Operating System

Definition:

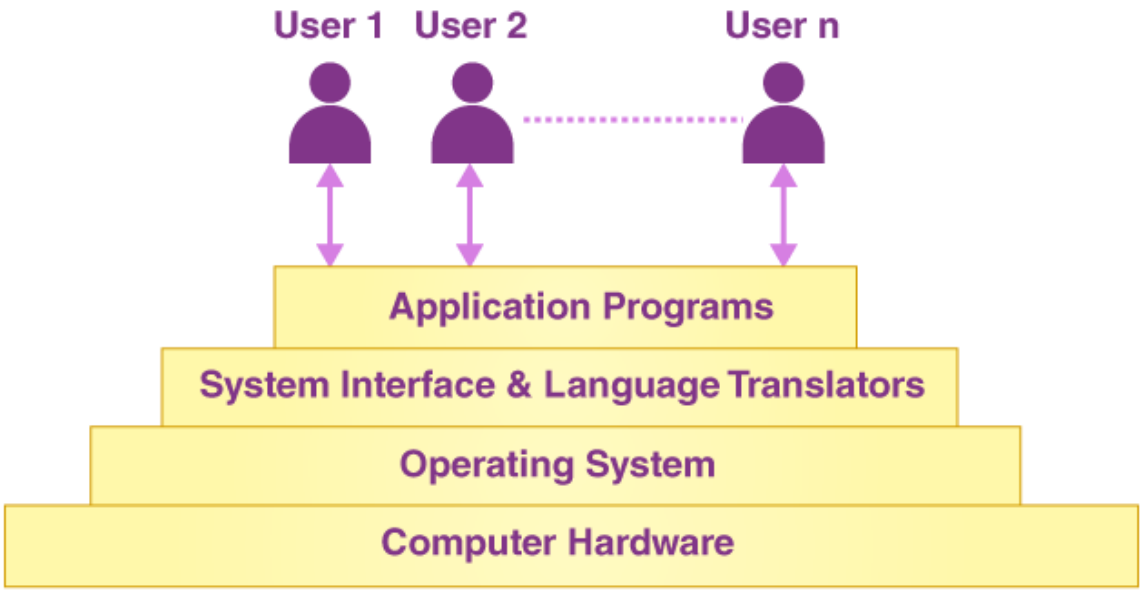
An operating system (OS) is system software that acts as an intermediary between computer hardware and application software, providing services and managing resources to facilitate efficient and secure computer operation.

Q. Why there is a need of an OS?

- Computer is a machine/hardware does different tasks efficiently & accurately.
- Basic functions of computer:

  1. Data Storage
  2. Data Processing
  3. Data Movement
  4. Control

- As any user cannot communicates/interacts directly with computer hardware to do different tasks, and hence there is need of some interface between user and hardware.



Operating System (OS) Overview

An Operating System (OS) is a critical component of a computer system, acting as a system software that facilitates communication between the user, programs, and hardware. Here are key aspects of an operating system:

1. **Interface Between User and Hardware:**
    - An OS serves as a crucial intermediary, enabling users to interact with the hardware components of a computer system without needing to have direct knowledge of hardware operations.
2. **Resource Allocation:**
    - One of the fundamental roles of an OS is to allocate resources to various running programs. These resources include:
        - Main Memory: Allocating memory for program execution.
        - CPU Time: Managing and distributing processing time among programs.
        - I/O Devices Access: Controlling communication with input/output devices.
    - The OS is, therefore, referred to as a **"resource allocator"** for its role in distributing and managing resources efficiently.
3. **Control Program:**
    - An OS has control over the execution of all programs within the system. It ensures the orderly execution of processes and manages the interactions with hardware devices connected to the computer and hence it is also called as a **"control program."**
4. **Resource Manager:**
    - As the OS oversees the distribution of limited resources among multiple running programs, it is also known as a **"resource manager."** This involves making decisions on resource utilization to optimize overall system performance.

Components of an Operating System

1. **Kernel:**
    - Core part of the OS, responsible for essential functions.
    - Manages hardware resources and provides low-level services.
2. **Shell:**
    - Interface between the user and the kernel.
    - Command-line or graphical interface for user interaction.
3. **File System:**
    - Hierarchical organization of files and directories.
    - Manages storage and retrieval of data.
4. **Device Drivers:**
    - Software components that enable communication with hardware devices.
    - Translate generic OS commands into device-specific commands.
5. **Utilities:**
    - Additional software tools for system management and maintenance.
    - Examples include disk utilities, system monitors, and security tools.

