**Chapter 1**

**Operating System**

**Operating System**

A program that acts as an intermediary between a user of a computer and the computer hardware

**Goals of an Operating System**

**1. Run user programs**

* **How:** The OS loads your program into memory, gives it CPU time, and makes sure it can use input/output devices.
* **Example:** When you play a song in **VLC Media Player**, the OS loads it, gives CPU and RAM, and lets it use speakers to play sound.

**2. Make problem-solving easier**

* **How:** The OS provides built-in services like file management, copy/paste, networking, and device drivers.
* **Example:** You don’t need to know how a hard disk works physically – the OS just lets you “save” and “open” files with simple commands.

**3. Convenience**

* **How:** The OS gives you user-friendly interfaces like GUI (Graphical User Interface) with windows, icons, menus, or CLI (Command Line Interface) for text commands.
* **Example:** Instead of typing long commands to print, you just click “Print” in Word, and the OS handles the printer.

**4. Efficiency**

* **How:** The OS manages resources smartly through scheduling, memory management, and multitasking so that no hardware sits idle unnecessarily.
* **Example:** While you’re listening to music, downloading a file, and typing a document, the OS divides CPU time and memory so all three run smoothly.

**Computer System Structure**

**1. Hardware**

* The physical parts of the computer.
* Provides the basic resources needed for computing.
* **Examples:** CPU (processor), memory (RAM), and I/O devices (keyboard, mouse, monitor, printer, etc.).

**2. Operating System (OS)**

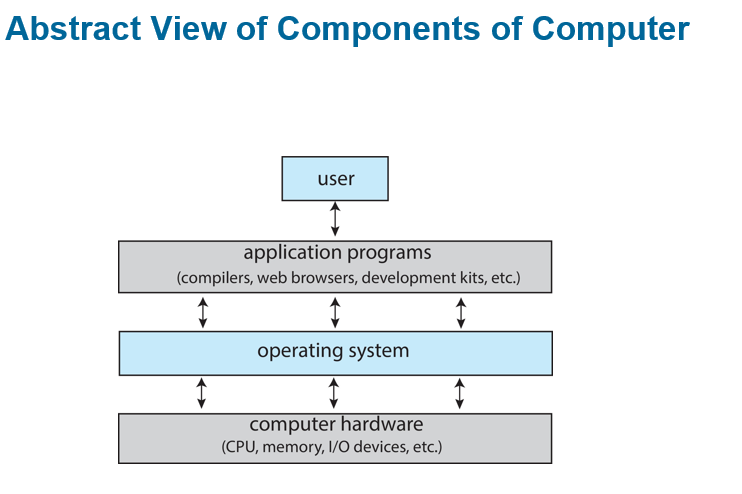
* The manager that controls and coordinates how the hardware is used.
* Make sure hardware is shared fairly between different applications and users.

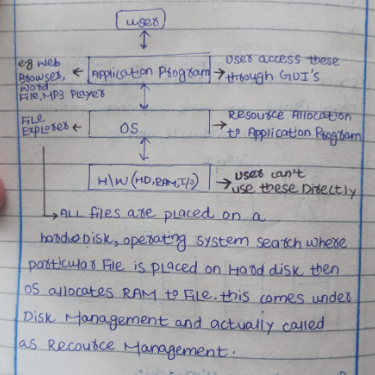
**3. Application programs**

* Application programs are software that directly helps the user perform tasks. They are not a part of the operating system itself, but they run on top of it and use its services, like file handling, memory, and networking.
* **Example:** Word processors (MS Word, Google Docs → for typing documents), Web browsers (Chrome, Firefox → for using the internet, Games (Free Fire, PUBG)

**4. Users**

* The people or systems that use the computer.
* **It could be** humans, machines (like robots), or even other computers connected in a network.



****

**What Operating Systems Do**

The role of an Operating System (OS) can look different depending on the situation:

**1. From a User’s View**

* Normal users just want the computer to be easy to use, fast, and convenient.
* They usually don’t care how the computer is using its internal resources.

**2. Shared Computers (Mainframes, Minicomputers)**

* But shared computers such as mainframe or minicomputer must keep all users happy

**3. OS as a Resource Manager**

* The OS is like a resource allocator (manager).
* It makes sure hardware (CPU, memory, devices) is used efficiently.
* Also, it is a control program that manages the execution of user programs.

**4. Dedicated Systems (Workstations, PCs)**

* Each user usually has their own computer resources.
* But sometimes, they also share resources from servers (like printers, cloud storage, or networks).

**5. Mobile Devices (Smartphones, Tablets)**

* These are resource-limited (less powerful than PCs).
* The OS is designed to:
  + Save battery life
  + Be very easy to use
* Interfaces often include touch screens, gestures, and voice recognition.

**Embedded Computers (Little or No User Interface)**

* Some computers are built inside devices (not like normal PCs).
* They usually have no screen, keyboard, or direct user interface.
* They are programmed to do specific tasks automatically.
* They work in the background without the user needing to control them.
* **Example:** Washing machine – runs wash cycles automatically once you set it.

**Defining Operating Systems**

**Many Roles of OS**

* The term Operating System (OS) covers many different roles because computers are used in so many ways.

**Different Uses**

OSes are found everywhere:

* Small devices like toasters, TVs, game machines.
* Chips inside electronics.
* Big systems like spacecraft and industrial machines.

**Origin of OS**

* At first, computers were made for fixed tasks (like military use).
* Later, computers became general purpose (for many users and tasks).
* This created the need for an OS to manage resources (CPU, memory, devices) and control programs.

**Operating System Definition**

**No Single Definition**

There is no universally agreed definition of what exactly an Operating System (OS) is.

**Practical View**

* A simple way to think of it:  
  **“Everything a company gives you, when you buy an OS.”**
* But this can be very different for each OS (Windows, Linux, Android, etc.)

**OS main Parts**

**Kernel (Core Part)**

The kernel is the most important part of an operating system. It is like the core or brain of the OS. The kernel is always present in the computer’s memory while the system is running, unlike other programs that may start and stop. Its main job is to control and manage the hardware, such as the CPU, memory, and input/output devices. In simple words, the kernel acts as a bridge between the hardware and the rest of the software, making sure everything works smoothly together.

**System Program**

System programs are programs that come with the Operating System to help manage the computer and make it easier for users to use.

**Example:** Text editors (Notepad, vi), Compilers and assemblers (to translate code)

**Application programs**

Application programs are software that directly helps the user perform tasks. They are not a part of the operating system itself, but they run on top of it and use its services, like file handling, memory, and networking.

**Example:** Word processors (MS Word, Google Docs → for typing documents), Web browsers (Chrome, Firefox → for using the internet, Games (Free Fire, PUBG)

**Computer-System Operation**

1. **CPU and Devices Work Together**

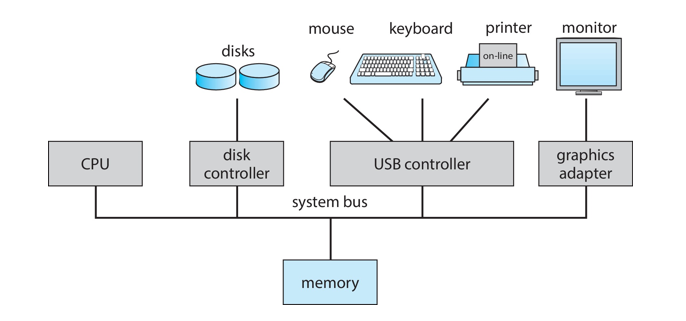
* A computer has **one or more CPUs** (the “brain” of the computer) and **device controllers** (like those for the keyboard, mouse, printer, or hard drive).
* All of these connect using a **common bus**, which is like a shared highway for data.

1. **Shared Memory**

* The CPU and devices use this bus to access the **same memory**. Think of memory as a big storage room that everyone can use.

1. **Concurrent Execution**

* The CPUs and devices can work at the **same time** (concurrently).
* But since they share the memory, sometimes they **compete** for memory access. Only one can use a part of memory at a time, so they take turns very fast.



**I/O devices and CPU Concurrency Concept**

**Concurrency**

Concurrency means that two or more tasks can happen during the same time. It doesn’t mean they must run at the exact same second, but their work overlaps.

