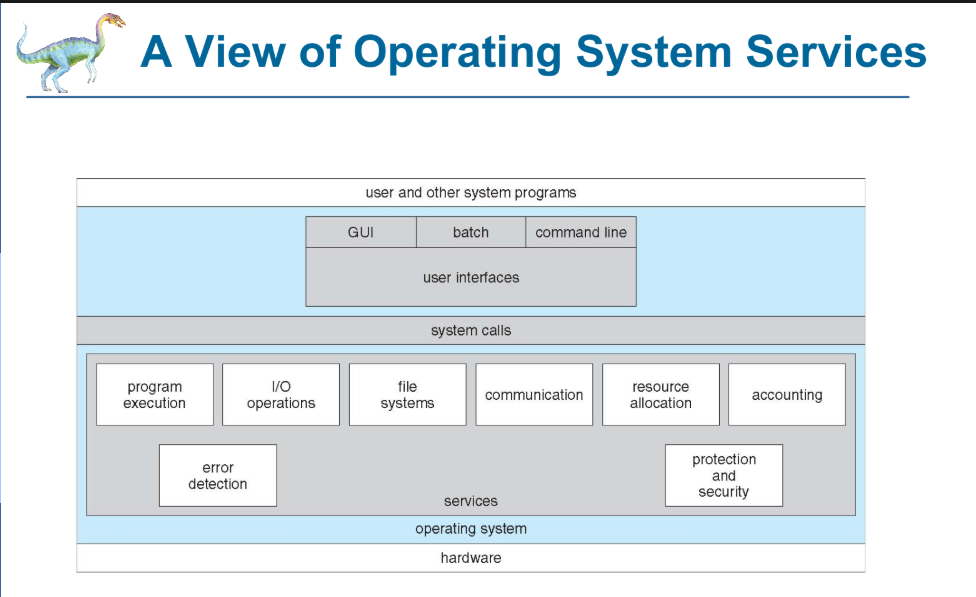
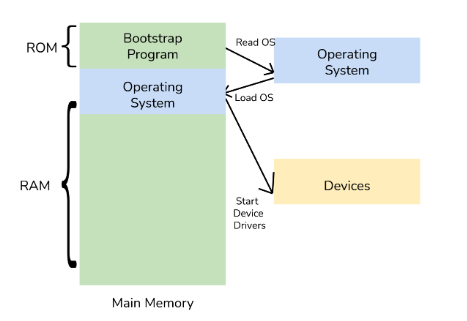
****

**What is Bootstrap in OS?**

The **bootstrap process** is the sequence of steps that a computer follows to load the operating system into memory and prepare it for execution. It ensures that the hardware is initialized, the OS is loaded, and the system is ready for user interaction.



**Interrupts:**An **interrupt** is a signal sent to the CPU to stop its current task and handle something important. It allows the OS to respond quickly to user inputs, hardware events, or errors.

**Types of Interrupts**

**1. Hardware Interrupts**

Generated by external devices (input/output devices, timers, etc.) to get the CPU’s attention. These interrupts occur asynchronously (not controlled by the program).

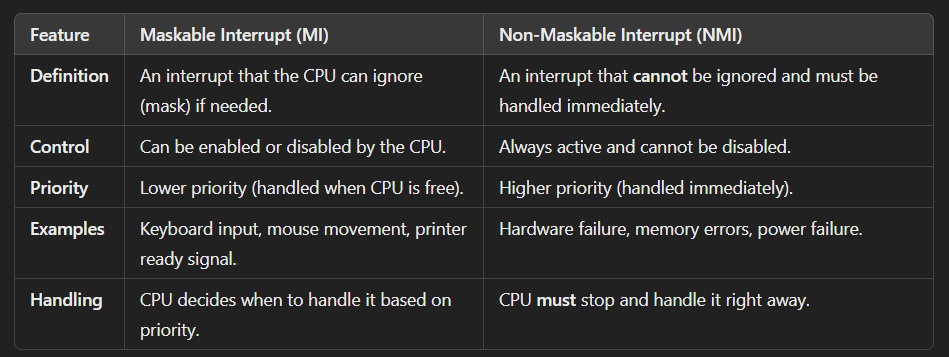
**Examples**:

* Pressing a keyboard key → OS registers the keypress.
* Moving a mouse → OS updates the cursor position.
* Printer is ready → OS sends the next page for printing.

