# Operating System

Ms.Nirmala Shinde Baloorkar

Assistant Professor

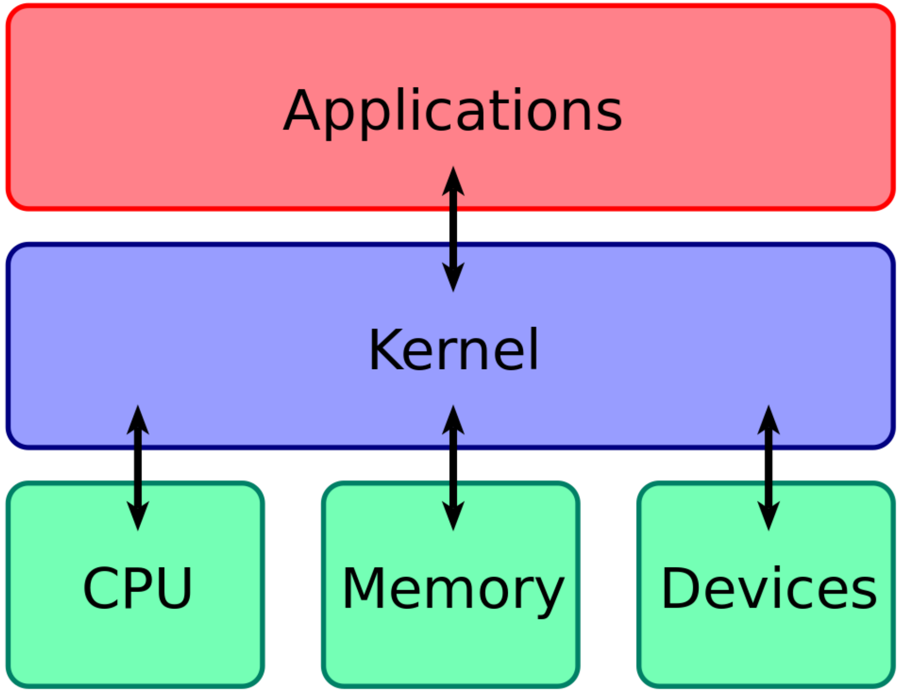
Department of Computer Engineering

# Outline

- System Calls
- System Boot
- Booting Process in Operating Systems

# Kernel

- What is kernel?
  - Kernel is the core component of the Operating System
  - Complete control of the hardware.



# Objectives of Kernel

- **Communication**
  - Establish communication between user-level applications and hardware
- **Process State**
  - Decide the state of incoming processes
- **Disk Management**
  - Control disk management
- **Memory Management**
  - Control memory management
- **Task Management**
  - Control task management

# User Mode Vs Kernel Mode

## User Mode

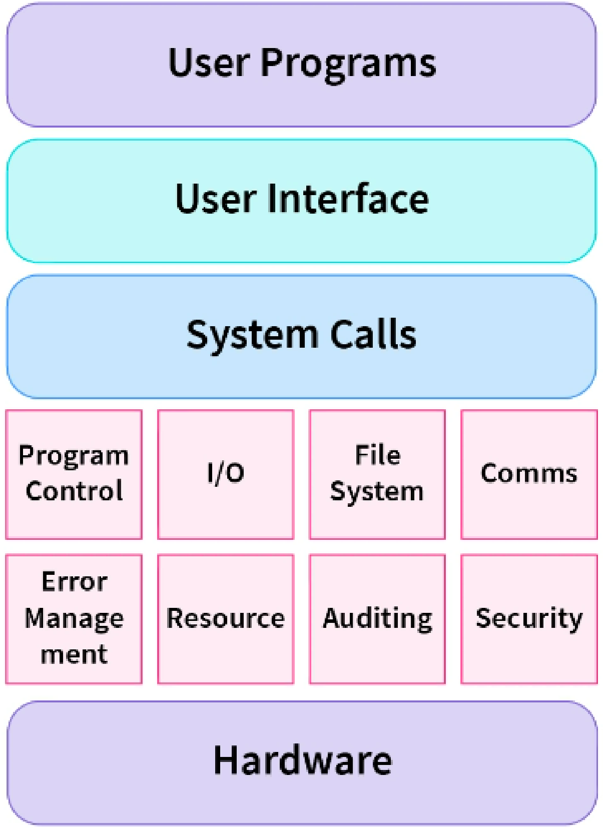
- **Restricted Access**:
  - In user mode, the CPU has limited access to system resources. This mode is designed to protect the system from accidental or malicious interference by user programs.
- **Running Applications**:
  - Most user applications, like web browsers and games, run in user mode. They can perform tasks like reading and writing etc.
- **System Calls**:
  - When a user application needs to perform a task that requires higher privileges, it makes a system call to request the operating system's services.