**CPU and I/O Devices Working Together**

The CPU is the brain of the computer that runs instructions and calculations. On the other hand, I/O devices (like the keyboard, disk, printer, or network card) handle input and output. Both the CPU and I/O devices can work at the same time. For example, while the CPU is busy running a program, an I/O device can be transferring data in the background.

**How Concurrency Happens**

Here’s how it usually works:

1. The CPU starts a task and then requests input/output (for example, reading data from the disk).
2. Instead of sitting idle and waiting, the CPU continues doing other work.
3. Meanwhile, the I/O device is still completing its task in the background.
4. Once the I/O is finished, the CPU takes the data and uses it.

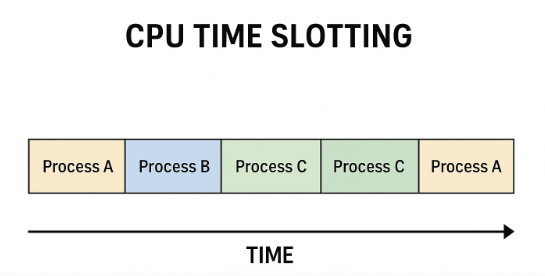
In this way, the CPU and I/O devices overlap their work, which is called concurrent execution.

**Illusion of Multitasking and CPU Time Slotting**

The **illusion of multitasking** happens because a single CPU can only execute one instruction at a time. However, the operating system makes it **look like multiple programs are running simultaneously**. This is where **CPU time slotting**, or time slicing, comes in.

Time slicing divides the CPU’s time into small segments called **time slices**, and each process gets a turn to use the CPU for a short period. When the time slice is over, the CPU quickly switches to the next process. This switching happens so fast that the user perceives all programs as running at the same time, even though the CPU is actually handling them **one by one**.

In short, **CPU time slotting is the mechanism**, and the **illusion of multitasking is what the user experiences**. Without time slicing, the CPU would run one process to completion before moving to the next, and the illusion would disappear.



**Note: -** Every Program Load in RAM before execution

**Device Controller and Device Driver**

**Device Controller**  
A **device controller** is a small hardware part (chip or circuit) inside the computer. It works like a **middleman** between the computer system and a hardware device (such as a keyboard, mouse, hard disk, or printer). The operating system sends instructions to the device controller, and the controller then passes those instructions to the actual device. In simple words, the device controller directly controls how the device works and manages its physical operations.

**Device Driver**  
A device driver is a **software component** that acts as an intermediary between the operating system (OS) and the hardware device. It works like a **translator**, so the OS doesn’t need to worry about the technical details of the device. Instead, the driver provides a simple way for the OS to give instructions.

**Note**

* The OS does not directly talk to the device controller.
* The Device Driver (software) first takes the OS instructions and translates them into commands the Device Controller (hardware) can understand.
* The Device Controller then passes those commands to the actual device and manages how it works physically.

**Printer Example**

1. You press Print on your computer.
2. The printer driver (software) takes your document and turns it into special commands that the printer understands.
3. These commands are sent to the device controller (hardware chip inside the printer or computer).
4. The controller then handles the physical work — moving the paper, spraying the ink, and producing the printout.

**Flow**

****

**Computer-System Operation**

**1. CPU and devices can work at the same time** → The computer’s brain (CPU) and devices (keyboard, printer, etc.) don’t have to wait for each other; they can do their jobs together.

**2. Each device has its own controller** → A controller is a small hardware part that manages only one type of device (like disk controller, printer controller).

**3. Controller has a local buffer** → A small temporary memory inside the controller, used to hold data for a short time.

**4. Controller has a device driver** → A software helper that lets the operating system talk to the device easily.

**5. CPU moves data using buffers** → Data goes between the CPU/main memory and the controller’s buffer, not directly to the device.

**5. I/O happens through the buffer** → Devices send or receive data only from their controller’s buffer.

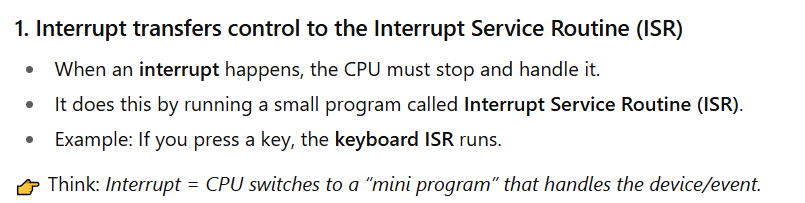
**6. Controller interrupts CPU when done** → After finishing a task (like printing a page), the controller sends a signal (interrupt) to tell the CPU “I’m done.”

**Interrupt and Trap**

**Interrupt**

An interrupt is a signal generated mainly by I/O devices to inform the CPU that they need atgtention — either because the device has finished its work (like data transfer complete) or because an error has occurred (like printer out of paper or disk error).

**Common functions of Interrupt**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Trap**

A trap is a special kind of software-generated interrupt. Unlike normal hardware interrupts (which come from devices like the keyboard or printer), a trap is triggered by the CPU itself while running a program.

**Why does a Trap happen?**

Traps usually happen for two reasons:

1. **Errors (Exceptions):**

* **Example:** Division by zero, invalid memory access, or overflow.
* The CPU raises a trap to alert the OS that something went wrong.

1. **System Calls (Intentional):**

* A program can purposely use a trap to request a service from the OS.
* **Example:** A program wants to read from a file → it uses a trap to ask the OS for help.

**How it Works**

1. Program runs and a condition occurs (error or system call).
2. The CPU generates a trap signal.
3. Control is transferred to the Trap Handler (special routine in the OS).
4. The OS handles the request or error.
5. Program either continues (if possible) or is stopped (if it crashes).

**In short:**

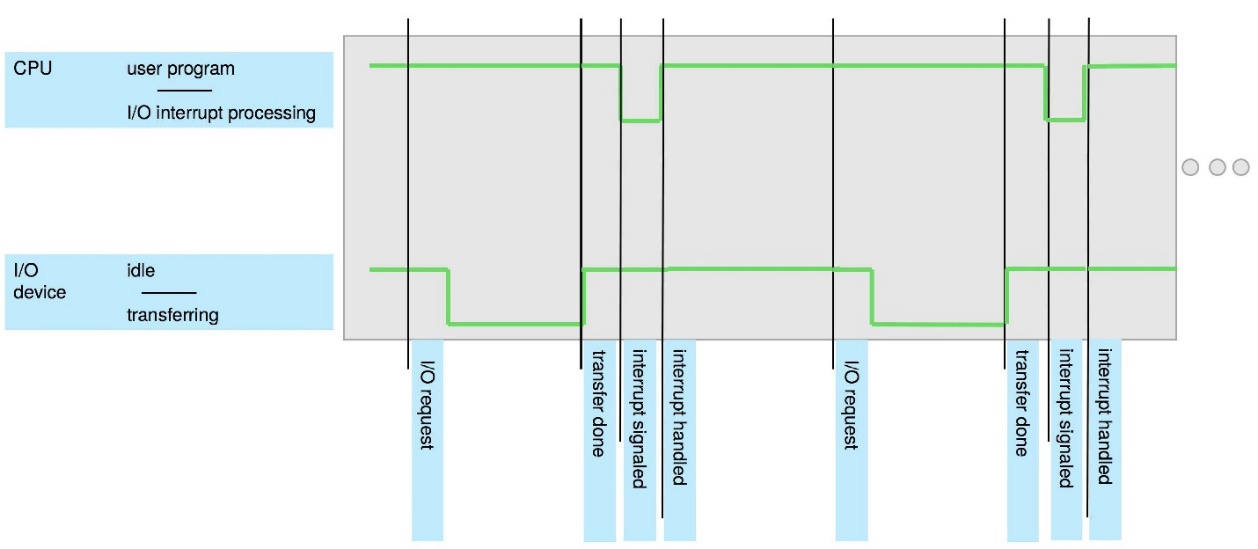
**Interrupt =** from outside (hardware like keyboard).