**Subtypes of Hardware Interrupts:**

* Maskable Interrupts (MIs) → Can be ignored or delayed by the CPU.
* Non-Maskable Interrupts (NMIs) → Must be handled immediately (e.g., hardware failure).

**Maskable vs. Non-Maskable Interrupts**

****

**Edge-Triggered vs. Level-Triggered Interrupts**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**2. Software Interrupts**

Triggered by software programs to request CPU services. These interrupts occur synchronously (controlled by the program).

**Examples**:

* System Calls → A program requests OS functions (e.g., opening a file, allocating memory).
* Divide by Zero Error → The OS stops the program to prevent a crash.
* Invalid Memory Access → The OS generates an exception and handles it.

**Subtypes of Software Interrupts:**

* System Calls → Requests for OS services (e.g., reading/writing files).
* Exception Interrupts → Errors like division by zero or memory violations.

**3. Timer Interrupts**

Generated by the OS timer to switch between tasks (important for multitasking).

**Examples**:

* Task Scheduling → Switching between different running programs.
* Preventing CPU Overload → Ensuring no single process takes up all resources.

**4. Input/Output (I/O) Interrupts**

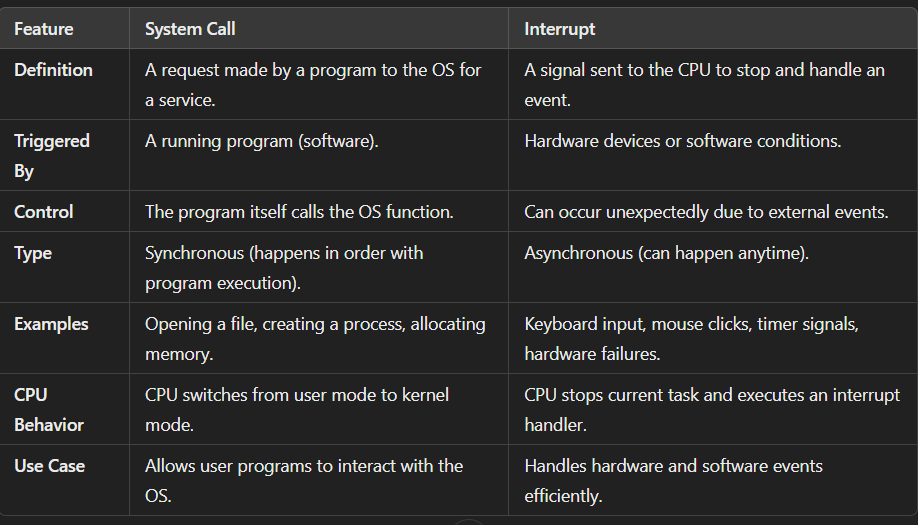
Generated by input/output devices when they need the CPU’s attention. Helps the CPU manage devices efficiently without constantly checking them.

Examples:

* Keyboard Interrupt → When a key is pressed, the OS registers the input.
* Mouse Interrupt → The OS processes mouse movements and clicks.
* Disk Read/Write Interrupt → The OS processes data from a hard drive or SSD.
* Printer Interrupt → The OS sends the next page when the printer is ready.

Why I/O Interrupts Are Important?

* The CPU doesn’t have to keep checking if an I/O device is ready.
* The device notifies the CPU only when needed, improving efficiency.

**Difference Between System Calls and Interrupts**

**Storage Hierarchy**

Storage systems are organized from fastest to slowest: **registers**, **cache**, **main memory**, **electronic disk**, **magnetic disk**, **optical disk**, and **magnetic tape**.

