Information Technology Institute

Introduction to Operating Systems

Prepared by:

Zeyad Ashraf

• Course Duration and Evaluation

- Duration: 18 hours
  - 6 Lectures
- Evaluation Criteria:
  - A comprehensive exam after finishing all the main conceptual courses .
  - Attendance during course .

Course Outlines

Introduction to Computer System and Operating System

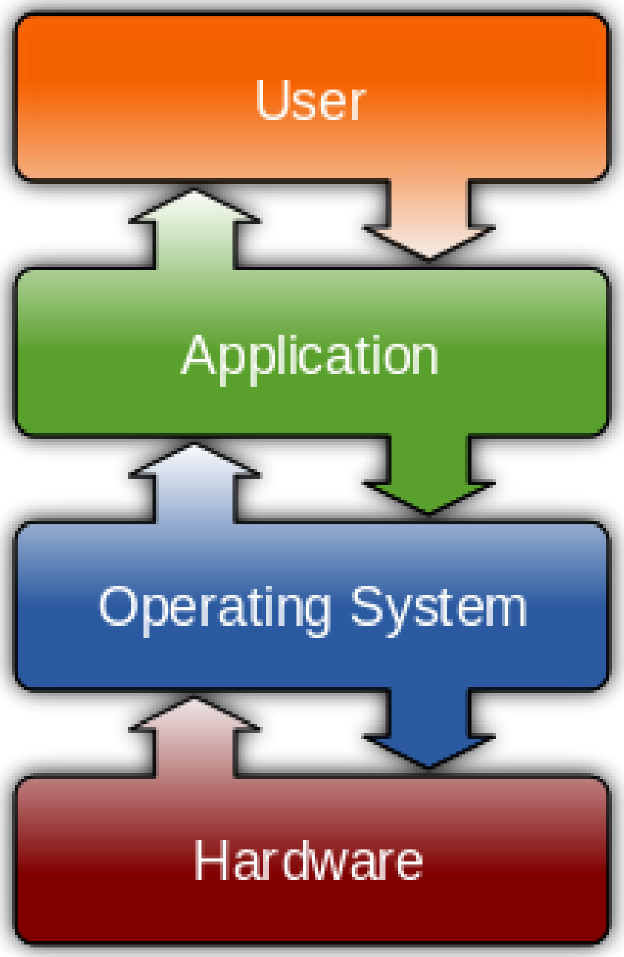
• Session Content

- Computer System Components
- Von Neumann Architecture
- Computer Operating System
- Why We Need an OS?
  - Complex and Multiprocessor Systems
  - Multiuser Systems
- Operating System Components
- Why Is It Important to Know About the OS?
- Responsibilities of an OS

Course Outlines

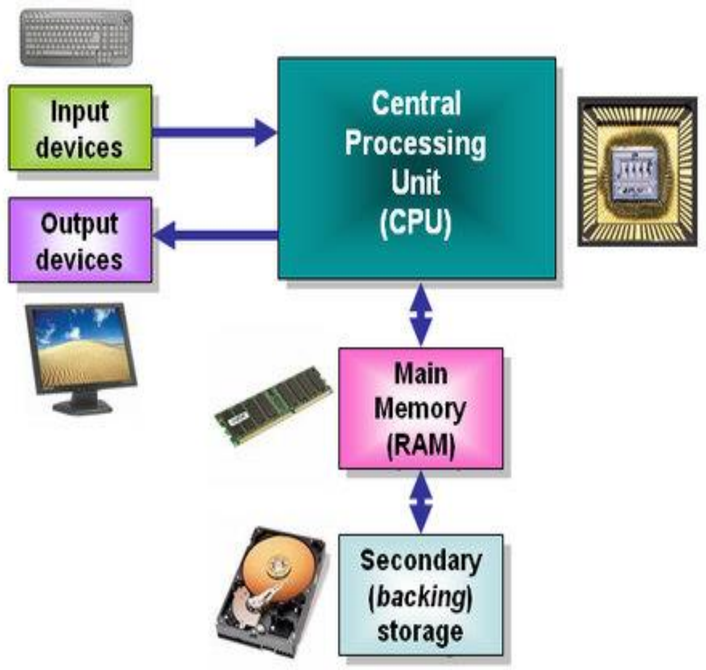
1.1 Computer System Components

- ***Hardware:*** provides basic computing resources (CPU, memory, I/O devices)
- ***Operating system:*** controls and coordinates the use of the hardware among the various application programs for the various users
- ***Applications programs:*** define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs)
- ***Users:*** people, machines, other computers