**Trap =** from inside (software error or request).

**A white text with black text

AI-generated content may be incorrect.**

**Interrupt Timeline**



**Interrupt Handling**

**1. Save CPU’s Current Work**

When an interrupt occurs, the operating system first saves the CPU’s current state (registers and program counter). This allows the CPU to return to exactly **where it left off** after the interrupt is handled.

**2. Identify the Interrupt Type**

Next, the OS checks the **type of interrupt** to find out who is requesting service — for example, the keyboard, printer, disk, or even a software error.

**3. Run the Correct ISR**

Once the source is identified, the OS runs the correct **Interrupt Service Routine (ISR)**. Each device or event has its own ISR, which contains the instructions to handle that specific request.



**Two methods for handling I/O**

**Synchronous I/O (Wait for completion)**

* When an I/O (input/output) operation starts, the program pauses.
* The program continues only after the I/O is finished.
* **Example:** If you ask a printer to print, your program will wait until printing is done before moving on.

**Asynchronous I/O (Don’t wait)**

* When an I/O operation starts, the program does not wait.
* The program continues running while I/O happens in the background.
* **Example:** If you ask a printer to print, your program keeps working on other tasks while the printer is still printing.

**Continue…**

**Method 1: Control returns after I/O completion (Synchronous I/O)**

* The program waits until I/O is finished.
* CPU can sit idle (doing nothing) until an interrupt comes.
* Sometimes the CPU keeps checking repeatedly (busy waiting), wasting time.
* Only one I/O request can happen at a time → no parallel I/O.
* **Example:** You click “Download” and your computer just freezes until the file finishes downloading.

**Method 2: Asynchronous I/O (Don’t Wait for I/O)**

* Here, the program does not wait after starting I/O.
* The program can keep running other work immediately.
* If later the program needs the I/O result, it asks the OS to check whether it’s ready.
* The OS keeps a device-status table, which stores info like:
  + Which device is (type)
  + Where it is (address)
  + What is it doing (busy or ready)
* The OS updates this table whenever an interrupt occurs.
* **Example:** You click “Download” and while the file is downloading, you can still use your computer to type or browse.

**Computer Startup**

* When you turn on (power-up) or restart (reboot) the computer, a small program called the Bootstrap Program runs first.
* This program is stored in ROM/EPROM (a chip inside the computer). Because it’s permanent, it’s also called firmware.
* The **bootstrap program’s job** is to prepare the computer:
* It checks and sets up the hardware (CPU, memory, devices).
* Then it loads the operating system (OS) kernel into memory.
* Finally, it starts the OS so the computer becomes ready to use.

**Storage Structure**

**Main Memory (RAM)**

* The CPU can directly use main memory.
* It is fast and allows random access (CPU can jump to any location quickly).
* It is usually volatile → data is lost when the power is off.
* Normally it is DRAM (Dynamic RAM).

**Secondary Storage**

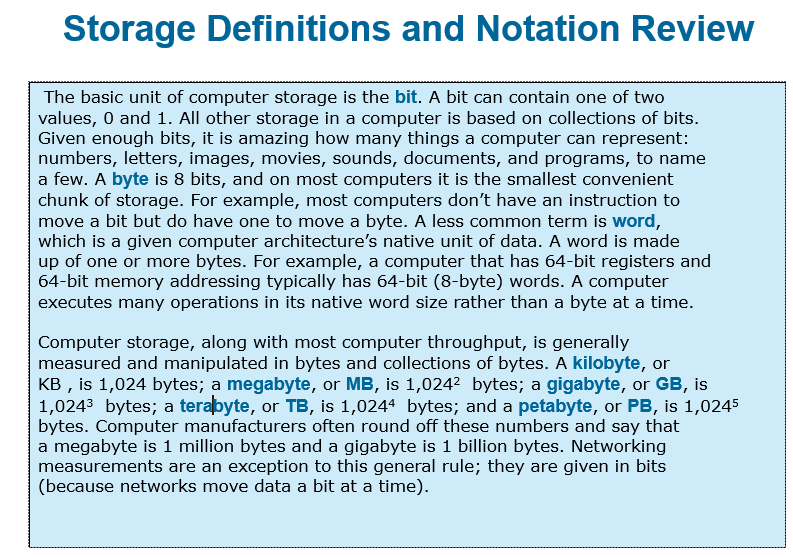
* Works as an extension of main memory.
* Stores much larger amounts of data than RAM.
* It is non-volatile → data stays even after the power is off.
* **Examples:** Hard disk, SSD, USB drive.

**Hard Disk Drives (HDD)**

* Made of metal or glass plates coated with magnetic material.
* The surface of the disk is divided into tracks (big circles) and then into sectors (small parts of a track).
* A disk controller manages how the computer and the disk talk to each other.
* HDD is non-volatile, meaning data stays even when power is off.

**Non-Volatile Memory (NVM) Devices (like SSDs, Flash drives)**

* Faster than hard disks.
* Data is also non-volatile (doesn’t disappear when power is off).
* Made with different technologies (flash memory, SSDs, etc.).
* Becoming more popular because they are getting cheaper, bigger in size, and faster.

****

**Storage Systems Organized in a Hierarchy**

Computer storage is arranged in levels (hierarchy) based on:

1. **Speed →** Faster storage is usually smaller.
2. **Cost →** Faster storage is more expensive.
3. **Volatility →** Some storage (like RAM) loses data when power is off, while others (like hard disks) keep data.

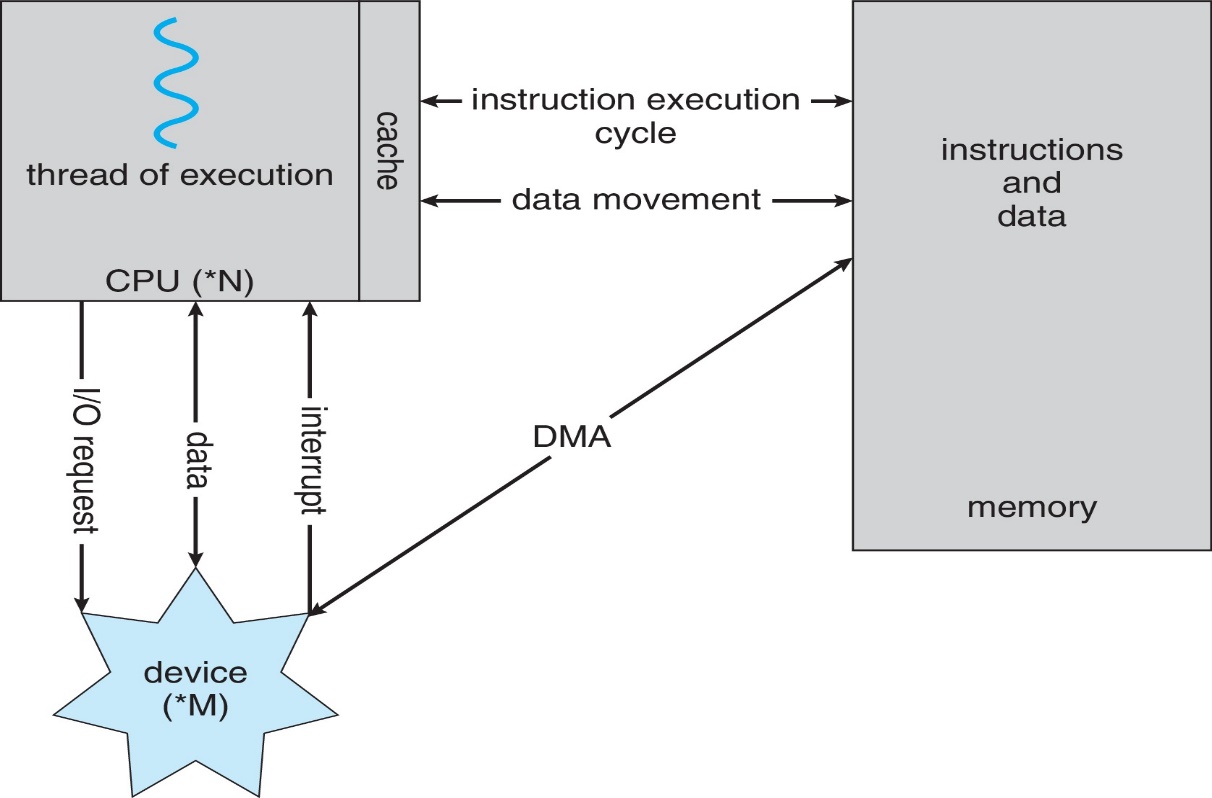
**Cache**

* **Caching =** making a copy of data into faster storage for quick access.
* **Example:** Main memory (RAM) works as a cache for secondary storage (hard disk/SSD).