Key Concepts

1. **Process:**
    - An instance of a program in execution.
2. **Thread:**
    - Smallest unit of a process, representing the execution of a set of instructions.
3. **Semaphore:**
    - Synchronization mechanism for controlling access to resources in a multi-process environment.
4. **Deadlock:**
    - A situation where two or more processes are unable to proceed due to each waiting for the other to release a resource.
5. **Interrupts:**
    - Signals to the processor that require immediate attention.
    - Used for asynchronous events and hardware interrupts.
6. **Virtual Memory:**
    - Extends the available RAM by using disk space as an overflow.
    - Allows more extensive applications to run than physical memory alone would support.

OS from an End User's Perspective

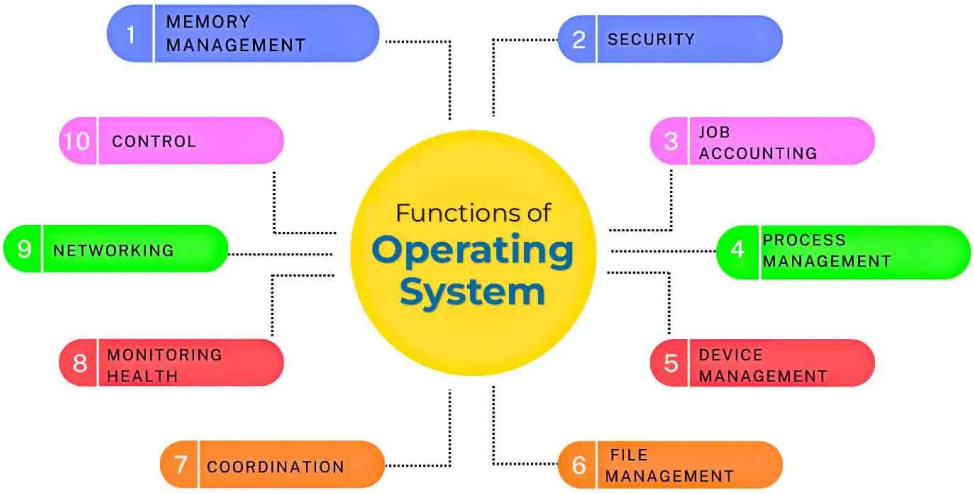
From the end user's viewpoint, an Operating System (OS) is a comprehensive software package that typically comes in CDs or DVDs. It consists of several key components:

1. **Kernel:**
    - The **kernel** is a core program or component of the OS that runs continuously in the main memory. It performs basic minimal functionalities essential for the operating system's operation.
    - Examples of kernel files for different operating systems include:
        - Linux: vmlinuz
        - Windows: ntoskrnl.exe
2. **Utility Software:**
    - The OS includes **utility software**, which encompasses various tools and programs that provide additional functionalities beyond the basic OS operations. Some examples of utility software are:
        - Disk Manager: Manages and organizes data on storage devices.
        - Windows Firewall: Controls incoming and outgoing network traffic for security.
        - Anti-virus Software: Protects the system from malicious software and threats.
3. **Application Software:**
    - The OS also incorporates **application software**, which consists of programs designed for end-users to perform specific tasks. Examples of application software include:
        - Google Chrome, Shell, Notepad, MS Office

Functions of an OS

Basic minimal functionalities/Kernel functionalities:

1. Process Management
2. Memory Management
3. Hardware Abstraction
4. CPU Scheduling
5. File & IO Management **Extra utility functionalities/optional:**
6. Protection & Security
7. User Interfacing
8. Networking



Program and Process in Operating Systems

1. Program:

- A **program** is a set of instructions written in a programming language to perform a specific task or achieve a particular goal.
- Programs are typically stored as executable files on storage devices like hard drives or SSDs.
- Examples of programs include word processors, web browsers, games, and compilers.

2. Process:

- A **process** is the execution of a program in the computer's memory. When a program is loaded into the main memory for execution, it becomes a process.
- The execution of a process involves the allocation of system resources such as CPU time, memory space, and I/O devices.
- Processes have a life cycle, including states like New, Ready, Running, Blocked, and Terminated.

Loader:

- The **loader** is a system program, part of the operating system, responsible for loading an executable file from a storage device (e.g., hard disk) into the main memory.
- It is a crucial component that initiates the execution of a program by transferring it from non-volatile storage to volatile memory.

Operating System's Role:

- **Starting Execution:**
  - The **operating system (OS)**, through the loader, initiates the execution of a program by loading it into the main memory.
- **Allocating CPU Time:**
  - The OS manages the allocation of CPU time to processes. Each process is given a fair share of CPU resources to execute its instructions.
- **IO Device Control:**
  - The OS controls the interaction of processes with I/O devices. It ensures that processes can communicate with peripherals like keyboards, displays, and storage devices.