1.2 Von Neumann Architecture

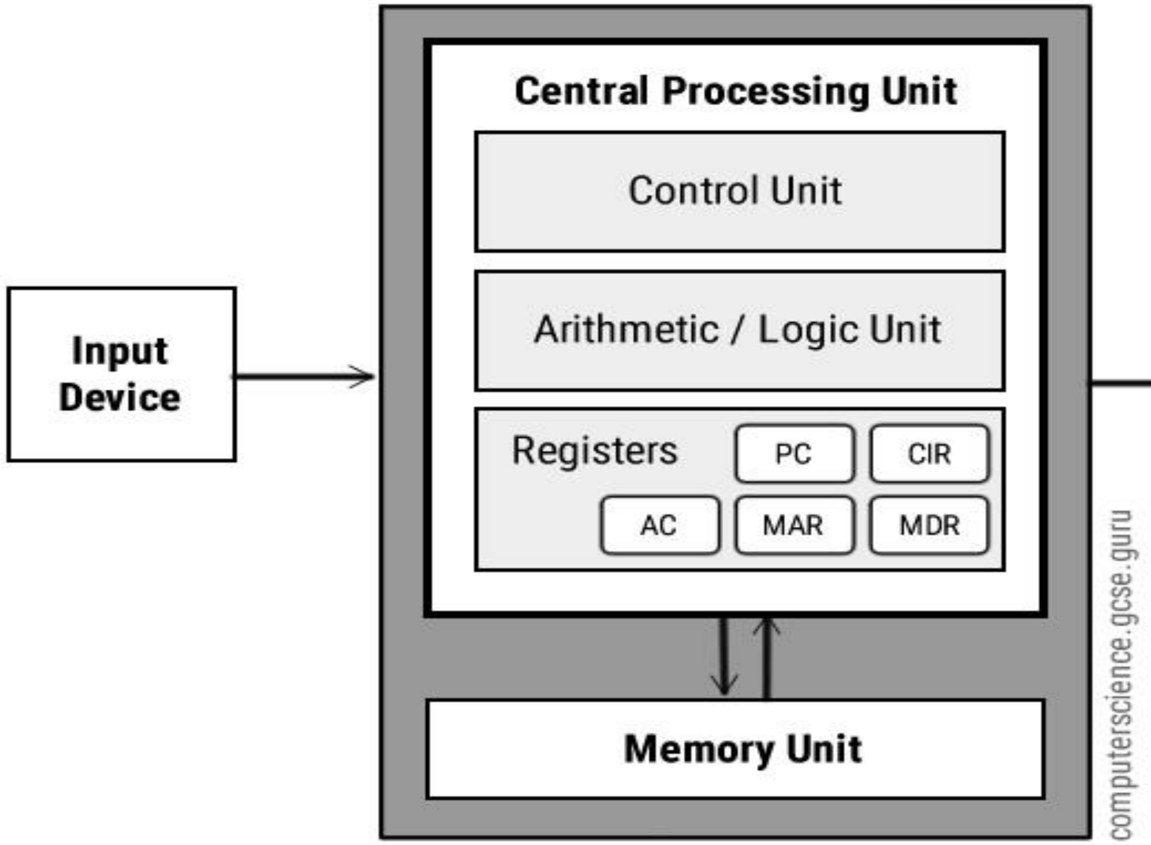
- Von Neumann architecture was first published by **John von Neumann** in 1945.
- His computer architecture design consists of
  1. Control Unit (CU)
  2. Arithmetic and Logic Unit (ALU)
  3. Memory Unit
  4. Registers
  5. Inputs/Outputs.
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the **same memory**.
- This design is still used in most computers **produced today**.



1.2.1 Central Processing Unit (CPU)

- The Central Processing Unit (CPU) is the electronic circuit responsible for executing the instructions of a computer program.
- It is sometimes referred to as the **microprocessor or processor**.
- The CPU contains the ALU, CU and a variety of registers.
- **1.2.1.1 Registers**
  - Registers are high speed storage areas in the CPU. All data must be stored in a register **temporarily** before it can be processed.

- CPU


Central Processing Unit
Control Unit
Arithmetic / Logic Unit
Registers
PC
CIR
AC
MAR
MDR
Memory Unit
Input Device
Output Device
computerscience.gcse.guru

1.2.1.1 Registers

- MAR   Memory Address Register

  Holds the memory **location** of data that needs to be accessed
- MDR Memory Data Register

  Holds **data** that is being transferred to or from memory
- AC Accumulator

  Where intermediate arithmetic and logic **results are stored**
- PC Program Counter

  Contains the **address of the next instruction** to be executed
- CIR Current Instruction Register

  Contains the **current instruction** during processing

• Instruction cycle (fetch , decode , execute cycle)

- This cycle describes how the CPU processes each instruction using its registers.

1. Fetch:

•The **Program Counter (PC)** sends the address of the next instruction to the **Memory Address Register (MAR)**.
•The instruction is fetched from memory and placed into the **Memory Data Register (MDR)**.
•The **MDR** then sends the instruction to the **Instruction Register (IR)**.
•The **PC** is incremented by 1 so it points to the next instruction.

2. Decode:

•The **Control Unit (CU)** decodes the instruction stored in the **IR** to determine the type of operation (e.g., add, load, jump).