**A diagram of a storage device

AI-generated content may be incorrect.**

**How a Modern Computer Works**



**Diagram Illustration**

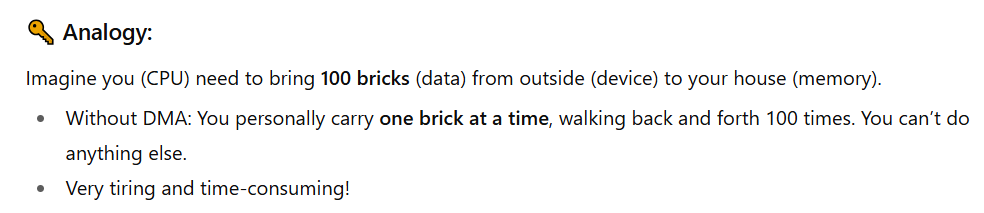
**A screenshot of a computer program

AI-generated content may be incorrect.**

**Direct Memory Access (DMA) Structure**

* Used for fast devices (like disk drives, graphics cards, etc.) that can send/receive data very quickly.
* The device controller (like a helper) moves a whole block of data directly from the device to main memory, without asking the CPU to move every single piece.
* Instead of disturbing the CPU for every byte, the CPU only gets one signal (interrupt) when the whole block of data is done.

**Analogy:**It’s like instead of a delivery guy ringing your bell for every single item, he delivers the entire package at once and rings the bell only once.

****

**Operating-System Operations**

1. **Bootstrap program**

* A very small program stored in ROM.
* Its job: turn on the computer, check hardware, and load the operating system kernel into memory.

1. **Kernel loads**

* The kernel (core part of the OS) is placed into memory.
* Now the OS can start working.

1. **System daemons (background services)**

* These are small helper programs that run in the background (like managing printing, networking, etc.).
* They are outside the kernel but essential.

1. **Interrupts (kernel is interrupt-driven)**

The kernel responds to interrupts (signals that something happened).

**Two types:**

* Hardware interrupt: comes from devices (like keyboard press, disk ready, network packet arrived).
* Software interrupt: comes from programs.
  + Could be an error (like division by zero).
  + Or a request for OS help (called a system call).

1. **Other problems the OS must handle**

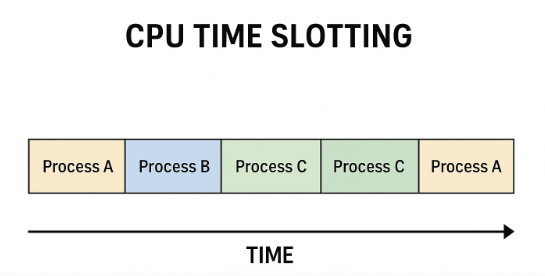
* Program stuck in an infinite loop.
* A program trying to modify another program or even the OS itself (dangerous).

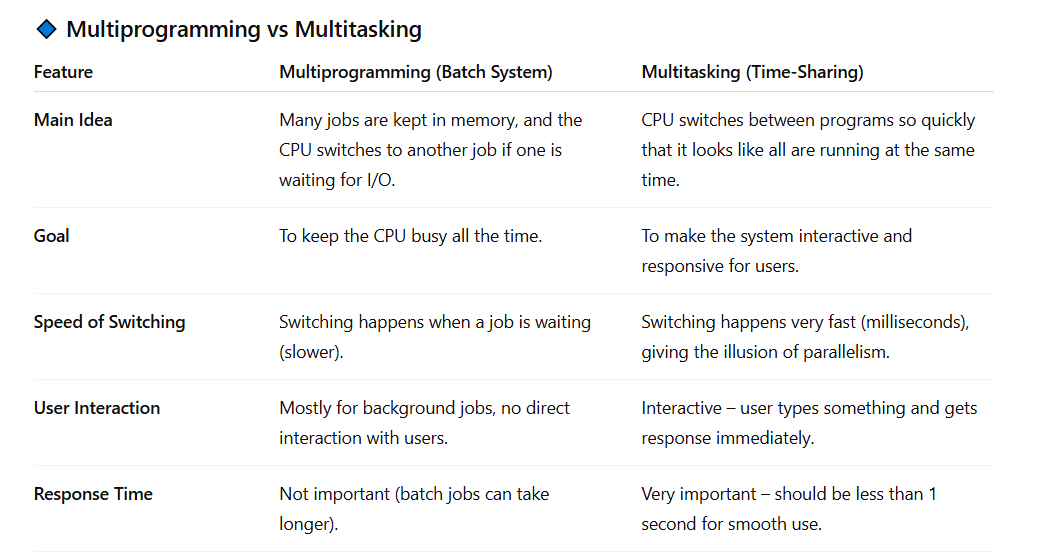
**Multiprogramming (Batch System)**

In a simple system, a single program cannot keep both the CPU and I/O devices busy all the time. For example, when one program is waiting for input/output, the CPU would sit idle. Multiprogramming solves this by keeping many jobs in memory at the same time. The operating system uses job scheduling to choose which job to run. If one job has to wait for input/output, the OS quickly switches to another job. In this way, the CPU is never left idle because it always has some work to do.

**Multitasking (Time-Sharing)**

Multitasking is a more advanced form of multiprogramming. Here, the CPU switches between programs very quickly — so fast that it looks like all the programs are running at the same time. This makes the computer interactive, meaning when you type something, the system responds almost instantly. For a smooth experience, the response time should be less than one second. In multitasking, each user has at least one program (process) in memory. If many jobs are ready, the operating system uses CPU scheduling to decide which one should run. When memory is not enough, the OS uses swapping (moving processes in and out of memory). Virtual memory also helps by allowing programs to run even if they don’t completely fit into the real physical memory.



****

**A screenshot of a computer

AI-generated content may be incorrect.**

**Dual-Mode Operation**

**1. Two modes of CPU operation**

* User mode → normal programs run here (games, apps, etc.).
* Kernel mode → operating system runs here (has full control of hardware).

**2. Mode bit**

* A special bit in hardware tells whether the CPU is in user mode or kernel mode.
* If mode bit = user → program cannot do dangerous things.
* If mode bit = kernel → OS has full power.

**3. Why?**

To protect the system:

* Accidentally or intentionally **crash the operating system**.
* Directly **control hardware** (like disk, memory, CPU instructions) and mess things up.

**4. How is mode changed?**

A user program cannot change mode directly (for safety).

Only the OS can do it.

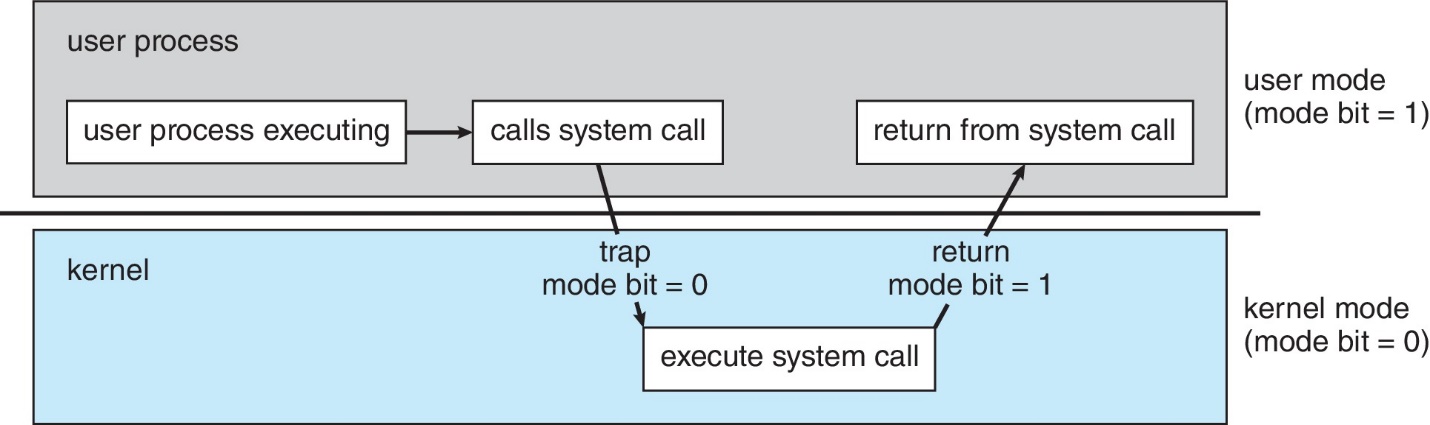
**Example:**

* When a program makes a system call (like asking OS to read a file), the CPU automatically switches to kernel mode.
* After finishing, the OS switches back to user mode.