## Kernel Mode

- **Full Access**:
  - In kernel mode, the CPU has unrestricted access to all system resources, including hardware and memory. This mode is reserved for the operating system's core functions.
- **Operating System Tasks**:
  - The operating system's kernel runs in kernel mode. It manages system resources, handles hardware interactions, and enforces security policies.
- **Privileged Operations**:
  - Tasks like managing memory, scheduling processes, and handling interrupts are performed in kernel mode.

# System Calls

- System calls provide an interface to the services made available by an Operating System.



User Programs

User Interface

System Calls

Program Control

I/O

File System

Comms

Error Management

Resource

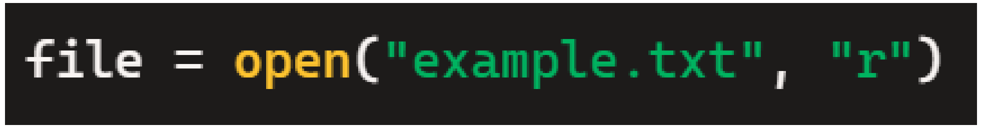
Auditing

Security

Hardware

# Working of System Call

- **User Process Executing**:
  - The user program calls the open function, specifying the file name and the mode.
- **Get System Call**:
  - The open function in the user program invokes the system call interface, which switches the CPU to kernel mode.
- **Execute System Call**:
  - The kernel processes the request, locates the file, and checks permissions. If successful, it returns a file object to the user program.
- **Return System Call**:
  - The CPU switches back to user mode, and the file object is used by the user program to read or write to the file.



```
file = open("example.txt", "r")
```

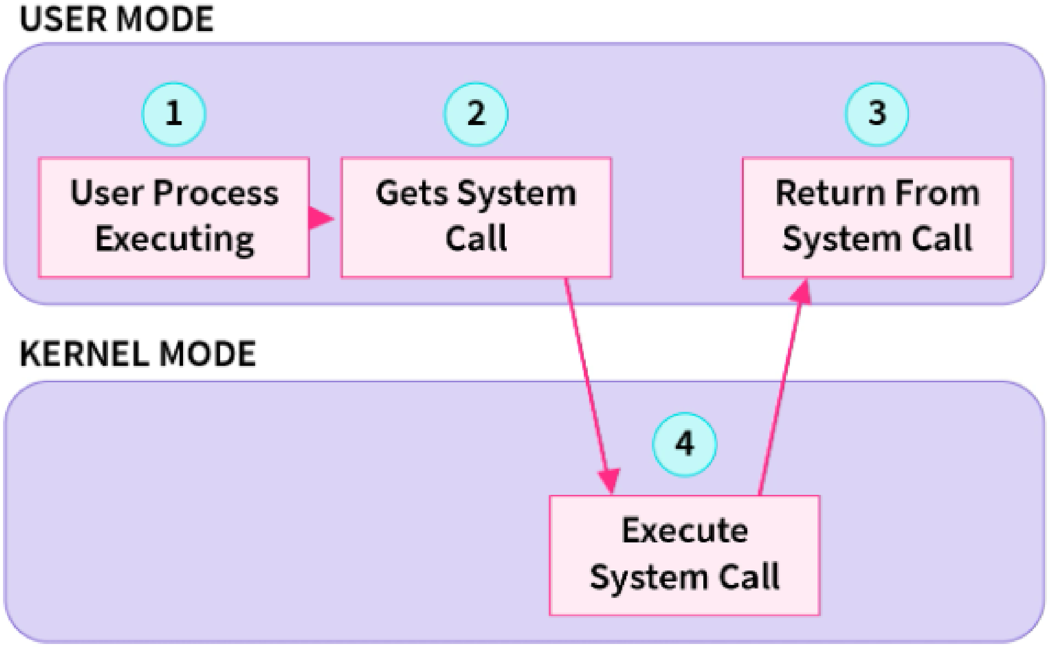
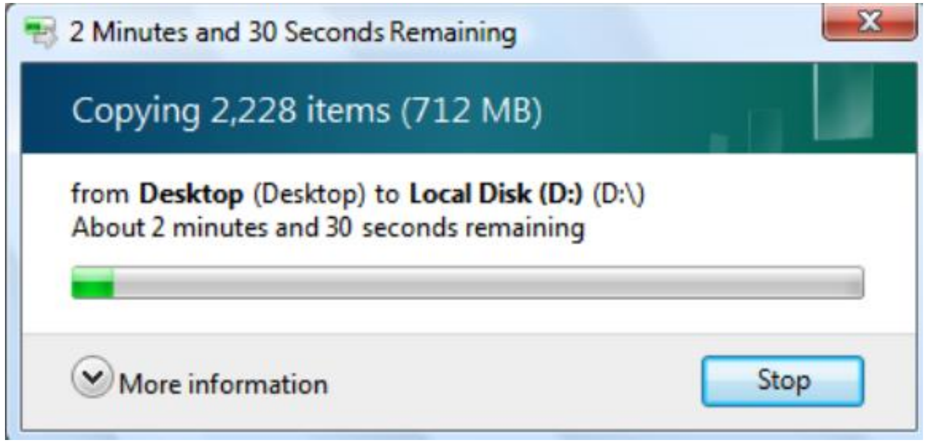


