3. Computer Architectures

Fundamentals of Operating Systems

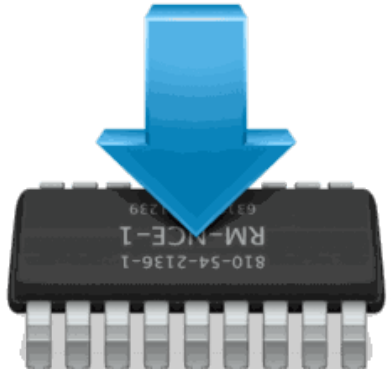
Summary

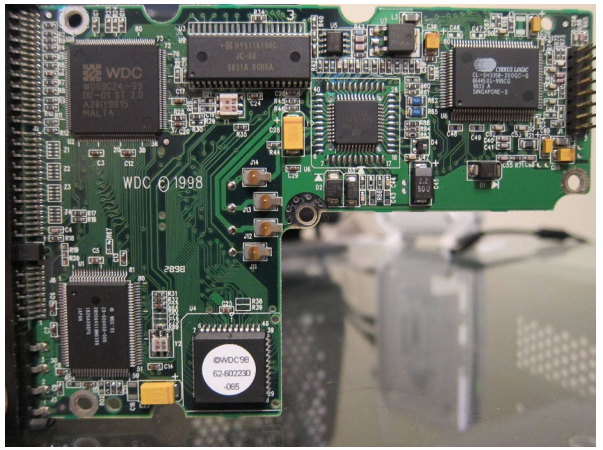
- Firmware
- Non-volatile Memory
- Motherboard
- Basic Input/Output System (BIOS)
- Unified Extensible Firmware Interface (UEFI)
- Drivers
- Operating System
- Operating System Structure
- Operating System Kernel

Firmware/1

Firmware is a type of software that is **embedded** within an **electronic component**. In this context, electronic component refers to any **intelligent integrated circuit** present in various devices, including **processors**, **graphics cards**, **sound cards**, **network interface cards**, and **peripheral devices** like printers and monitors.

The **primary function** of firmware is to **initialize the component** and **facilitate its interaction** with other components within the device. Essentially, firmware provides the component with a **set of communication interfaces** and **protocols**, allowing it to use a common language (encompassing syntax and semantics) to communicate effectively with other components.





ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Firmware/2

The term **firmware** is a combination of '**firm**' and '**ware**' indicating that firmware is **not easily modified** by the user. It serves as a **bridge** between hardware and software, enabling the **hardware components** to **communicate** and **function** according to **the software instructions**.

The **BIOS (Basic Input/Output System)** is a well-known example of firmware in computers. It resides on the **motherboard** and is crucial for managing **the initial startup process** of the computer, including **hardware initialization** and **booting the operating system**.

Inside every device, whether it is a desktop PC, a laptop, a car's onboard computer, or an airplane's internal computer, there are **similar fundamental components,** such as **motherboards**. These components may **differ in size** and **technical specifications**, but they perform **analogous functions**.

Firmware is typically stored in **non-volatile memory** within the electronic component or device, meaning the memory retains its contents even when the power is turned off.