**5. Privileged instructions**

* Some special instructions (like controlling I/O, changing mode bit) are marked as privileged.
* These can only run in kernel mode.
* If a user program tries to run them, the CPU will stop it (trap/exception).

**Transition from User to Kernel Mode**



**Link Between System Calls and Dual-Mode**

Programs run in **user mode** where they cannot directly access hardware or perform dangerous operations.

When a program needs something special (like reading a file, printing, or allocating memory), it makes a **system call**.

The **system call triggers a switch from user mode to kernel mode**.

In **kernel mode**, the OS performs the requested task because only the OS has full hardware access.

After finishing, the CPU switches back to **user mode**, and the program continues.

**In short:**

* **System call =** the “door” to enter kernel mode.
* **Dual mode =** ensures only the OS (in kernel mode) can do privileged tasks.

**Timer in Operating System**

The timer in an operating system is used to make sure that no single program runs forever and wastes CPU time. The operating system sets a counter that decreases with every clock tick. When this counter reaches zero, the CPU creates an interrupt that stops the program and gives control back to the OS. Only the operating system can set this timer because it is a privileged job. The timer is always set before giving control to a process, so the OS can take the CPU back when the time ends or stop the process if it misbehaves. This helps prevent CPU hogging and keeps the system safe and responsive.

**Process Management**

**What is a Process?**

A process is simply a program that is running. A program stored on disk is **passive entity** (inactive), but when it has been assigned RAM and starts executing, it becomes an **active process.** It is the basic unit of work inside the computer system.

**Resources Needed by a Process**

To finish its task, a process needs some resources:

* The CPU to run instructions
* Memory to store data
* Input/Output devices like keyboard, screen, or printer
* Files
* Some initial data to start with

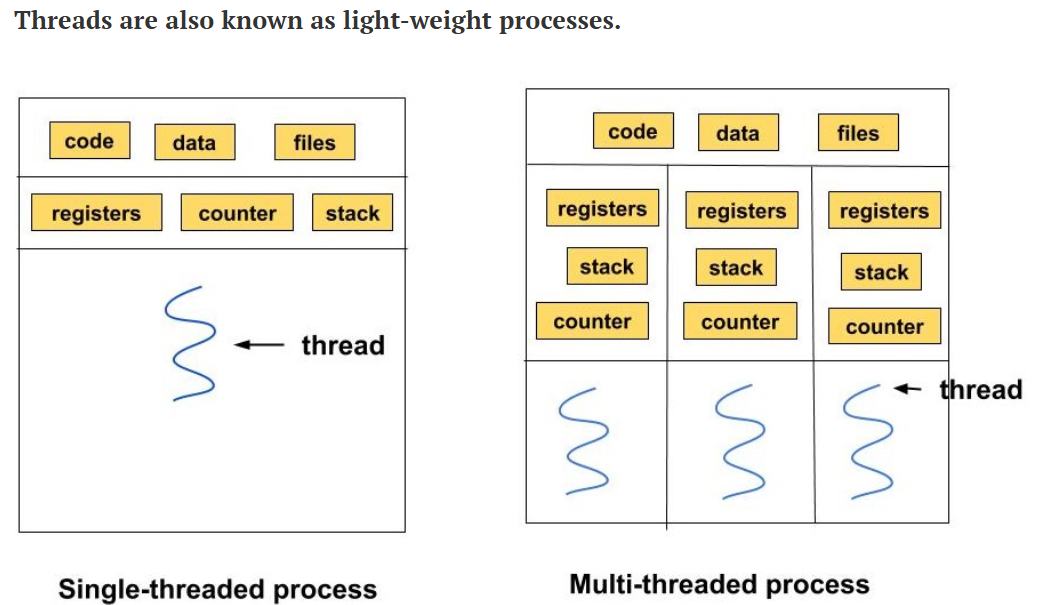
**When a Process Ends**

Once the process finishes, the operating system takes back all the resources it was using so they can be used again by other processes.

**Single-Threaded and Multi-Threaded Processes**

A single-threaded process has only one sequence of instructions. It has one program counter that shows which instruction to run next, and it executes them one by one until the end.

A multi-threaded process has more than one thread inside it. Each thread has its own program counter, which means different parts of the process can run at the same time.



**Many Processes in the System**

Usually, many processes are running together in a system. Some belong to the **user** (like apps), and some belong to the **operating system** its

**Concurrency**

The CPU is shared among these processes using **multiplexing**. This means the CPU switches quickly between processes or threads, giving the feeling that they are running at the same time.

**Process Management Activities**

The **operating system (OS)** takes care of many jobs related to processes, such as:

**1. Creating and Deleting Processes**

The OS can **create new processes** (both user programs and system programs) and also **end/delete processes** when they are finished.

**2. Suspending and Resuming Processes**

The OS can **pause (suspend)** a process for some time and later **start it again (resume)** when needed.

**3. Process Synchronization**

When many processes run at the same time, sometimes they need to use the same resource (like a file, printer, or memory). If they don’t take turns properly, they might interfere with each other or create wrong results.

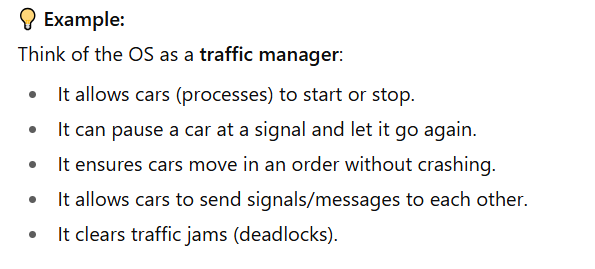
The OS provides rules so that processes wait for each other and work in the correct order without clashing.

**4. Process Communication**

Processes may need to **exchange data or messages** with each other. The OS provides safe methods for this communication.

**5. Deadlock Handling**

Sometimes, processes get **stuck waiting for each other’s resources** (called deadlock). The OS has methods to **detect, prevent, or solve** such deadlocks.



**Memory Management**

**What is Memory Management?**

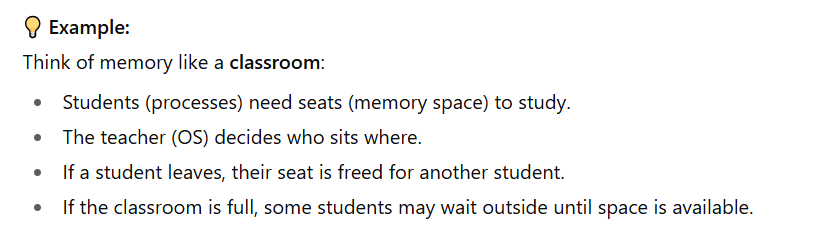
When a program is running, its **instructions** and the **data it needs** must be in the computer’s memory (RAM).  
Memory management is the job of the operating system that decides:

* What goes into memory,
* When it goes in, and
* When it should be removed.

The goal is to **use the CPU efficiently** and give **fast responses** to users.

**Activities of Memory Management**

1. **Keeping Track of Memory**  
   The OS keeps a record of which parts of memory are in use and which parts are free, as well as which process is using them.
2. **Deciding Movement of Processes/Data**  
   The OS decides which processes (or their parts) and which data should be moved into memory and which can be taken out (for example, moved to hard disk temporarily).
3. **Allocating and Deallocating Memory**  
   The OS gives memory space to processes when they need it (allocation) and takes it back when they are done (deallocation).