Image source: System Calls in OS (Operating System) - Scaler Topics

# Example of System Call

**Sequence of system calls to copy contents of one file to another**

1. Display dialog to choose source file
2. Display dialog to choose destination folder
3. Display progress dialog
4. Open source file for reading
5. Open destination file for creating and writing
6. Loop while more source bytes remaining:
    1. Read n bytes from source file
    2. Write n bytes to destination file
    3. Update progress dialog
7. Close source file
8. Close destination file
9. Close progress dialog
10. Terminate normally



2 Minutes and 30 Seconds Remaining

Copying 2,228 items (712 MB)

from **Desktop** (Desktop) to **Local Disk (D:)** (D:\)
About 2 minutes and 30 seconds remaining

# System Calls

- Typically written in a high-level language (C/C++)
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call
- Most Common APIs are:
  - Win32 API for windows
  - POSIX API for Unix
  - Java API for JVM

# Types of System Calls

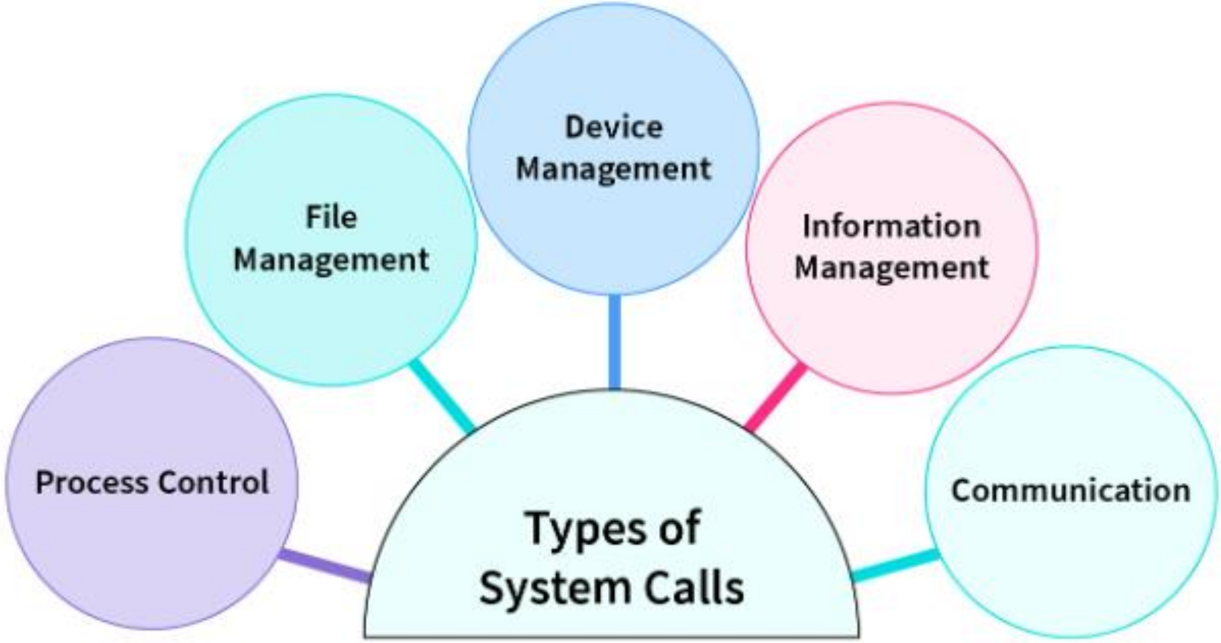


Image source: System Calls in OS (Operating System) - Scaler Topics

# Process System Calls

- **fork()**:
  - Creates a new process (child process) that runs concurrently with the parent process.
- **exec()**:
  - Loads a new program into the current process space, replacing the current process with the new program.
- **wait() / waitpid()**:
  - Suspends execution of the calling process until one of its child processes terminates. waitpid() allows more control over which child process to wait for.
- **exit()**:
  - Terminates the calling process and optionally returns an exit status to the operating system.
- **getpid()**:
  - Returns the process ID (PID) of the current process.
- **getppid()**:
  - Returns the PID of the parent process.
- **kill()**:
  - Terminates a specified process.

# File  System Calls

- open()
- close()
- read()
- write()
- lseek()
- mkdir()
- rmdir()
- unlink()
- rename()
- stat() / fstat()
- chmod()
- chown()
- Link()
- Unlink()
- Mount()
- Umount()

# Information Management System Calls

- **time()**
- **gettimeofday()**
- **clock_gettime()**
- **alarm()**
- **sleep()**
- **nanosleep()**
- **getitimer()**
- **setitimer()**
- **Uptime()**

# I/O management related system calls

- **open() / close()**:
  - Opens or closes a file or device.
- **read() / write()**:
  - Reads from or writes to a file descriptor.
- **ioctl()**:
  - Performs device-specific I/O operations.

# Communication System Calls

It facilitate communication between processes.

Communication facilitates data sharing, resource sharing, and cooperation among processes, allowing them to work together towards common goals.

- Pipe()
- Shmget()
- msgget()
- Send()
- Recv()