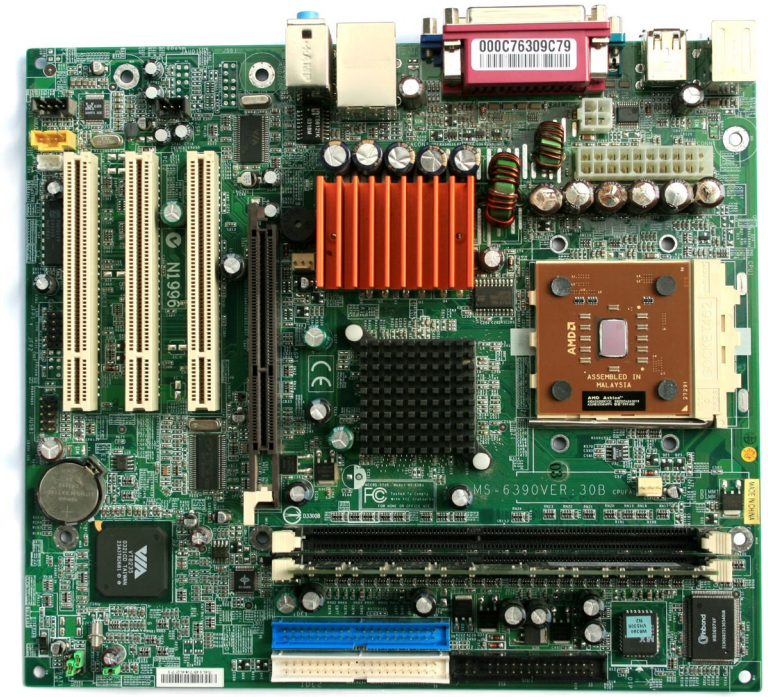
Non-volatile Memory

Common types of non-volatile memory used for storing firmware include:

- **ROM (Read-Only Memory):** Traditionally used for firmware storage, but once written, it cannot be modified.
- **EPROM (Erasable Programmable Read-Only Memory):** Can be erased and reprogrammed using UV light, but this process is not as convenient as modern alternatives.
- **EEPROM (Electrically Erasable Programmable Read-Only Memory):** Can be erased and reprogrammed using electrical charge, making it more flexible than EPROM.
- **Flash Memory:** A type of EEPROM that can be erased and rewritten in blocks, commonly used in modern devices due to its flexibility and higher storage capacity.

These storage types are **embedded** on the device, ensuring that the firmware is **readily accessible** to **initialize** and **control** the device components when powered on.

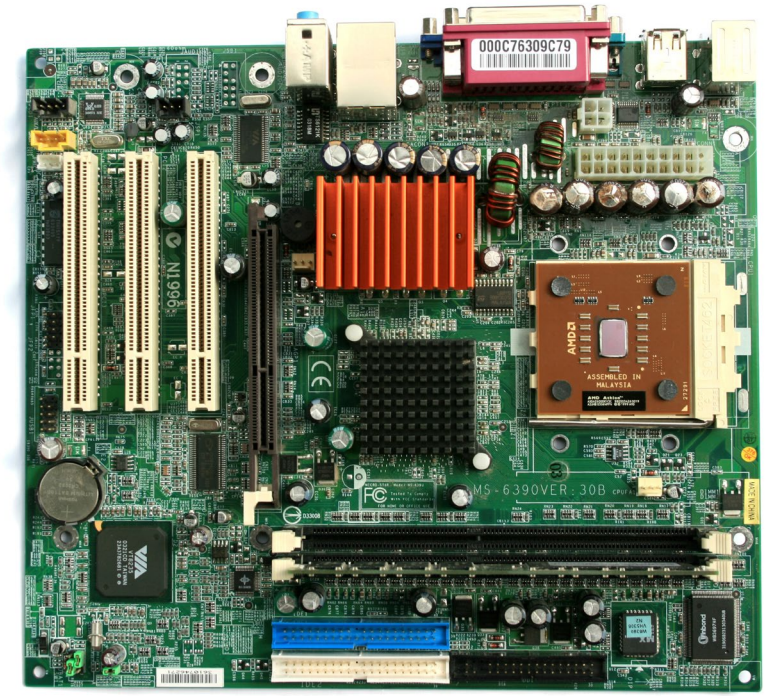
Motherboard/1



A **motherboard** is the main **printed circuit board** (PCB) in a computer or other electronic device that holds and allows communication between many of the crucial electronic components.

- **Central Hub:** It acts as the central hub that connects all the parts of the computer together, allowing them to communicate and work together.
- **CPU Socket:** The motherboard contains a socket that holds the central processing unit (CPU). This is where the CPU is installed.
- **Memory Slots:** It has slots for memory (RAM) modules, allowing the computer to temporarily store and quickly access data.
- **BIOS/UEFI:** The Basic Input/Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) firmware is stored on the motherboard, providing the necessary instructions for the initial boot-up process and hardware initialization.