**Registers** and **cache** are volatile, while **magnetic disks** and **tapes** are non-volatile.

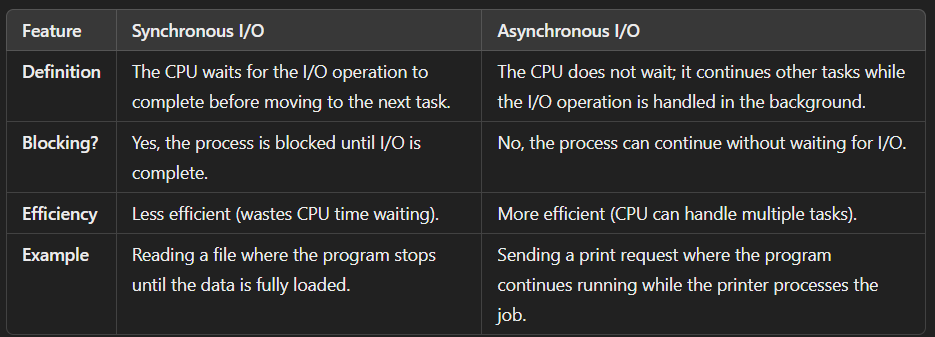
**Hardware Protection**

* **Dual-mode operation** ensures that user programs cannot interfere with the OS or other programs.
* The CPU operates in two modes: **user mode** (for user programs) and **monitor mode** (for the OS).
* **I/O protection** prevents user programs from directly accessing hardware.

A diagram of a programmable interrupt control

AI-generated content may be incorrect.

**Synchronous vs. Asynchronous I/O**



**Direct Memory Access (DMA)**

* **Key Points**:
  + **DMA** allows devices to transfer data directly to/from memory without CPU involvement.
  + This reduces CPU overhead and improves performance.
  + The DMA controller manages the transfer and interrupts the CPU when done.
* **Explanation**:
  + DMA is used for high-speed data transfers, like reading from a hard disk. It frees up the CPU to perform other tasks.

A diagram of a storage device hierarchy

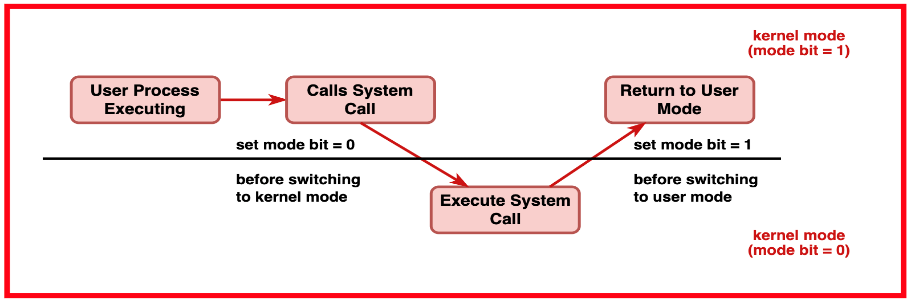
AI-generated content may be incorrect.

**Cache Coherency and Consistency**

* **Key Points**:
  + In multiprocessor systems, **cache coherency** ensures that all caches have the same data.
  + **Cache consistency** ensures that updates to data in one cache are reflected in all caches.
* **Explanation**:
  + Cache coherency and consistency are critical in systems where multiple processors access the same data. They prevent conflicts and ensure data integrity.

**Network Structure**

* **Key Points**:
  + **Local Area Networks (LANs)** connect devices in a small area (e.g., an office).
  + **Wide Area Networks (WANs)** connect devices over large distances (e.g., the internet).
  + **Routers** and **modems** are used to connect networks and devices.
* **Explanation**:
  + Networks allow computers to communicate and share resources. LANs are fast and local, while WANs are slower but cover larger areas.

**Past Paper Question:**

**What is Dual Mode in CPU?**

**Dual mode** in a CPU refers to the two different execution modes that ensure **system security and stability** by separating user processes from critical system operations.