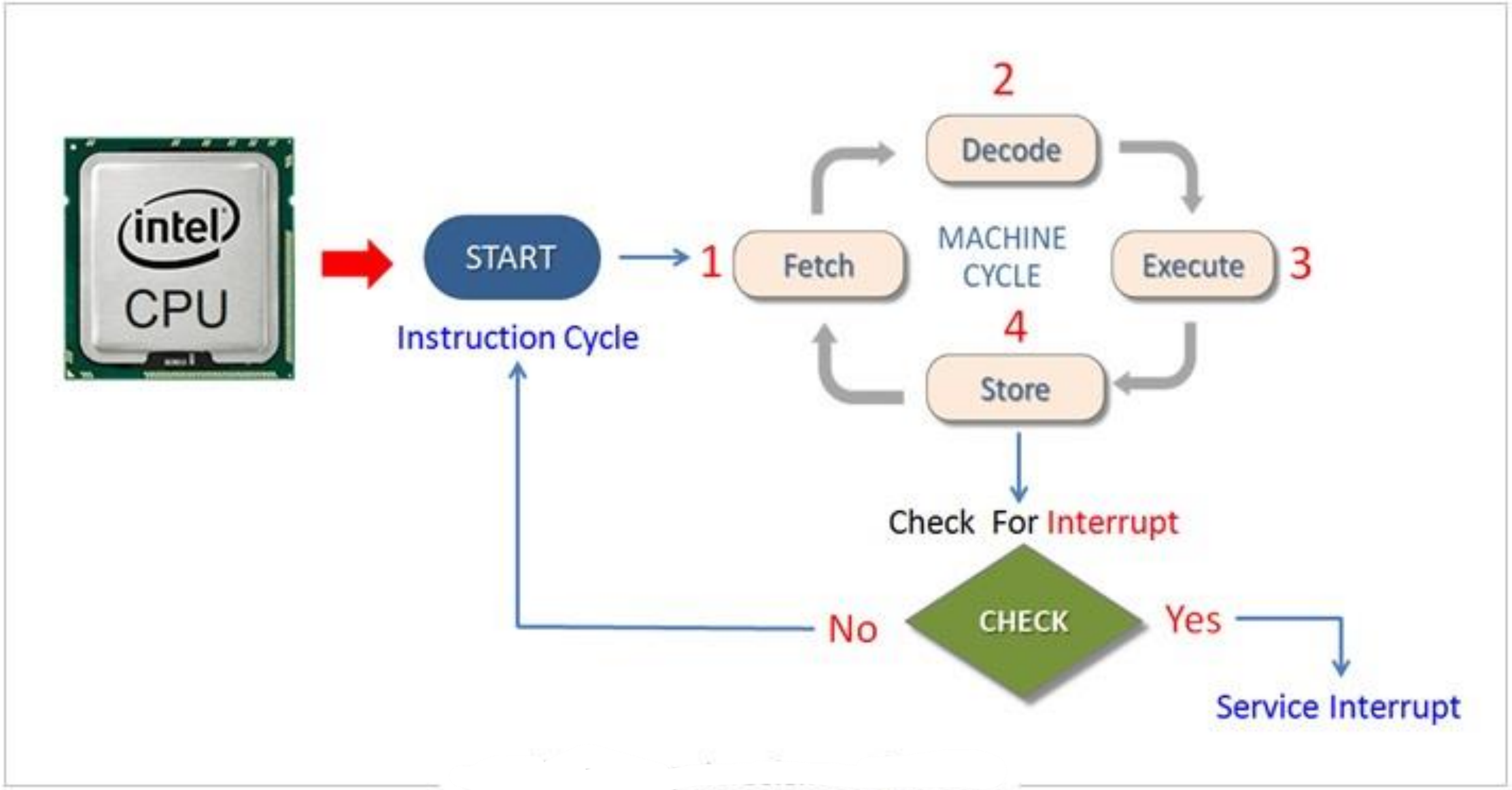
3. Execute:

•If it's an **arithmetic or logical operation**, the **Arithmetic Logic Unit (ALU)** performs the operation and stores the result in the **Accumulator (ACC)** or a **General Register**.

•If it's a **data transfer operation**, the **MDR** handles the data movement to or from memory.

•The **Status Register** is updated based on the result (e.g., zero, carry, overflow).

- Instruction cycle (fetch , decode , excute cycle)



1.2.1 Central Processing Unit (CPU)

- **1.2.1.2 Arithmetic and Logic Unit (ALU)**
  - The ALU allows arithmetic (add, subtract etc) and logic (AND, OR, NOT etc) operations to be carried out.
- **1.2.1.3 Control Unit (CU)**
  - The control unit controls the operation of the computer’s ALU, memory and input/output devices, telling them how to respond to the program instructions it has just read and interpreted from the memory unit.
  - The control unit also provides the **timing** and control **signals** required by other computer components.

1.2.3 Buses

- Buses are the means by which data is transmitted from one part of a computer to another, connecting all major **internal components to the CPU and memory.**
- A standard CPU system bus is comprised of a control bus, data bus and address bus.