Motherboard/2



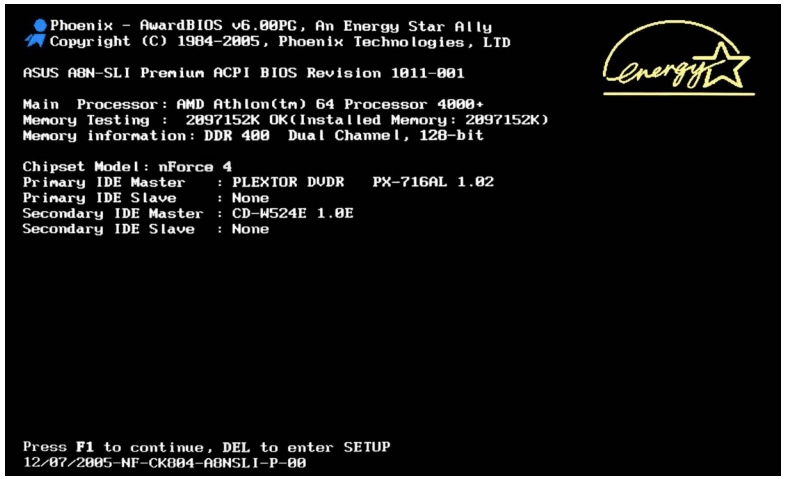
- **Expansion Slots:** These slots allow additional cards to be added, such as graphics cards, sound cards, network cards, and other peripherals.
- **Storage Connectors:** The motherboard has connectors for storage devices, like hard drives and SSDs.
- **Power Connectors:** It includes connectors to supply power to the motherboard and its components from the power supply unit (PSU).
- **Peripheral Connectors:** Ports and connectors for peripherals such as USB devices, audio jacks, network cables, and display connections (HDMI, DisplayPort).

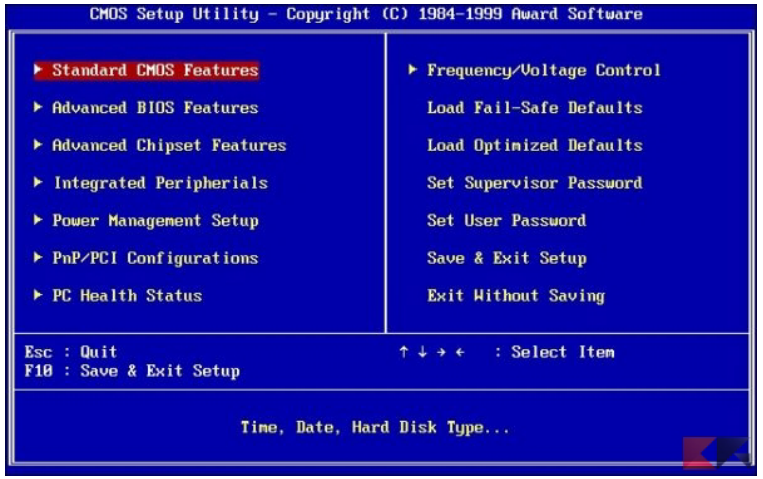
In summary, the motherboard is the **backbone** of a computer, integrating all the critical components and ensuring they can communicate and function together effectively.

Basic Input/Output System (BIOS)

The **Basic Input/Output System (BIOS)** is a set of **software routines** that provides the **foundational software structure** allowing the **Operating System** to **interface** with the device's **hardware**.

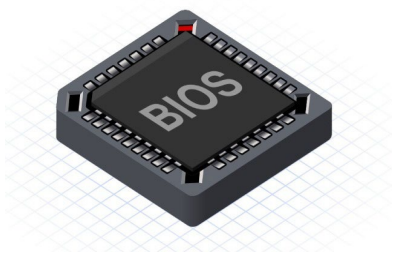
The **BIOS** is stored in a **non-volatile memory** chip, meaning it retains data even without power. Nowadays, the **BIOS** is typically stored on **FLASH** or **EEPROM** memory soldered directly onto the **motherboard**, which can be rewritten using an appropriate upgrade procedure.