**The Two Modes:**

1. **User Mode (Mode Bit = 1)**
   * Used for running user applications (e.g., browsers, text editors).
   * Restricted access to hardware and system resources.
   * If a program tries to perform a restricted operation (e.g., directly accessing hardware), it results in a **trap (exception)**.
2. **Kernel Mode (Mode Bit = 0)**
   * Used by the **operating system** (OS) to execute privileged instructions.
   * Full access to hardware (CPU, memory, I/O devices).
   * Only system-level processes (like device drivers, memory management) can run here.

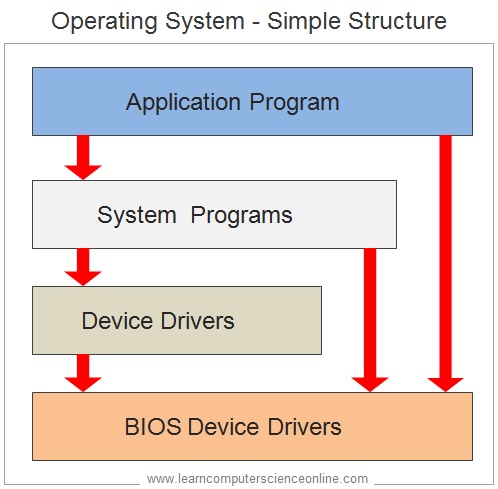
**Mode Switching:**

* When a user program requests an OS service (e.g., file access), it **triggers a system call**, switching the CPU from **User Mode → Kernel Mode**.
* After the OS completes the task, the CPU **switches back to User Mode**.

**What Hardware Support is Needed for Dual Mode?**

To implement **Dual Mode**, the CPU requires specific hardware features:

1. **Mode Bit (1-bit register)**
   * A special flag (0 for Kernel Mode, 1 for User Mode) to differentiate user and system operations.
2. **Privileged Instructions**
   * Certain instructions (e.g., direct memory access, I/O control) can only execute in **Kernel Mode**.
3. **Timer Interrupt**
   * Ensures the OS can regain control if a user process runs indefinitely.
   * The OS sets a timer, and when it expires, an **interrupt** forces a switch to Kernel Mode.
4. **Memory Protection**
   * The CPU enforces **memory access restrictions** to prevent user processes from modifying system memory.

**Operating System Structures**

**1️ Simple OS Structure – MS-DOS**

MS-DOS follows a simple and minimalistic structure where all OS functions are combined into a single program. It does not have clear separations between different components like file management, memory management, and process scheduling. The OS provides direct access to hardware, allowing user programs to interact with system resources without strict protection mechanisms.

* **Advantages:**
  + Fast and lightweight due to minimal layers.
  + Direct access to system hardware improves efficiency.
* **Disadvantages:**
  + No security mechanisms to prevent direct access to system resources.
  + Difficult to expand and maintain as there is no modular separation.
* **Example:** MS-DOS (Microsoft Disk Operating System), used in early personal computers.

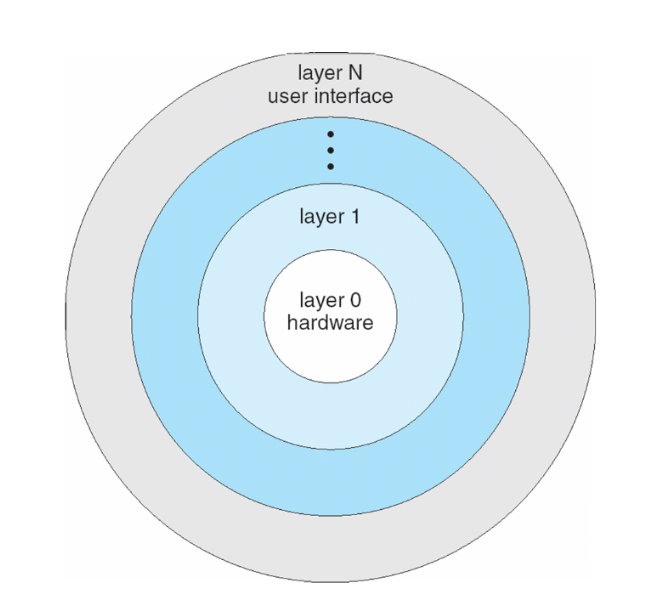
**2️ More Complex OS Structure – UNIX**

UNIX is more structured and modular compared to MS-DOS. It follows a layered approach, where different system functions such as process management, file system handling, and device management are divided into separate components. UNIX supports multitasking and multi-user environments, making it more versatile and secure.