**File-System Management**

**What OS Does for Storage**

The operating system (OS) gives users a **simple and uniform way** to handle data, no matter what type of storage device is used (hard disk, SSD, CD, tape, etc.).  
It hides the physical details and shows everything as **files**.

* Each device (disk drive, tape drive, etc.) has different properties like:
  + **Speed** (how fast data can be read/written),
  + **Capacity** (how much data it can store),
  + **Data-transfer rate** (how quickly data moves),
  + **Access method** (sequential = one by one, random = direct jump).

**Files and Directories**

* Files are usually grouped into **directories (folders)** to stay organized.
* **Access control** is used to decide **who can open, edit, or delete** files.

**Activities of the OS in File-System Management**

The OS is responsible for:

1. **Creating and Deleting** files and directories.
2. Giving **commands (primitives)** to read, write, or modify files and directories.
3. **Mapping files** onto the secondary storage (deciding where on the disk they are stored).
4. Making **backups** of files onto safe storage (non-volatile media like another disk, cloud, or tape) so data isn’t lost.

**Mass Storage Management**

**What is Mass Storage?**

* Mass storage usually means **disks** (like HDDs or SSDs).
* It is used for data that **doesn’t fit in main memory (RAM)** or data that needs to be stored for a **long time** (e.g., files, software, backups).
* Managing storage properly is very important because the **speed of the whole computer** depends a lot on how well the disk system is managed.

**Activities of the OS in Mass Storage Management**

**1. Mounting and Unmounting**

**Mounting**

* **Mounting** means making a storage device (like a hard disk, SSD, or USB) **ready to use** by the operating system.
* After mounting, you can **see the files** and use the device.

Example:  
When you plug in a **USB drive**, your OS mounts it so you can open it in “My Computer” or “This PC.”

**Unmounting**

* **Unmounting** means safely **removing or disconnecting** the storage device from the OS.
* This ensures that all data is saved properly and nothing is lost or corrupted.

Example:  
When you click **“Safely Remove USB”**, you are unmounting the drive before pulling it out.

**2.** **Free-Space Management**

* The OS keeps track of which parts of the disk are free and which are already used.

**3**. **Storage Allocation**

* The OS decides **where on the disk** to store each file or piece of data.

**4.** **Disk Scheduling**

* When many requests come at once, the OS decides the **order** in which disk operations are done to make things faster.

**5.** **Partitioning**

* Dividing one physical disk into **separate sections** (partitions) that act like independent disks.

**6**. **Protection**

* The OS protects files and data on the disk so that **unauthorized access** is not allowed.

**Caching**

**What is Caching?**

Caching means keeping a **temporary copy of data** in a **faster storage place** so it can be used quickly.

It happens at many levels in a computer:

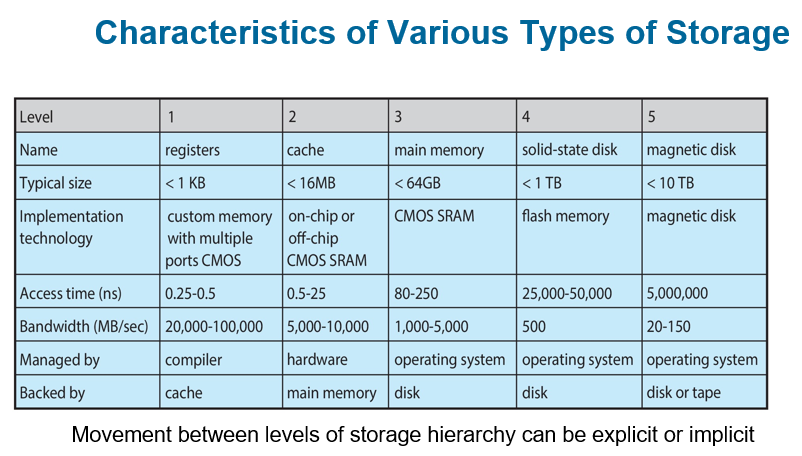
* **Hardware** (CPU cache, disk cache),
* **Operating system**,
* **Software** (like web browsers storing pages).

**How Caching Works**

1. When the computer needs some data, it **first checks the cache** (the fast storage).
2. If the data is **found in the cache** → it is used directly (this is very fast).
3. If the data is **not in the cache** → it is brought from the slower storage into the cache, and then used.

**Important Points**

* The **cache is smaller** than the main storage, so it cannot hold everything.
* Managing the cache is important: the system must decide **which data to keep** and **which data to remove** when the cache gets full.
* This is called a **replacement policy** (e.g., removing the least recently used data).

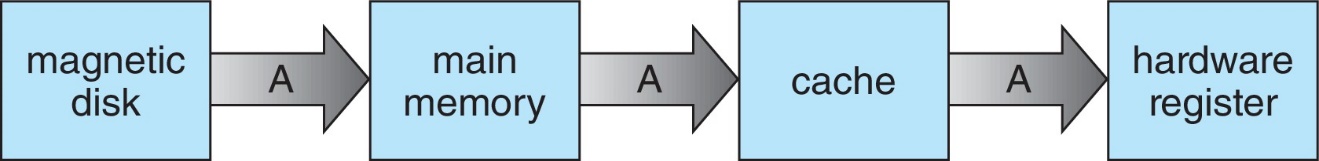


**Migration of Data**

**From Disk to Register**

Data moves step by step in the storage hierarchy:

* Disk → Main Memory → Cache → Register (CPU’s fastest storage).  
  When data “A” is used, it may exist in different places at the same time.



**In Multitasking Systems**

When many programs are running, the system must make sure it always uses the latest (most recent) value of the data, no matter where it is stored (disk, memory, or cache).

**In Multiprocessor Systems**

If the computer has many CPUs, each CPU has its own cache.  
The system must ensure cache coherency → all CPUs should see the same updated value of data “A” in their caches.

**In Distributed Systems**

Things are even more complex because data can exist on different computers connected in a network.  
Several copies of the same data may exist, and the system must keep them consistent (all showing the correct, updated value).

**A computer error message

AI-generated content may be incorrect.**

**I/O Subsystem**

**Purpose**

One job of the Operating System (OS) is to hide the details of hardware devices (like printer, keyboard, mouse, disk) from the user, so users can use them easily without worrying about how they actually work.

**What the I/O Subsystem Does**

1. **Memory Management for I/O**
   * **Buffering →** temporarily storing data while it is being moved (like a waiting area).
   * **Caching →** keeping a copy of frequently used data in faster storage for quick access.
   * **Spooling:** Spooling means lining up tasks in a queue so devices can handle them one by one, while the user or computer can keep working on other things.
2. **General Device-Driver Interface**  
   The OS provides a common way for programs to talk to hardware devices.
3. **Device Drivers**  
   For each specific device (like a printer or graphics card), the OS uses a driver, which is special software that knows how to control that device.

**Protection and Security in OS**

**Protection**

* Protection means controlling who can use which resources in the computer.
* Example: Some users can open a file, but others cannot.

**Security**

* Security means defending the system from attacks (inside or outside).
* Attacks can be things like viruses, worms, hacking, stealing data, or blocking services.

**How the OS manages this**

1. **User Identification (User ID):**

* Every user has a unique ID (like a username with a number).
* This ID is linked to all the user’s files and processes.

1. **Group Identification (Group ID):**

* Users can be put into groups (like "students", "teachers", "admins").
* Permissions can be given to the whole group instead of individual users.

1. **Access Control:**

* The OS checks user IDs and group IDs to decide who can read, write, or run a file or program.

1. **Privilege Escalation:**

* Sometimes, a user temporarily gets more power (more rights) to do special tasks.
* Example: "Run as administrator" in Windows.

**Distributed Systems**

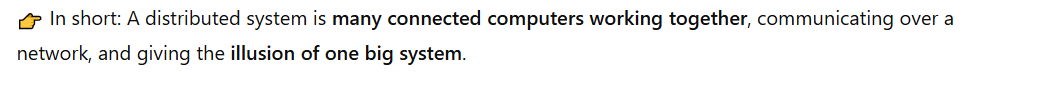
A distributed system is a collection of many computers that are connected to each other through a network. These computers may be different in type or design, but they still work together as a single system. They share tasks and resources to complete work more efficiently. The connection between them is usually done through a network, and the most common way they communicate is by using the TCP/IP protocol, which is also used on the internet.