BIOS – BOOTSTRAP/1

When a device is started, **before** the Operating System loads, the **BIOS** performs several **key tasks**:



- **Manage Hardware:** It runs various **test routines** called **POST (Power On Self Test)** to verify the proper functioning of different components, including the motherboard, processor, memory, and others.
- **Configure Hardware Settings:** If no serious errors are detected during the POST phase, the BIOS configures the hardware settings, preparing the system for the operating system to take over.

- **Display Startup Screen:** It presents a video screen that allows the user to configure and manage some basic BIOS settings.
- **Initialize Boot Routine:** The BIOS starts the **boot routine** to load the **Operating System**. The boot process is incremental, loading the essential parts of the Operating System from the boot sector to bring the system to a stable operating state.

BIOS – BOOTSTRAP/2

- **Locating MBR:** After executing the basic BIOS routines, the BIOS's first instruction is to locate the **'Starting Point'** of the hard disk (or other mass storage media) where the Operating System is stored. At this location, the **Master Boot Record (MBR)** can be found.

- **Control to Operating System:** The BIOS reads the content of the MBR and transfers control to the Operating System (from this point, the device management is fully handled by the Operating System).

The **MBR** contains information about the **location** of the **BOOT SECTOR** on the hard disk (or other mass storage media hosting the Operating System).

The **BOOT SECTOR** is the **entry point** for starting the **Operating System**.

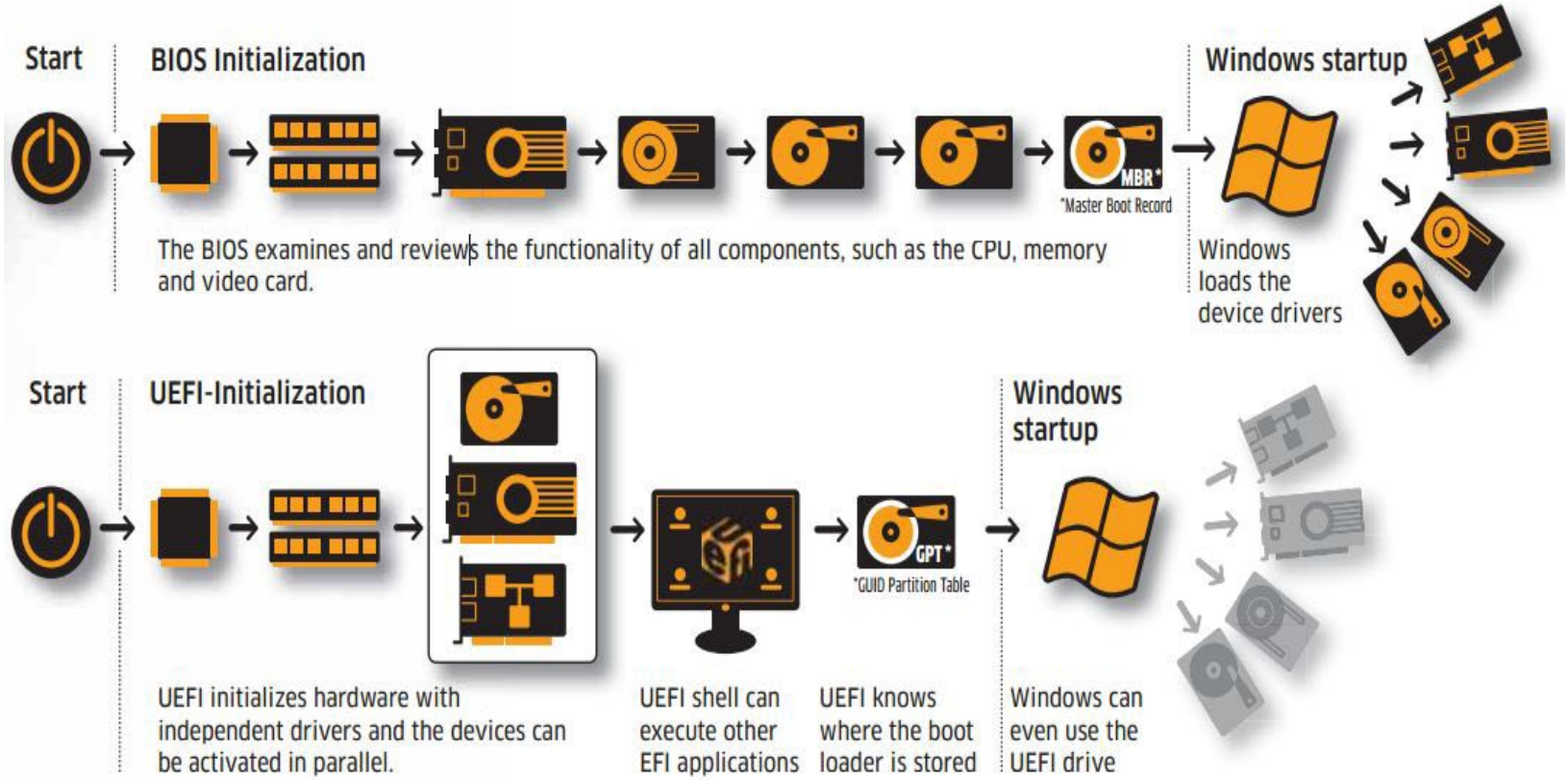
Unified Extensible Firmware Interface (UEFI)/1