* **Advantages:**
  + Better organization of OS functionalities compared to MS-DOS.
  + Enhanced security and stability with separate user and kernel spaces.
  + Supports multitasking and multiple users efficiently.
* **Disadvantages:**
  + More complex to develop and maintain than simpler OS structures.
  + Can require higher system resources compared to lightweight operating systems.
* **Example:** UNIX-based operating systems, including Linux and macOS.

**3️ Layered OS Structure – An Abstraction**

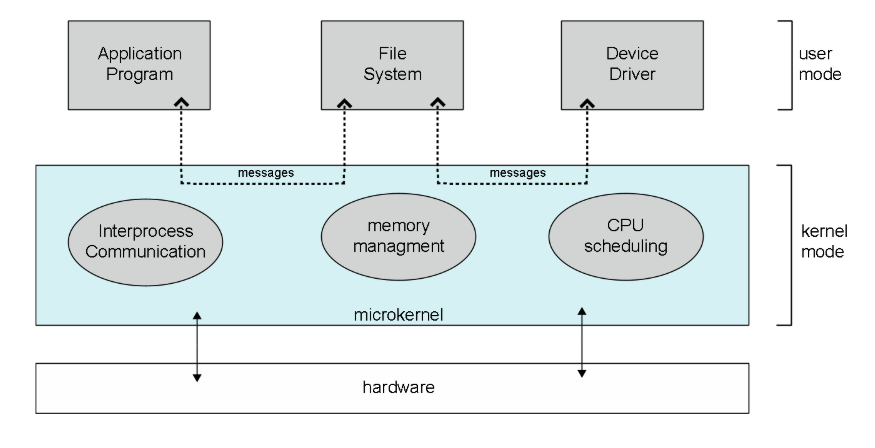
A layered operating system is divided into different levels, each handling specific tasks. Each layer interacts only with the layer directly above and below it. The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

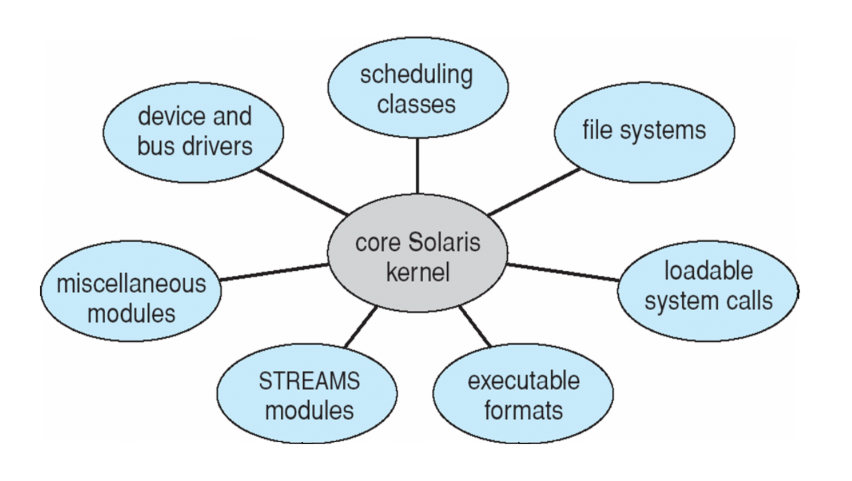
**Advantages:**

* + Clear separation of responsibilities enhances system reliability.
  + Debugging and system modifications are easier due to modularity.
  + Improved security since each layer has restricted access to certain system resources.
* **Disadvantages:**
  + Slower performance due to increased overhead from layer-to-layer communication.
  + Can be difficult to design an efficient layered structure that balances performance and security.
* **Example:** THE Operating System, one of the earliest OS designs based on the layered approach.