Address Bus	Carries the **addresses** of data (but not the data) between the **processor** and **memory**
Data Bus	Carries **data** between the **processor**, the **memory** unit and the **input/output** devices
Control Bus	Carries **control signals/commands** from the **CPU** (and status signals from other devices) in order to control and coordinate all the activities within the computer & **HW**

1.2.4 Memory Unit

- The memory unit consists of **RAM** (Random Access Memory) and **ROM** (Read Only Memory), sometimes referred to as primary or main memory .
- Unlike a hard drive (**secondary memory**), this memory is fast and also directly accessible by the CPU.
- **RAM** is split into **partitions** (bytes). Each partition consists of an address and its contents (both in binary form).
- The address will **uniquely identify** every location (byte) in the memory.
- Loading data from permanent memory (secondary storage or hard drive), into the faster and directly accessible **temporary memory** (RAM), allows the CPU to operate **much quicker**.

1.2.4 Memory Unit

RAM	ROM
Volatile (loses data when the computer is turned off)	Non-volatile (retains data permanently)
Read and write	Mostly read-only
Stores data and programs during execution (runtime)	Stores permanent programs (Firmware/BIOS)
Very fast but temporary	Slower but permanent

- Search on its example

1.2.5 Input Devices

- Which are peripherals used to **provide** data and **control signals** to a computer.
- Input devices allow us to enter raw data for processing. For Example:
    - Keyboard (default input device)
    - Microphone
    - Scanner (2D and 3D)
    - Mouse
    - Trackball
    - Touchpad
    - Barcode and QR Code readers
    - Digital Camera

1.2.6 Output Devices

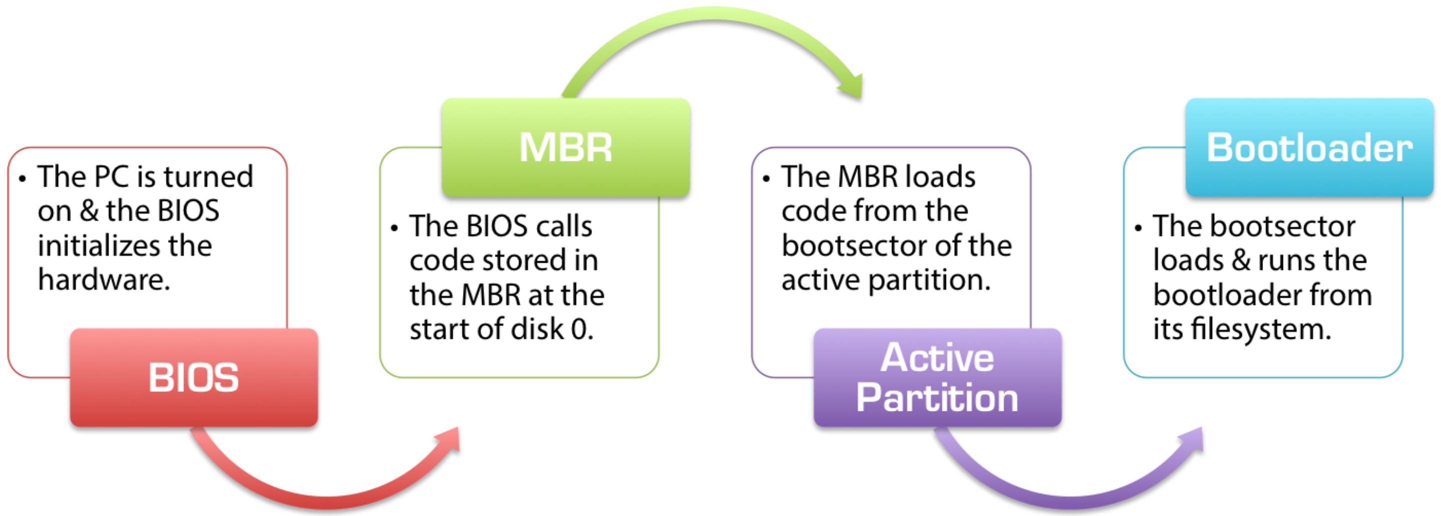
- Which are pieces of computer hardware used to **communicate the results of data processing** performed by a computer.
- The objective of output devices is to **turn computer information into a human friendly/readable form**.
- **For Example:**
  - Screen (Monitor or Console) (default output device) – LED or LCD
  - Data Projectors
  - Speaker and Headphones
  - Printer (2D and 3D) – inkjet or laser
  - Plotter (wide format printer)
  - Cutter (2D or 3D)

1.2.7 Input/Output Devices

- Some devices work as Input and Output devices like:
  - Touch Screen
  - Network
  - Most of I/O Ports (Both of serial and parallel)
  - New types of ports, like USB and Type-C ports
- All of the secondary storages used as I/O devices for permeant storage, like:
  - Hard disk
  - Floppy disk
  - Flash memory
  - CD and DVD

• Steps in the Booting Cycle

- Power On
- POST (Power On Self Test)
- BIOS/UEFI from ROM
- Bootloader (Boot sector)[MBR GPT]
- Load Kernel into RAM
- Initialize OS (drivers + services)
- Login Screen / Desktop


The PC is turned on & the BIOS initializes the hardware.
BIOS
MBR
The BIOS calls code stored in the MBR at the start of disk 0.
The MBR loads code from the bootsector of the active partition.
Active Partition
Bootloader
The bootsector loads & runs the bootloader from its filesystem.