Resources Required for Program Execution:

- For the successful completion of program execution, various resources are essential:
    1. **Main Memory (RAM):** The program's instructions and data need to be loaded into the main memory (RAM) for execution.
    2. **CPU Time:** The Central Processing Unit (CPU) allocates time to execute the program's instructions.
    3. **Input Device Access (e.g., Keyboard - KBD):** If the program interacts with user input, access to input devices is required.
    4. **Output Device Access (e.g., Monitor):** Output devices like monitors are necessary for displaying results or interacting with the user.

Installation of OS:

- The installation of an Operating System (OS) involves the process of storing OS programs onto the hard disk drive (HDD).
    - The OS programs include the kernel, utility software, and application software.
    - This process is typically performed during the initial setup of a computer system.

Booting Process:

- **Booting** is the process by which a computer system starts up and prepares for operation. It involves several key steps:
    1. **Bootstrap Program:**
        - The **bootstrap program** is a special program stored in the boot sector, which is the first sector of the HDD.
        - It is a small program that initiates the booting process and is capable of locating the OS kernel on the hard disk.
    2. **Bootstrap Loader:**
        - The **bootstrap loader** is responsible for loading the bootstrap program into the main memory (RAM).
        - Once in memory, the bootstrap program takes control and begins the process of locating and loading the OS kernel.
    3. **Locating and Loading Kernel:**
        - The bootstrap program locates the OS kernel on the hard disk drive and loads it into the main memory.
        - The kernel, being the core part of the OS, starts its execution, initiating the operating system.
    4. **Initialization and OS Start-up:**
        - The OS, after being loaded into memory, initializes various components, sets up necessary data structures, and prepares the system for user interaction.
        - The OS then starts its execution, providing a platform for running user programs.

Computer Fundamentals

Components of a Computer:

- **Processor (CPU):** Responsible for executing instructions and performing calculations. It is often considered the brain of the computer.
- **Memory Devices (RAM, ROM):** RAM (Random Access Memory) is used for temporary data storage, while ROM (Read-Only Memory) stores permanent instructions for the computer.

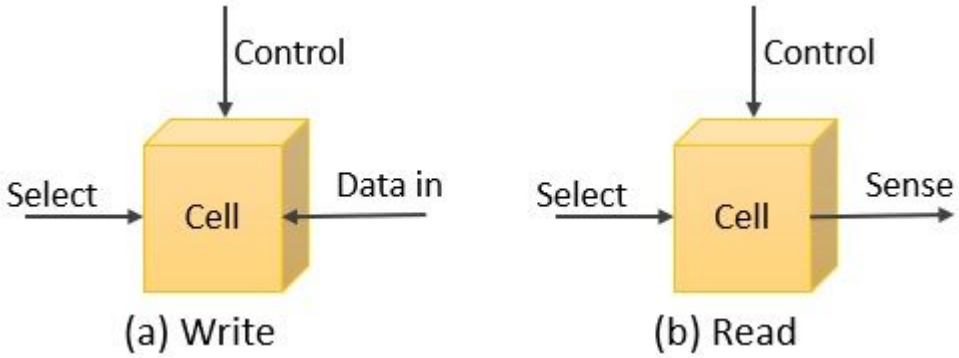
- **I/O Devices (Input/Output):** Examples include keyboards, mice, monitors, and printers. These devices allow users to input data into the computer and receive output.

There are two basic/fundamental **structural** and **functional** units of computer hardware system:

Memory Cell

Basic storage unit capable of holding a single binary value (0 or 1). It is where data is stored temporarily during processing.

- **Binary Storage:** Memory cells store binary data, representing 0s and 1s. The combination of these binary values forms the basis for digital information storage and processing.



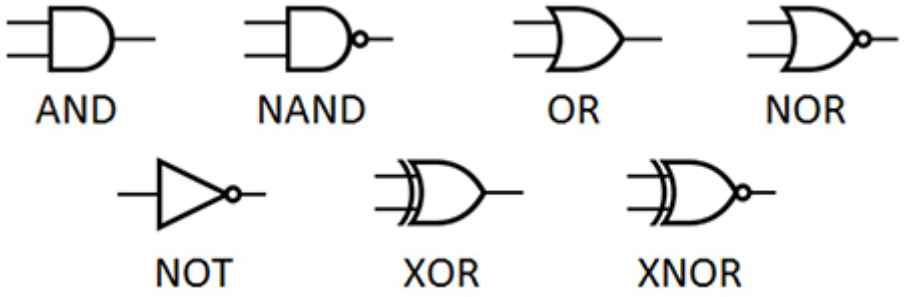
1. **Memory Cell and Bit:**
    - A memory cell can store either a 0 or a 1, representing a binary digit, commonly referred to as a "bit (**bi**nary digi**t**)."
    - A byte consists of 8 bits.
    - The smallest addressable unit in memory is typically one byte. When the processor reads or writes data, it interacts with data in units of bytes.
2. **Big Endian vs. Little Endian:**
    - **Big Endian Format:** Data is stored in memory in its binary equivalent format as it is.
    - **Little Endian Format:** Data is stored in memory in the reverse order of its binary equivalent format.
    - The "Pentium family processors" follow the little endian format, meaning they store the least significant byte at the lowest memory address.