**4️ Microkernel OS Structure – Mach**

A microkernel OS keeps only essential functions, such as inter-process communication and process scheduling, inside the kernel. Other OS services, including device drivers and file management, run in user mode as separate processes. The goal is to enhance security and system stability.

* **Advantages:**
  + Improved reliability since most services run in user mode, reducing system crashes.
  + Easier to extend and update as services are modular and independent.
  + Greater security as faults in user-level services do not crash the kernel.
* **Disadvantages:**
  + Performance overhead due to message passing between user-mode services and the microkernel.
  + Increased complexity in implementing communication between system components.
* **Example:** Mach microkernel, which influenced modern systems like macOS and QNX.



**5️ Modular OS**

In a modular OS, the core functionality is kept in the kernel, while additional features are implemented as loadable modules. These modules can be dynamically loaded or removed as needed, allowing for greater flexibility and efficiency.

* **Advantages:**
  + Highly flexible, as modules can be added or removed without rebooting the OS.
  + More efficient than microkernel architecture, as modules interact with the kernel directly.
  + Easier to maintain and update specific OS components.
* **Disadvantages:**
  + Managing dynamic dependencies between modules can be complex.
  + Some security risks exist if unauthorized or unstable modules are loaded.
* **Example:** The Linux kernel, which uses Loadable Kernel Modules (LKM) to add features dynamically.

**6 Hybrid Operating System**

A **hybrid operating system** combines features of multiple OS architectures, such as monolithic, layered, microkernel, and modular approaches. The goal is to take advantage of the strengths of different designs while minimizing their weaknesses.

**Characteristics of Hybrid OS**

1. **Combines Monolithic and Microkernel Designs** – Critical functions remain in the kernel (for performance), while less critical services run in user space (for security).
2. **Supports Multiple OS Environments** – Can run different OS types (e.g., Windows can support UNIX-based subsystems).
3. **Uses Loadable Modules** – Allows the system to load and unload features dynamically.
4. **Enhanced Performance and Stability** – Reduces system crashes by isolating essential kernel functions from user processes.

**System Boot Process Explained**

When a computer is powered on, it does not immediately start running the operating system. Instead, it follows a step-by-step process to load and initialize the OS. This process is called **system boot**.

**1️ Execution Starts at a Fixed Memory Location**

* When the power is turned on, the **CPU** starts execution from a **predefined memory location** in the system firmware (ROM or EEPROM).
* This memory location contains **bootstrap code** that initializes the hardware and starts the boot process.

**2️ Firmware and ROM Hold Initial Boot Code**

* The **firmware** (BIOS or UEFI) is stored in **ROM (Read-Only Memory)** or **EEPROM (Electrically Erasable Programmable ROM)**.
* The firmware performs basic hardware checks and prepares the system for the next steps.

**3️ Loading the Operating System**

The system needs to **locate and load the OS kernel** into memory before it can start running. This process involves:

1. **A small program called the Bootstrap Loader**
   * Stored in ROM or EEPROM, it is responsible for finding the OS kernel.
2. **Loading the OS Kernel**
   * The bootstrap loader loads the kernel into RAM (main memory) and executes it.

**4️ Two-Step Boot Process (For Some Systems)**

* In some systems, the boot process is done in two steps:
  1. **Boot Block Execution**: A small boot block stored at a fixed disk location is loaded by the ROM code.
  2. **Bootstrap Loader Execution**: The boot block loads a more advanced bootstrap loader from disk, which then loads the OS.

**5️ GRUB Boot Loader**

* **GRUB (Grand Unified Bootloader)** is a common bootloader used in Linux and other OSs.
* It allows users to **select from multiple operating systems**, different kernel versions, or custom boot options before loading the OS.

**6️ Kernel Execution and System Startup**

* Once the **kernel** is loaded into RAM, it starts initializing system processes, drivers, and hardware components.
* After this, the **operating system is fully running**, and the user can interact with it.