1.3 Computer Operating System

- When a computer turns on, the processor will execute the instructions that are presented to it; generally, the first code that runs is for the boot flow, called ***bootstrap***, which is a framework stored in **ROM** contains some instructions called basic input output instructions (**BIOS**).
- For a computer that is used for general purposes and after it has booted up, there may be a variety of applications that need to be run on it **simultaneously**.
- Additionally, there could be a wide range of devices that could be connected to the computer (not part of the main system, for instance) **ex** keyboard , mouse and printer .
- All these need to be **abstracted** and handled efficiently and seamlessly. The user expects the system to “just work.” The ***operating system*** facilitates all of this and more.

1.3.1 What is an Operating System?

- **System** is: *some components integrated together for perform a desired task.*
- computer operating system consists of components integrated together for operating the computer system (HW &SW).
- The definition of the operating system is:

*a **program** that controls the execution of other programs running on the system. It acts as a facilitator and intermediate layer between the different software components and the computer hardware*

- When any operating system is built, it focuses on three main objectives:
  - *Efficiency* of the OS in terms of responsiveness, fluidity, and so on
  - *Ease of usability* to the user in terms of making it convenient Linux & win
  - *Ability to* **abstract** and ***extend*** to new *devices and software*

1.3.1 What is an Operating System?

- Most OSs typically have at least two main pieces:
  1. There is a core part that handles the complex, low-level functionalities and is typically referred to as the ***kernel*** *and* must be running at all times – resident in the main memory.

  2. There are generally some ***libraries***, ***applications***, and *tools* that are shipped with the OS.

     For **example**, there could be browsers, custom features, and OS-native applications that are bundled together.

1.3.1 Some Operating Systems

- list of operating systems that are commonly prevalent:
  - Microsoft Windows
  - GNU/Linux-based OS
  - macOS (used for Apple's computers and client models)
  - iOS (used for Apple's smartphone/tablet models)
  - Android
- All of these operating systems have different generations, versions, and upgrades.

1.3.2 OS Categories

- The OSs can be categorized based on the different methods in use.
- The two most common methodologies are by the ***usage type*** and the ***design/supported features*** of the OS.S

1.3.2.1 OS Categories – Usage Types

- Based on this, there are five main categories:
  - ***Batch***:

    For usages where a **sequence** of steps needs to be executed **repeatedly** without any human intervention. These classes are called batch OSs.

    Ex : (Old Mainframe and DOS)
  - ***Time Sharing***:

    For systems where **many users** (or many applications) access common hardware, there could be a need to timeshare the **limited resources**. The OSs in such cases are categorized as time-sharing OSs.

    Ex : (Windows, Mac, and Linux)
  - ***Parallel – Distributed (over tightly coupled systems)***:

    For hardware that is distributed physically and a **single OS** needs to coordinate their access, we call these systems distributed OSs.

    Ex :(AIX for IBM RS/6000 and Solaris for workstations)

1.3.2.1 OS Categories – Usage Types

- ***Network (loosly coupled)***:

  Another usage model, similar to the distributed scenario, is when the systems are connected over a network protocol, like IP (Internet Protocol), and therefore referred to as network OSs.

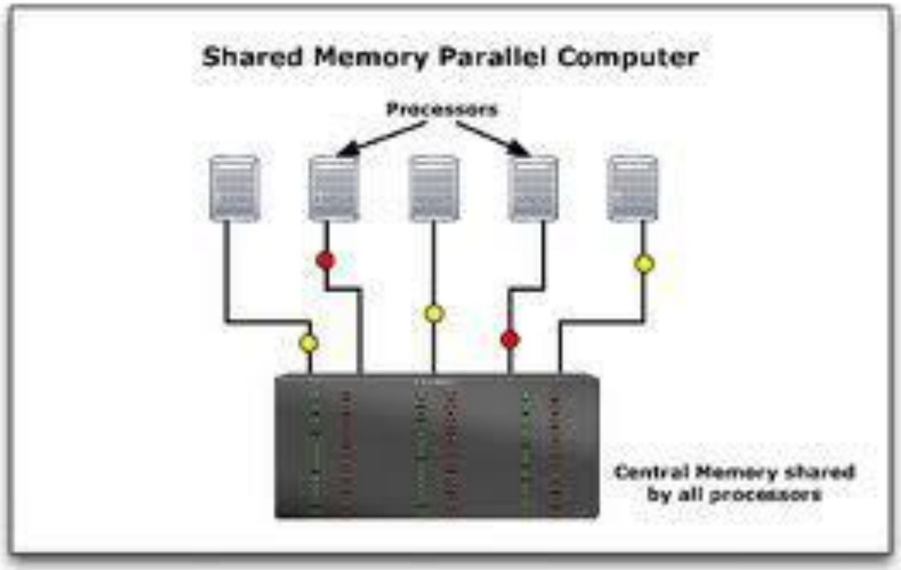
  Ex : (Windows server 2008, Novell Netware)

- ***Real Time***:

  In some cases, we need fine-grained time precision in execution and responsiveness. We call these systems real-time OSs.