**Types of Networks**

1. **LAN (Local Area Network):**  
   Small area, like inside your home, school, or office.
2. **WAN (Wide Area Network):**  
   Covers a very large area, like the internet.
3. **MAN (Metropolitan Area Network):**  
   Covers a city or town.
4. **PAN (Personal Area Network):**  
   Very small, around one person (like Bluetooth between your phone and earbuds).

**How it Works**

* A **Network Operating System** helps these computers work together.
* Computers exchange information by sending **messages** over the network.
* To the users, the whole distributed system can feel like **one single computer**, even though many machines are working behind the scenes.



**Computer-System Architecture**

* Most computers have one main processor (general-purpose processor).
* But many systems also use special-purpose processors (like graphics cards, sound processors, etc.).
* These days, multiprocessor systems (computers with more than one processor) are becoming more common and important.
* Multiprocessor systems are also called parallel systems or tightly-coupled systems.

**Advantages of Multiprocessor Systems**

1. **Increased Throughput:** More processors can do more work at the same time → faster performance.
2. **Economy of Scale:** Cheaper than having many separate computers, because they share memory, power, and devices.
3. **Increased Reliability:** If one processor fails, the others can still keep the system running (fault tolerance).

**Types of Multiprocessing**

1. **Asymmetric Multiprocessing (AMP):**

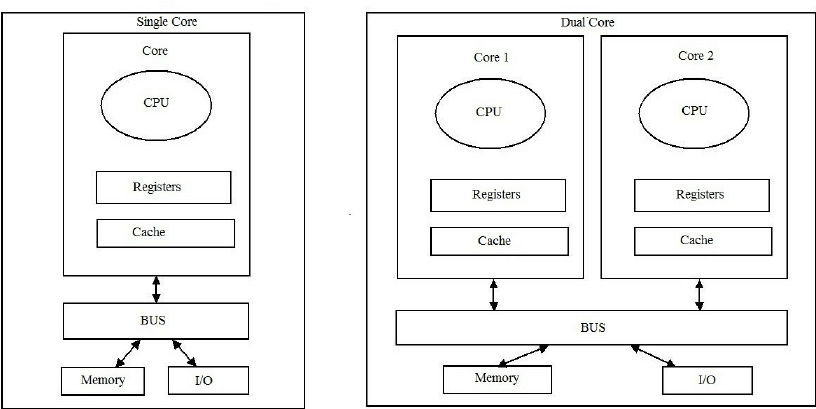
* Each processor has a **specific job**.
* Example: One processor controls the system, while others do assigned tasks.

1. **Symmetric Multiprocessing (SMP):**

* All processors are **equal**.
* Each can do **any task** and share the workload.

A diagram of a computer hardware system

AI-generated content may be incorrect.



**Continue…**

**So,**

* Single-core = concurrency (taking turns).
* Multi-core = parallelism (at the same time).

**Chapter 2**

**Operating-System Services**

**Operating System Services**

An **Operating System (OS)** provides an environment where programs can run and offers different services that make the computer easier to use.

**1. User Interface (UI)**

Almost every operating system has a user interface that allows people to interact with the computer. This interface can appear in different forms:

* **Command-Line Interface (CLI):** users type commands.
* **Graphical User Interface (GUI):** users interact through windows, icons, and menus.
* **Touch-Screen Interface:** users control the system using fingers or gestures.
* **Batch System:** A method where jobs/programs are submitted together and executed automatically in groups with little or no user interaction.

**2. Program Execution**

The operating system helps in running programs. It loads a program into memory, starts its execution, and manages it until the program ends. A program can finish either:

* **Normally,** when the task is completed, or
* **Abnormally,** if some error occurs.

**3. I/O Operations**

Many programs require **input and output (I/O)** operations. This could mean reading and writing files or interacting with devices like the keyboard, printer, or monitor. The operating system manages these I/O tasks safely and efficiently so that programs can work properly.

**4. File System Manipulation**

Files are very important for storing information. The operating system provides services to manage them. With the help of the OS, programs can:

* Create, delete, read, and write files.
* Create and remove directories (folders).
* Search for specific files.
* List file information such as size, type, and date.
* Manage **permissions** to control who can read, write, or execute a file

**Operating System Services (Cont.)**

Apart from user interface, program execution, I/O operations, and file management, the operating system also provides other important services.

**5. Communications**

The operating system allows **processes** (running programs) to share information with each other. This communication can happen:

* **On the same computer**, between two or more processes.
* **Between different computers** connected through a network.

Communication is done in two main ways:

* **Shared Memory:** Processes use the same memory space to exchange data.
* **Message Passing:** Information is sent in the form of packets, which the OS manages and delivers.

**6. Error Detection**

The OS must always be **watchful for errors** to keep the system stable and safe. Errors can happen in:

* **CPU or memory hardware** (like hardware faults).
* **Input/Output devices** (like printer or disk failures).
* **User programs** (like dividing by zero or invalid instructions).

When an error occurs, the OS takes proper action to fix the problem or stop it from affecting other parts of the system.

To help users and programmers, the OS also provides **debugging tools**, which make it easier to find and correct mistakes in programs.

**Operating System Services (Continued)**

Besides helping users run programs and handle files, the operating system also provides services that ensure the **system itself works efficiently** and resources are shared properly.

**7. Resource Allocation**

When many users or programs are running at the same time, the operating system must **divide resources fairly**.

* Resources include the **CPU**, **main memory**, **file storage**, and **I/O devices**.
* The OS decides how much of each resource will be given to each job or user so that all can work smoothly without conflict.

**8. Logging (Accounting)**

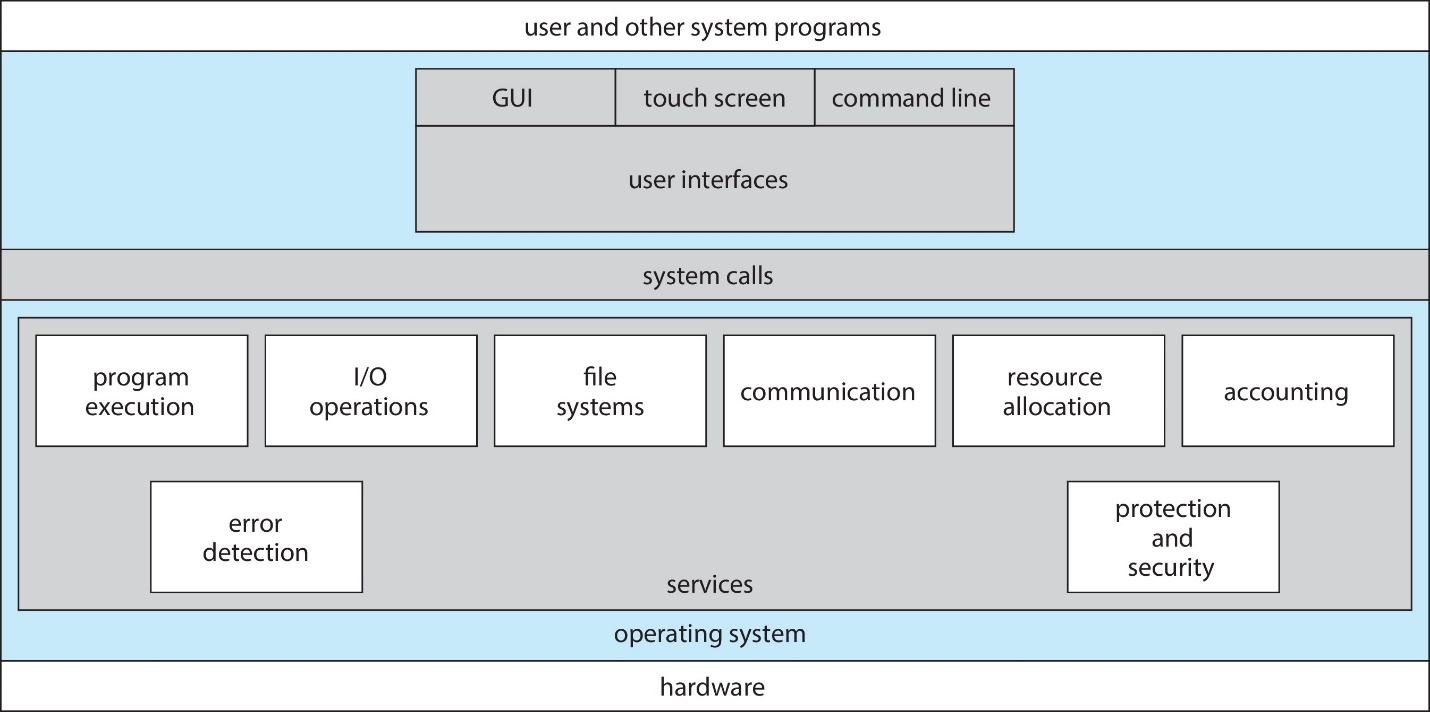
The OS also keeps **records of resource usage**.