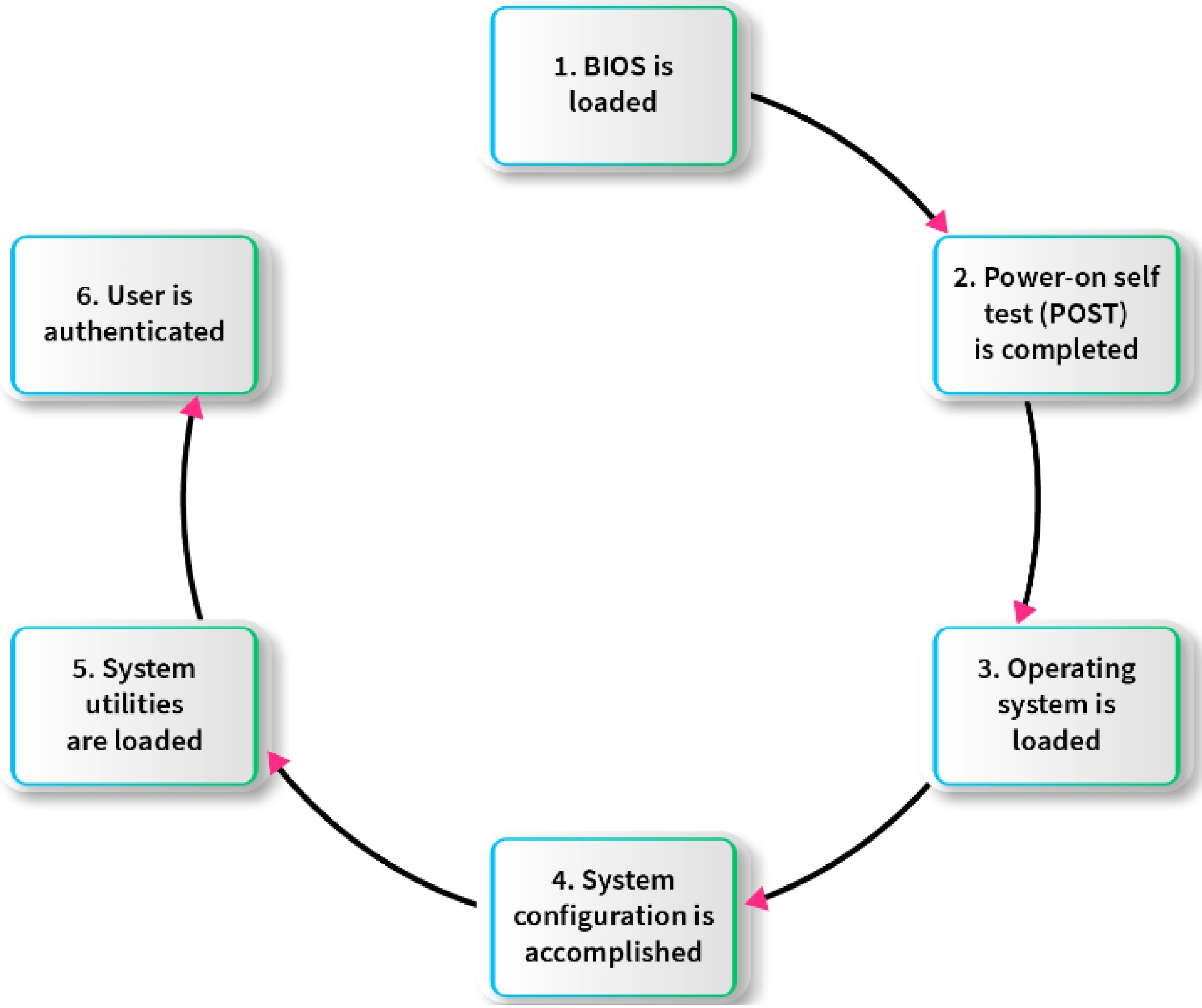
# What is Booting?

- Booting is the initialization process of a computer system, where the hardware is prepared, and the operating system is loaded to enable functionality.
- Why is it important?
  - Initializes hardware
  - Loads and starts the operating system
  - Verifies the system’s functionality
- Booting Process

# Types of Booting

- **Cold Booting (Hard Booting)**
  - Cold booting is the process of starting a computer from a completely powered-off state by pressing the power button.
  - Requires full initialization of hardware and software.
- **Warm Booting (Soft Booting)**
  - Warm booting is the process of restarting a computer without turning off the power, often done to apply new configurations or after installing new software or hardware. This process is also known as rebooting.
  - Achieved using options like "Restart" in the operating system.

# Booting Process



1. BIOS is loaded

2. Power-on self test (POST) is completed

3. Operating system is loaded

4. System configuration is accomplished

5. System utilities are loaded

6. User is authenticated

# Booting Process (continued)

**1. BIOS and Setup Program**:

- **ROM (Read-Only Memory)**: Permanent and unchanging memory that stores the BIOS.
- **BIOS (Basic Input/Output System)**: Part of the system software that includes instructions for input and output operations.
- **Load**: Transferring data from a storage device to memory. The ROM loads BIOS into the computer’s memory.
- **Setup Program**: A special program containing settings to control hardware, accessible only while the BIOS information is visible.

# Booting Process (continued)

**2. Power-On Self-Test (POST)**:

- **POST**: A series of tests conducted on the computer’s main memory, input/output devices, disk drives, and the hard disk.
- The BIOS conducts POST to check the input/output system for operability.
- If any problem occurs, the computer produces a beeping sound and an error message appears on the monitor.