The **Unified Extensible Firmware Interface (UEFI)** is a modern **firmware** interface designed to **initialize hardware** components and **load the operating system** when a device starts up.

Similar boost process BUT with advantages over BIOS:

- **Graphical User Interface (GUI):** UEFI can offer a more user-friendly graphical interface with mouse support, making it easier to navigate and configure settings.
- **Boot Manager:** A built-in program within UEFI that determines which operating system or boot loader to execute. This helps streamline the boot process and offers more flexibility compared to BIOS.
- **Driver Support:** UEFI can load drivers directly, enabling better hardware compatibility and performance during the boot process.
- **Enhanced Security:** UEFI includes features like Secure Boot, which helps protect the system by ensuring that only trusted software is loaded during the boot process.
- **Extensibility and Updates:** UEFI is modular and can be updated or extended with new functionalities, providing better support for new technologies and improved system stability over time.

Unified Extensible Firmware Interface (UEFI)/2



Drivers

Drivers are software programs that allow the **operating system** and **other software** to communicate with **hardware** devices.

Drivers act as **intermediaries**, translating high-level commands from the OS into low-level commands that the hardware can understand. They **manage the interaction** between the **operating system** and the **hardware**.

Characteristics:

- Installed on the operating system.
- Facilitates communication between OS and hardware.
- Can be updated or replaced more frequently.

Firmware vs Drivers

Key Differences:

- **Location:**

Firmware: Embedded in the hardware.
Drivers: Installed on the operating system.

- **Function:**

Firmware: Manages and controls hardware at a fundamental level.
Drivers: Translates OS commands into actions performed by the hardware.

- **Update Frequency:**

Firmware: Updated less frequently, often through specialized update processes.
Drivers: Can be updated regularly to improve performance or add new features.

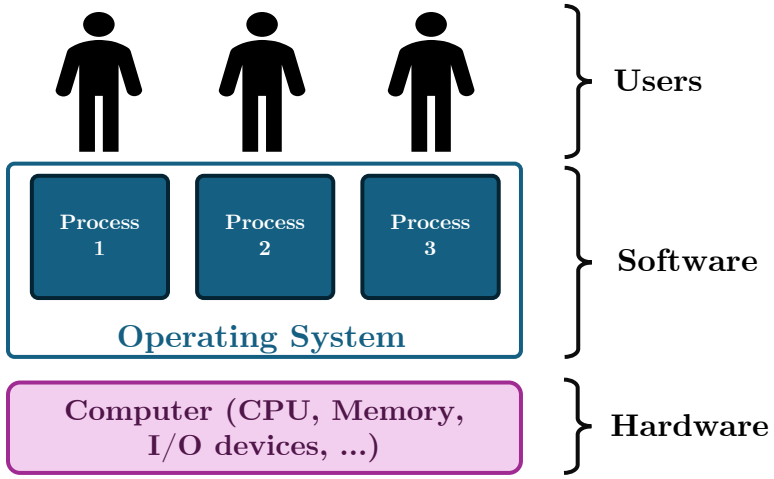
Operating System/1

The **Operating System** is the **fundamental software** responsible for **managing** the entire **device's operations**, including **running applications**, **programs**, and **services**, as well as **interacting** with the user.