* It tracks how much CPU time, memory, or storage a user or program consumes.
* These logs can be used for **billing users** in commercial systems, or just for **statistics** to monitor performance and improve the system.
* **Billing Example:** In cloud services, if a user runs a program that uses the CPU for 5 hours and stores 10 GB of data, the OS logs this usage, and the provider generates a bill based on it.
* **Statistics Example:** If logs show that CPU usage is very high between 9 AM and 12 PM, admins can add more CPUs or memory to improve performance during peak hours.

**9. Protection and Security**

In a multi-user or networked system, it is very important to keep information safe and prevent unwanted access. **Protection** ensures that processes do not interfere with each other and that every access to system resources is properly controlled. On the other hand, **security** focuses on protecting the system from outsiders. This includes methods such as **user authentication** through passwords or biometric logins, preventing unauthorized access to files, memory, and devices, and defending external devices like network ports or USB drives from harmful or invalid attempts. Together, protection and security make the system reliable, safe, and fair for all users.

**A View of Operating System Services**



**Command Line Interpreter (CLI)**

A **Command Line Interpreter (CLI)** allows users to give direct instructions to the computer by typing commands. It can be built directly into the operating system kernel or provided as a separate system program. Many operating systems offer different versions of CLI, which are called **shells** (for example, Bash in Linux or Command Prompt in Windows). The main task of a CLI is to **take a command from the user and execute it**.

Some commands are **built into the shell itself**, such as cd (change directory) or pwd (print working directory). These work directly because the shell already knows how to handle them. Other commands are actually **external programs** stored in the system, such as python, git, or gcc. In this case, the shell only calls these programs when their names are typed. The advantage of this design is that new features can be added simply by installing new programs, without needing to modify the shell itself.

A computer screen with green text

AI-generated content may be incorrect.

**User Operating System Interface – GUI**

A **Graphical User Interface (GUI)** makes the operating system more **user-friendly** by allowing people to interact with the computer through visuals instead of only typing commands. In a GUI, users work with a **desktop-like environment** using a **mouse, keyboard, and monitor**. Files, programs, and actions are shown as **icons**, and clicking or right-clicking on them can open folders, show options, provide information, or run programs.

The idea of GUI was first invented at **Xerox PARC**, and now it is used in almost all modern systems. Many operating systems provide **both CLI and GUI**, so users can choose the one they prefer. For example, **Microsoft Windows** mainly uses a GUI but also has a command-line tool called *Command Prompt*. **Apple’s macOS** provides the “Aqua” GUI on top of a UNIX-based kernel, with command-line shells also available. Similarly, **Unix and Linux** are usually command-line based, but they also offer optional GUIs such as **CDE, KDE, and GNOME**.

A screenshot of a computer

AI-generated content may be incorrect.

**Touchscreen Interfaces**

Touchscreen devices need special interfaces because using a mouse is either not possible or not convenient. In these systems, actions and selections are done through **gestures** such as tapping, swiping, or pinching on the screen. For typing, a **virtual keyboard** appears on the screen instead of a physical one. Many touchscreen interfaces also support **voice commands**, making it even easier for users to interact with the system without needing extra devices

A screenshot of a cell phone

AI-generated content may be incorrect.

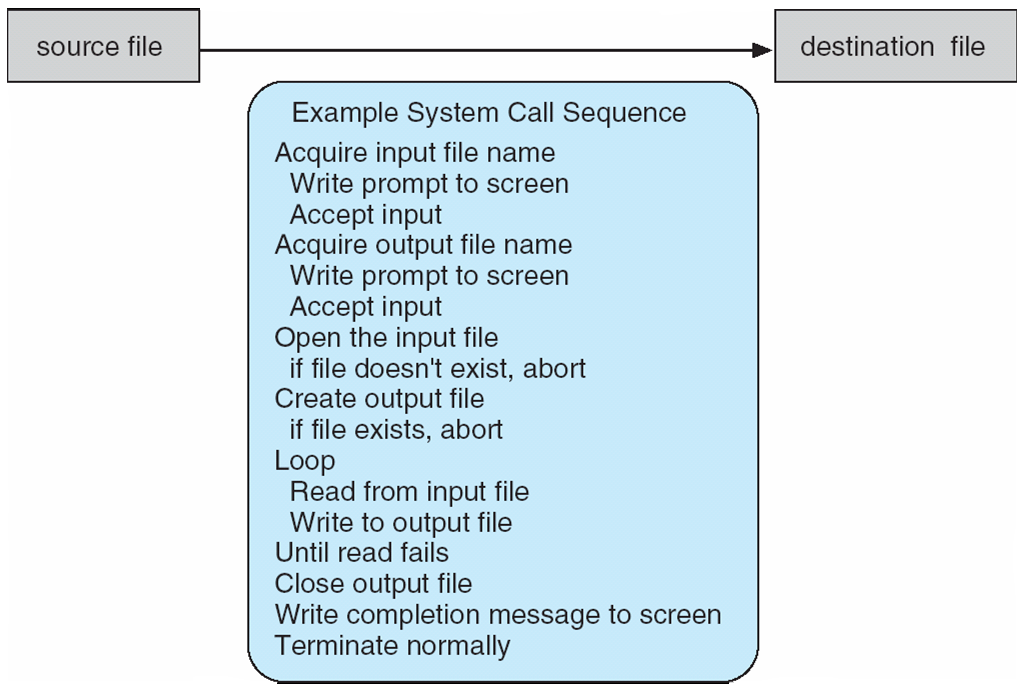
**System Calls**

System calls act as a bridge between a program and the operating system (OS), allowing the program to use the services provided by the OS, such as reading files, creating processes, or managing hardware. They are usually written in high-level programming languages like C or C++. However, most programs do not directly use system calls; instead, they access them through an easier interface known as an Application.

**Link between System Call and APIs**

APIs act as an easier interface that applications use to access system calls.  
The API functions you use in programming (like printf() in C or readFile() in JavaScript) internally make system calls to ask the operating system to perform tasks such as reading a file or displaying output.

**Example of System Calls**



**Application Programming Interface (API)**

An **API (Application Programming Interface)** is a set of rules and protocols that allows one software application to communicate and interact with another. It defines how different software components should exchange information, making it easier for them to send requests and receive responses in a well-structured way.

**Example**

For example, in **data access**, APIs enables applications to retrieve or send data to other systems — like a weather app using an API to get the latest weather updates from an online server.

**Commonly Used APIs**

There is also **Web APIs (or HTTP APIs)** that use web protocols such as **HTTP** to transfer data, usually in formats like **JSON** or **XML**. Common examples of web APIs include **REST APIs** and **GraphQL APIs**, which are widely used for web and mobile app development

**Example of Standard API**

A screenshot of a computer program

AI-generated content may be incorrect.

**System Call Implementation**

Each system call has a unique number assigned to it. The operating system keeps a **table** that matches these numbers to the correct system call functions. When a program makes a system call, the **system call interface** looks up this number in the table, runs the correct function inside the **OS kernel**, and then sends back the **result or status** to the program.