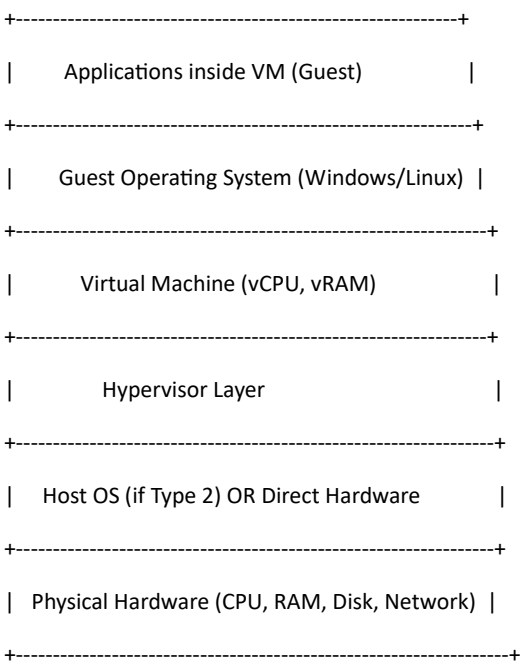
- 8 GB RAM
- 4 core CPU

You can run:

- 2–3 lightweight VMs
- or 1 heavy VM (Windows + software)

---

5. VM Architecture Diagram (text version)



---

6. Advantages of Virtual Machines

✓ Run multiple OS on one computer
✓ Safe testing environment
✓ Easy backup and cloning
✓ Better hardware utilization
✓ Isolation (one VM crash does not affect others)

---

7. Disadvantages of Virtual Machines

Slow compared to physical machine
Needs large RAM and CPU
Heavy applications may lag
Requires virtualization support in CPU (VT-x / AMD-V)

---

8. Real-Life Examples

Example 1: Students

Installing Ubuntu in VirtualBox on Windows.

Example 2: Developers

Testing apps on Windows, Linux, macOS using VMs.

Example 3: Cloud Computing

AWS EC2 instances are Virtual Machines running on huge servers.

Summary (for exam)

A Virtual Machine is **software that behaves like a complete computer**.
It uses a **hypervisor** to share physical hardware resources.
VMs allow multiple OS to run simultaneously, give isolation, and are widely used in cloud computing and testing.