# Booting Process (continued)

**3. The Operating System (OS) Loads**:

- The BIOS searches for the operating system.
- **CMOS (Complementary Metal-Oxide-Semiconductor)** settings determine where to look for the operating system.
- The operating system's kernel is loaded into the computer's memory.
- The operating system takes control of the computer and begins loading system configuration information.

# Booting Process (continued)

**4. System Configuration**:

- **Registry**: A database to store information about peripherals and software.
- **Peripheral**: A device connected to a computer.
- **Driver**: A utility program that makes peripheral devices function properly.
- The operating system’s registry configures the system.
- Drivers are loaded into memory during this step.

**5. System Utility Loads**:

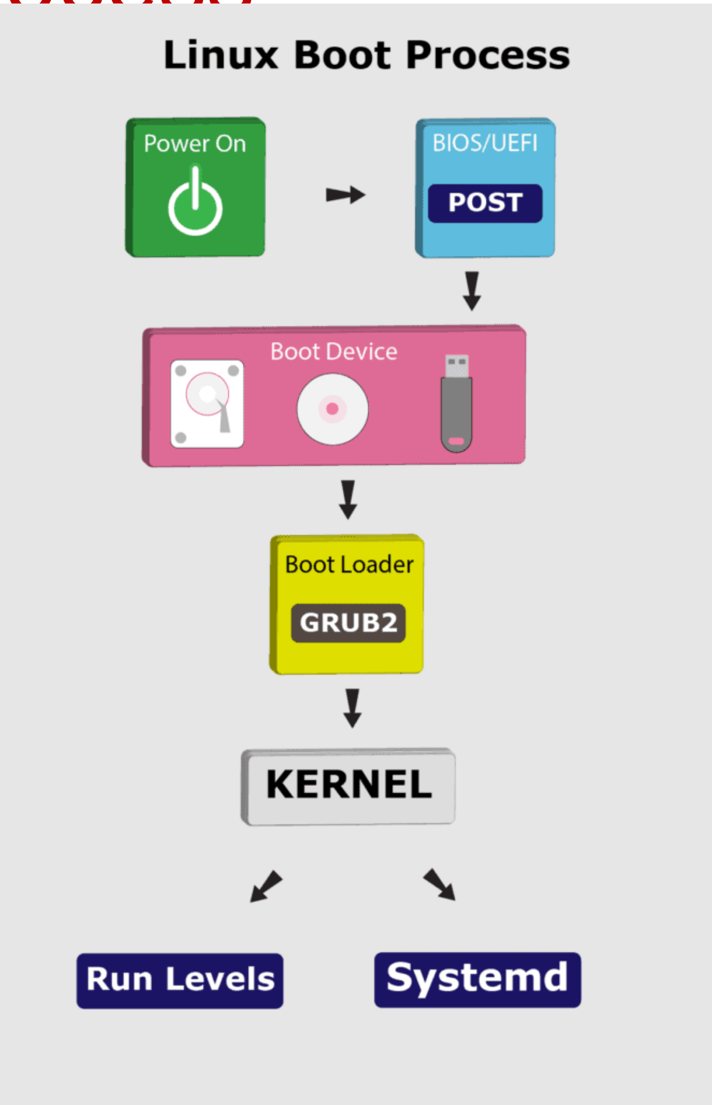
- System utilities are loaded into memory, including:
  - Volume control
  - Antivirus software
  - PC card unplugging utility

# Booting Process (continued)

**6. User Authentication**:

- Authentication or user login occurs, requiring:
    - Username
    - Password
- After this process, the user interface starts, enabling user interaction with the computer and its programs.

# Example – Linux Boot Process



https://www.baeldung.com/linux/boot-process

# Example – Boot Loader

- GRUB (also known as GRand Unified Bootloader)
- Bootloader and boot manager for **Linux** and other **Unix**-based OSes.
- GRUB starts after **BIOS** finishes the necessary hardware tests and loads it from the Master Boot Record (MBR).
- Once loaded, GRUB takes control of the system and loads the **Linux kernel**.
- **Example**: The GRUB configuration file is typically located at /boot/grub/grub.cfg.
- LINUX BOOT PROCESS

# Common Boot Issues

- Hardware failure (e.g., RAM, storage device).
- Corrupted bootloader or operating system files.
- Incorrect BIOS/UEFI settings.

# Question ?