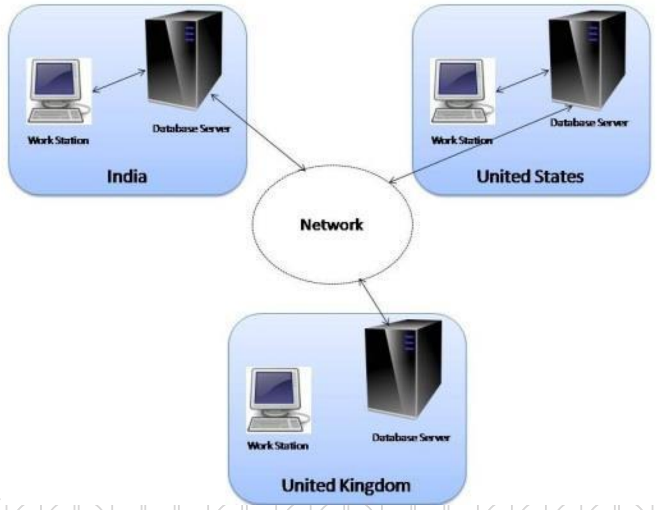
• Parallel Systems

- Systems with more than one CPU in close communication Also known as **multiprocessor systems**
- Tightly coupled system
  - processors share memory and a clock; communication usually takes place through the shared memory
- Advantages of parallel system:
  - Increased throughput
  - Economical
  - Increased reliability
  - graceful degradation
  - fail-soft systems


Shared Memory Parallel Computer
Processors
Central Memory shared by all processors

Distributed Systems

- Distribute the computation among several physical processors
- *Loosely coupled system*
  - Each processor has its own local memory
  - processors communicate with one another through various communications lines, such as high-speed buses or telephone lines
- Advantages of distributed systems
  - Resources Sharing
  - processing speed up
  - load sharing
  - Reliability



• Loosely and Tightly Coupled Multiprocessor System (Comparison)

	LOOSELY COUPLED **(Distributed systems)**	**TIGHTLY COUPLED** **(Parallel Systems)**
memory	Each processor has its own memory module.	Processors have shared memory modules.
Efficient	when tasks running on different processors, **has minimal interaction.**	Efficient for **high-speed processing**.
Memory conflict	No memory conflict.	more memory conflicts.

• Hard Realtime & Soft Realtime

	Hard Real-Time (HRT)	Soft Real-Time (SRT)
Definition	A system that must execute tasks within a *strictly defined* time limit (deadline) — absolutely no delays are allowed.	A system that *tries* to meet deadlines, but small delays are acceptable and not catastrophic.
Deadline	**Strict:** Missing the deadline by even a fraction of a second → catastrophic failure.	**Flexible:** Missing the deadline slightly → reduced performance, but the system continues working.
Importance	Delay = danger/failure (e.g., an aircraft might crash if the control system is late).	Delay = lower quality (e.g., video lag or stutter).
Response Time	Must be **deterministic** and always within the required time.	Should be fast, but occasional delays are acceptable.
Applications	- Aviation systems - Medical devices (e.g., pacemaker) - Automotive safety systems (Airbags, ABS brakes)	- Video games - Video calls

1.3.2.2 OS Categories – Designed and Supported Features

- Based on this, there are three main categories:
  - ***Monolithic***:

    In this case, the entire OS is running in a high-privilege **kernel** space and acts as the **supervisor for all other programs** to run. Common monolithic OSs include many of the UNIX **flavors**.
  - ***Modular***:

    In some OSs, a few parts of the OS are implemented as so-called **plug-and-play modules** that can be updated independent of the OS kernel

    - plug-and-play , flexibility and reliability .
    - Many modern OSs follow this methodology, such as Microsoft Windows, Linux flavors, and macOS.
  - ***Micro-service based***:

    More modern OSs are emerging and leverage the concept of micro-services where many of the previously monolithic OS features may be **broken down into smaller parts** that run in either the **kernel** or **user mode**. The micro-service approach helps in assigning the **right responsibility** of the components and easier **error tracking** and **maintenance**.

    Some versions of Red Hat OS support micro-services natively.