Positioned between the **hardware** layer (including firmware) and the **application** layer (applications, programs, services), the Operating System's key role is to abstract the computer's complexities for the application layer.

Specifically, the Operating System handles four **primary tasks**:

1. **Managing** hardware components
2. **Executing** applications, programs, and services
3. **Interfacing** with peripheral devices
4. **Facilitating** user interaction



Operating System/2

In general, the Operating System defines **two main aspects** of a device:

- **Operational Modality of the Device**

This modality determines the purpose and scope of the device, explaining why a specific device is used.

- **User Interaction**:

This modality defines how users can engage with and operate the device.

Examples of Operating Systems include:

- Microsoft Windows,
- GNU/Linux,
- macOS,
- Android,
- iOS

... and many others.

Classification Based on Processing Features

- **Batch Operating Systems**: These Operating Systems run programs in a non-interactive manner. Users load both programs and data, and results are provided only after processing is completed.
- **Interactive Time-Sharing Operating Systems**: These Operating Systems run multiple programs simultaneously, sharing hardware resources (such as CPU time and memory) among them. Users can interact with programs during execution, providing input and receiving output throughout the process.
- **Real-Time Operating Systems**: These Operating Systems ensure that programs execute in real-time (or within a fixed time interval) without delays, latencies, or performance degradation. They are typically used in industrial processes or applied robotics.
- **Embedded Operating Systems**: These Operating Systems are integrated within dedicated hardware, often found in specific devices like cars or airplanes.
- **Hypervisor Operating Systems**: These Operating Systems enable the subdivision of hardware resources into different virtual machines, allowing multiple OSs, programs, and operational modes to run concurrently.

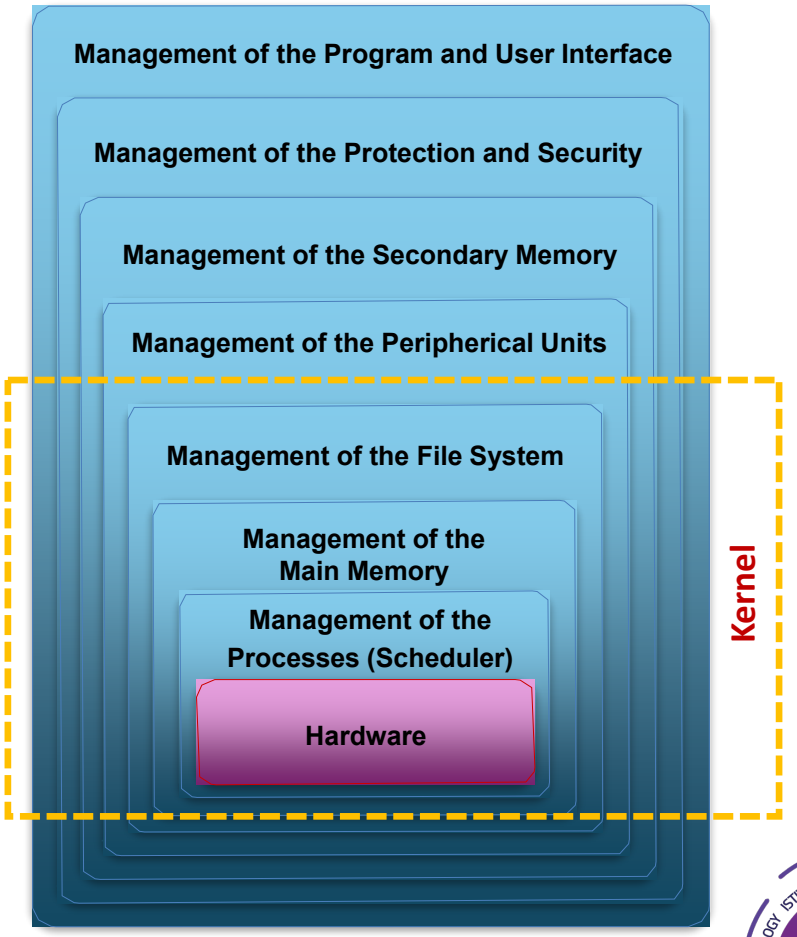
Classification Based on Functional Characteristics

- **Mono-Task Operating Systems:** These operating systems allow only one program (i.e., task) to be executed at a time. A new program can only start once the current one has finished.
- **Multi-Task Operating Systems:** These operating systems enable the parallel execution of multiple programs. Hardware resources (e.g., CPU time, memory) are shared among various programs, allowing them to run simultaneously.
- **Multi-Threading Operating Systems:** In these operating systems, each single program is divided into multiple sub-programs (i.e., threads). Each thread is executed on the CPU (or on a core), enhancing the device's performance.
- **Mono-User Operating Systems:** These operating systems are designed to interact with and execute programs for only one user at a time.
- **Multi-User Operating Systems:** These operating systems can interact with and execute programs for multiple users simultaneously.

Operating System Structure

An **Operating System** can be structured as follows:

1. **Management** of Programs and User Interface
2. **Management** of Protection and Security
3. **Management** of Secondary Memory
4. **Management** of Peripheral Units
5. **Management** of the File System
6. **Management** of Main Memory
7. **Management** of Processes (Scheduler)

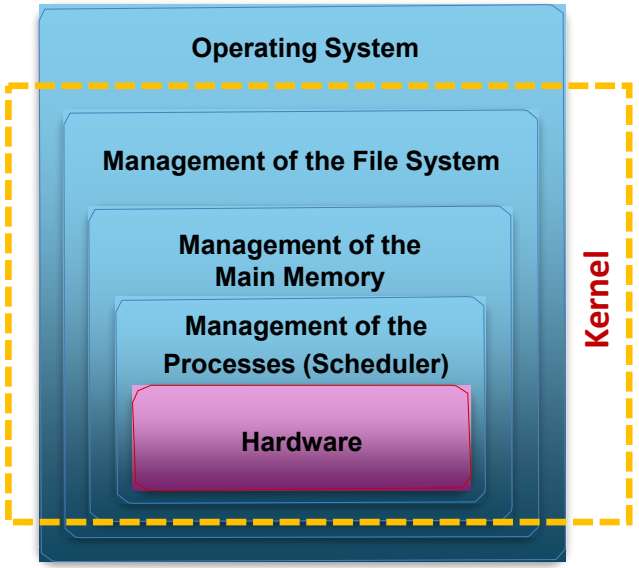


Operating System Kernel

The **kernel** is the **central component** of the **Operating System**. It serves as a **bridge** between **applications** and **physical hardware**, ensuring smooth and coordinated task execution.

Based on their **internal structure** and **kernel design**, Operating Systems can be categorized into three types:

- **Monolithic** kernel Operating System
- **Micro-kernel** Operating System
- **Hybrid** kernel Operating System



Monolithic Kernel Operating System

Monolithic kernel Operating System:

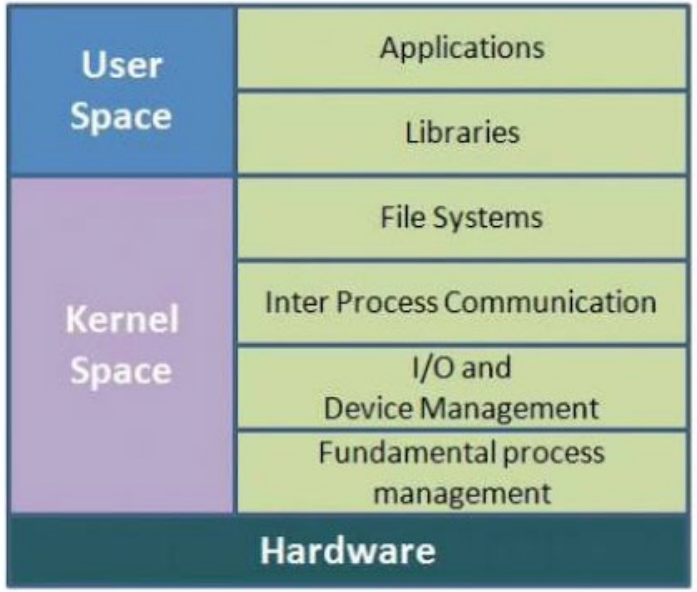
This system implement a **complete abstraction** of the computer within the **kernel**, which runs the operating system itself.