1.4 Why We Need an OS?

- As a global view we need the OS for perform the following tasks:
  - **Application Excution** It makes it easier for them to use hardware resources (CPU, memory, I/O).
  - **Run** and **facilitate** different applications running on the system.
  - **Manage conflicts** among different applications.
  - In practice, there are many **common features** that may be needed by your programs including, for example, security, which would have **services** like encryption, authentication, and authorization
  - **Abstracts the HW and facilitates** the seamless execution of our applications using the system.
  - **Easy execution programs**

1.4.1 Complex and Multiprocessor Systems

- Many modern computing architectures support **microprocessors** with **multiple CPU cores.**
- When all cores provide the **same** or identical capabilities, they are called as ***homogeneous platforms***.
- There could also be systems that provide different capabilities on **different** CPU cores. These are called ***heterogeneous platforms***.
- There are also additional execution engines such as Graphics Processing Units ***(GPUs)***, which accelerate graphics and 3D processing and display

1.4.1 Complex and Multiprocessor Systems

- An operating system supporting such a platform will need to ensure ***efficient scheduling*** of the different programs on the different execution engines (cores) available on the system.
- Similarly, there could be **differences in the hardware** devices on the platform and their capabilities such as the type of display used, peripherals connected, storage/memory used, sensors available, and so on.
- The OS would also be required to **abstract** the differences in the hardware configurations to the applications.

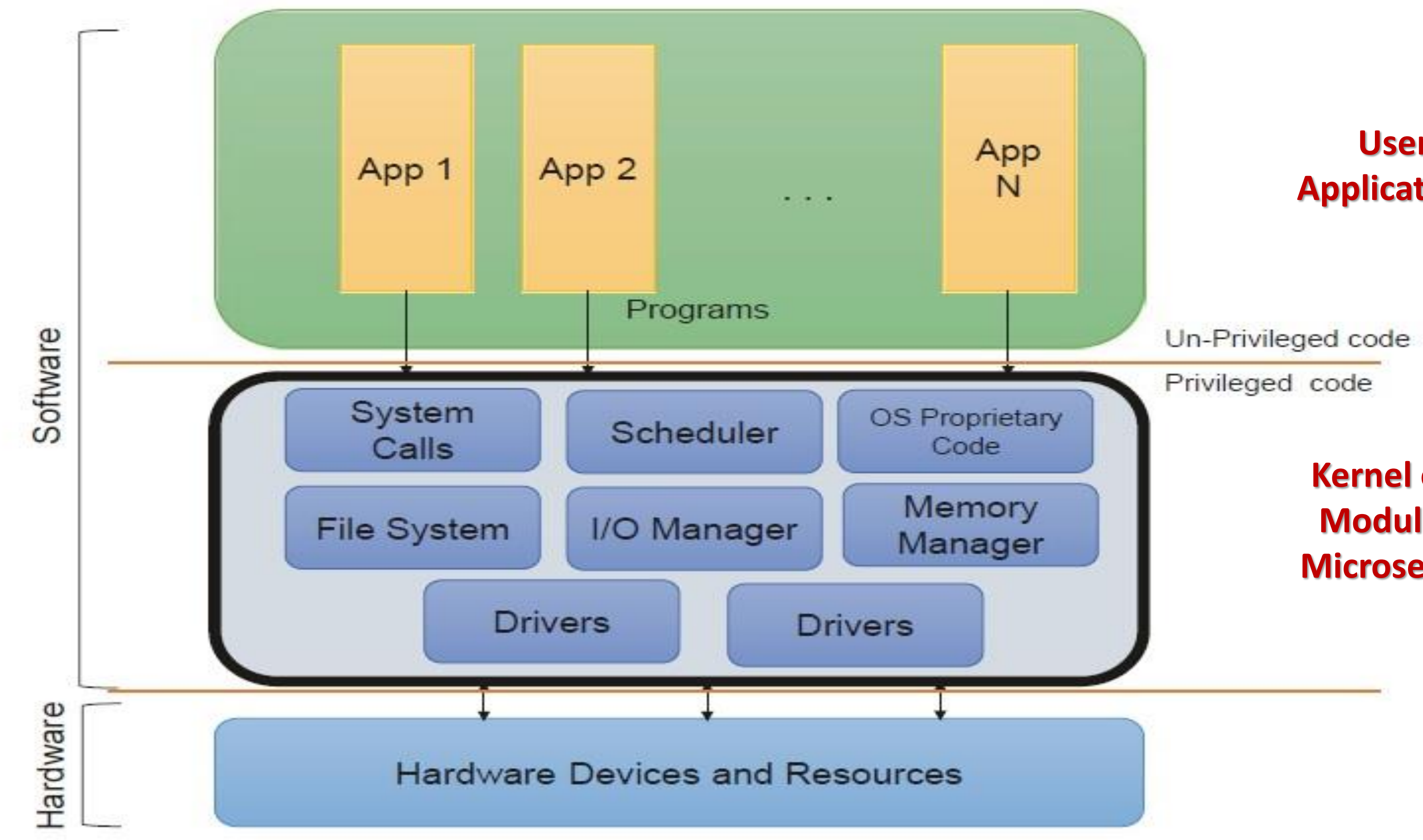
1.4.2 Multitasking and Multifunction Software

- In general, there could be ***many applications*** that may need to be running on the system ***at the same time***.
- These could include applications that the ***user initiated***, so-called "***foreground***" applications, and applications that the ***OS has initiated*** in the "***background***" for the effective functionality of the system.
- Multifunction **ex** Microsoft office .
- It is the OS that ensures the streamlined execution of these applications.

1.4.3 Multiuser Systems

- Often, there could be more than one user of a system such as an ***administrator*** and ***multiple*** *other users* with ***different*** levels of ***access permission*** who may want to utilize the system.
- It is important to ***streamline execution*** for each of these users so that they do not find any perceived *delay of their* ***requests***.
- At the same time, there need to be **controls** in place to manage **privacy and security** between users. The OS facilitates and manages these capabilities as well.
- It is the role of the operating system to handle these complexities in a **consistent**, **safe**, and **performant** manner. Most general-purpose OSs in use today, such as Windows, Linux, macOS, and so on, provide and handle most of the preceding complexities.

1.5 Operating System Components



User Applications

Kernel or OS Modules or Microservices

1.5 Operating System Components

- Typically, there are **APIs** (application programming interfaces) that are exposed to access system resources. These APIs are then **used by programs** to **request** for communicating to the **hardware**.
- The first one to use is IBM

- While the communication happens, there could be **requests from multiple programs and users at the same time**. The OS streamlines these requests using efficient scheduling algorithms and through management of I/Os and handling conflicts.

1.6 Why Is It Important to Know About the OS?

- Software *developers must have a good understanding of the **environment, the OS, that their code is running in***.
- It's important for you to keep in mind the OS interfaces and **functionality** as this will impact the software being developed.
- For Example:
  - the choice of **language** and needed **runtime features** may be OS dependent ex .NET and java .
  - the choice of inter-process communication **(IPC)** protocols used for messaging between applications will depend on the OS offerings
- During development and **debug**, there could be usages where the developer may need to understand and **interact with the OS**.
- For example, debugging a **slowly performing or nonresponsive** application may require some understanding of how the OS performs input/output operations.

1.6 Why Is It Important to Know About the OS?

- some questions that may come up during the debug:
  - Are you accessing the file system too often and writing repeatedly to the disk?
  - Is there a garbage collector in place by the software framework/SDK?
  - Is the application holding physical memory information for too long?
  - Is the application frequently creating and swapping pages in memory? What it the average commit size and page swap rate?
  - Is there any other system event such as power event, upgrades, or virus scanning that could have affected performance?
  - Is there an impact on the application based on the scheduling policy, application priority, and utilization levels?
  - Security and auth .
- If an application needs to work with a custom device, it must use low-level OS APIs for communication .
- Developers should understand these APIs, use the OS's capabilities, and follow its authentication and permission protocols.

1.7 Responsibilities of an OS

- The OS needs to be able to **abstract** the complexities of the underlying hardware, support multiple users, and facilitate execution of multiple applications at the same time. The following table describe the Requirements and Solutions

Requirements	Solution
Applications require **time on the CPU** to execute their instructions.	The OS shall implement and abstract this using suitable **scheduling** algorithms.
Applications require **access to system memory** for variable storage and to perform calculations based on values in memory.	The OS shall implement **memory management** and provide APIs for applications to utilize this memory.
Each software may need to access different **devices** on the platform.	The OS may provide **APIs for device and I/O management** and interfaces through which these devices can be communicated.

1.7 Responsibilities of an OS

Requirements	Solution
There may be a need for the user or applications to save and read back contents from the **storage**.	Most OSs have a **directory and file system** that handles the storage and retrieval of contents on the disk.
It is important to perform all of the core operations listed in the preceding **securely** and efficiently.	Most OSs have a **security subsystem** that meets specific security requirements, virtualizations, and controls and balances.
Ease of access and usability of the system.	The OS may also have an additional **GUI** (graphical user interface) in place to make it easy to use, access, and work with the system.

1.7 Responsibilities of an OS

- To summarize, the OS performs different functions and handles multiple responsibilities for software to co-exist, streamlining access to resources, and enabling users to perform actions. They are broadly classified into the following functional areas:
  - Scheduling
  - Memory management
  - I/O and resource management
  - Access and protection
  - File systems
  - User interface/shell

In which programming language the OS written ?

- The **UNIX** operating system's development started in 1969, and its code was **rewritten in C in** 1972. The C language was actually created to move the UNIX kernel code from assembly to a higher level language.

- In 1985 **Windows** 1.0 was released. Although Windows source code is not publicly available, it's been stated is **mostly written in C** with some parts in assembly.

- **Linux** kernel development started in 1991, and it is also **written in C**.

In which programming language the OS written ?

- Most of the operating systems are written in the C/C++ languages.

- These not only include Windows or Linux (the Linux

kernel is almost entirely written in C), but also Google Chrome OS, RIM Blackberry OS 4.x, Symbian OS, Apple Mac OS X, IPAD OS, Apple iPhone iPod Touch, and Cisco IOS (which is mainly comprised of compiled C and C++ code).

Thank You

With My Best Wishes

Zeyad ashraf