Monolithic kernel includes all the **core functionalities** required for the operating system to manage hardware and system processes within the **Kernel Space**. This includes device drivers, memory management, file system management, and system calls, all running in a single address space. **Applications** and **libraries** run in **User Space**.

By running all essential services in kernel space, monolithic kernels can achieve **high performance** due to minimal context switching between user mode and kernel mode. This **reduces overhead** and **improves efficiency**.

Since all components run in the same address space, a **failure** or **bug** in one part of the **kernel** can potentially **crash** the entire system.

Monolithic kernel is less modular compared to micro-kernel. **Adding new features** or **drivers** often requires **modifying** and **recompiling** the entire kernel, which can be cumbersome and limits flexibility.



Micro-kernel Operating System

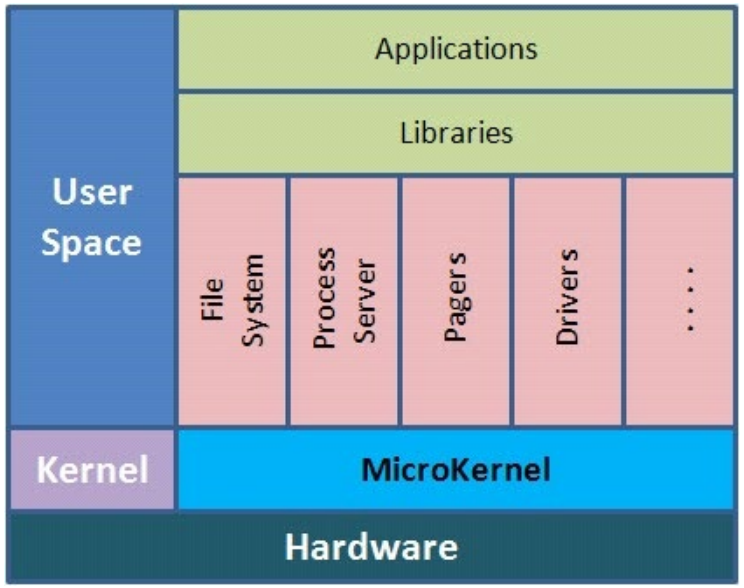
Micro-kernel Operating System:

This system implements only the **essential** Operating System **functions** in the **kernel**. Other functionalities are provided by separate system programs such as device drivers and servers.

Micro-kernel is designed to be **modular**. By keeping only **essential functions** like inter-process communication, basic scheduling, and minimal hardware abstraction in the **Kernel Space**, **other functionalities** are handled by **separate User-Space processes**. This design enhances **modularity** and allows for easier **updates** and **maintenance**.

By **isolating drivers** and **system services** from the kernel, the micro-kernel can **increase** system **stability** and **security**.

Failures or **bugs** in one service **do not** necessarily **crash** the entire system, as they would in a monolithic kernel.



Hybrid kernel Operating System

Hybrid kernel Operating System:

This system implements **various** Operating System **functionalities** within the **kernel** but can load **additional modules** at boot time to enhance and complete the kernel's features.

Hybrid kernel provides a **balance** between the monolithic and micro-kernel architectures. By including **essential functionalities** within the kernel and allowing **additional modules** to be **loaded** at **boot time**, they offer a flexible approach to system design.

This architecture aims to achieve the **performance** benefits of monolithic kernels while maintaining a degree of **modularity**.

Operating systems like modern versions of **Windows** and **macOS** use a **hybrid kernel** approach. These systems demonstrate the practical implementation and advantages of hybrid kernels in real-world scenarios.