      Example Using an Integer (num = 300):
    - Integer `num` with a value of 300 is represented in binary as `00000000 00000000 00000001 00101100` (32 bits).
    - In big endian format, it is stored as `00000000 00000000 00000001 00101100`.
    - In little endian format, it is stored as `00101100 00000001 00000000 00000000`.
    - The most significant bit **(MSB) is on the left**, and the least significant bit **(LSB) is on the right.**

Gates

Basic logic gates (e.g., AND, OR, NOT) are fundamental components used to perform logical operations and make decisions in a computer system.

- **Digital Logic Operations:** Gates perform basic logical operations such as AND, OR, and NOT. These operations are the building blocks for more complex arithmetic and logical operations in a computer system.
- **Arithmetic and Logic Operations:** Gates are crucial for executing mathematical calculations and logical comparisons.



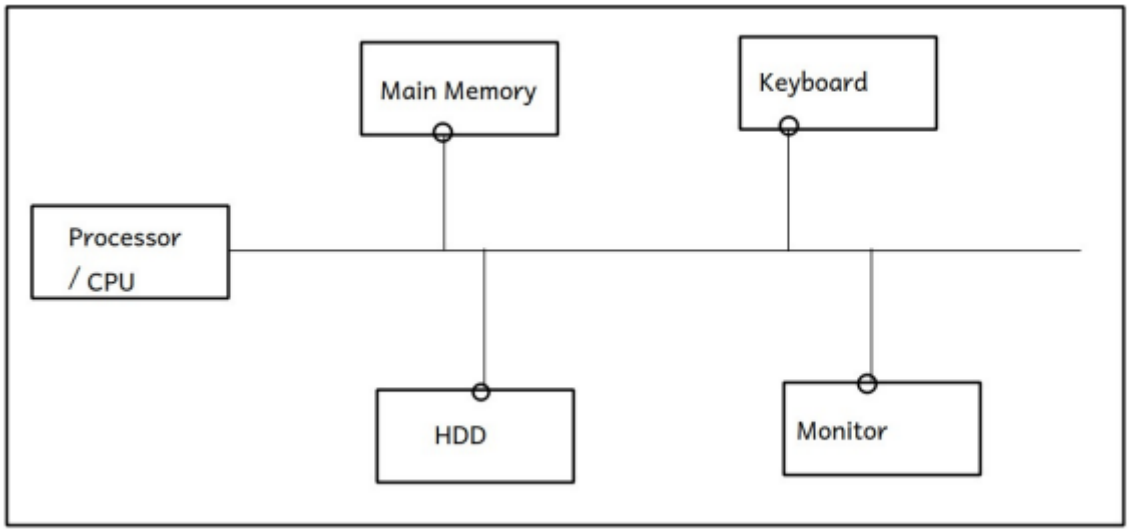
- As the data storage can be acheived by memory cell, data processing, data movement and control functions can be achieved by using gates, hence memory cell & gates are referred as functional units
- Wheras computer is a digital device made up of collection of millions of memory cells & gates, and hence memory cell & gates are referred as structural units as well.

Processor

The CPU, or Processor, is often referred to as the brain of the computer. It interprets and executes instructions from the computer's memory. It performs calculations, manages data, and controls the flow of information within the computer system.

1. **Processor Components:**
    - **ALU (Arithmetic Logic Unit):** The ALU is responsible for performing arithmetic and logic operations. It carries out tasks such as addition, subtraction, multiplication, division, and logical comparisons.
    - **CU (Control Unit):** The Control Unit manages and coordinates the activities of the computer's components. It fetches instructions from memory, decodes them, and executes the necessary control signals to coordinate the flow of data within the CPU and between the CPU and other parts of the computer.
    - **Registers:** Registers are small, high-speed storage locations within the CPU. They store data that is being actively used or processed. Examples include the instruction register (IR) and program counter (PC).
2. **Computer System Components:**
    - **Main Memory (RAM):** This is the volatile memory that the computer uses to temporarily store data that is actively being used or processed. It is faster than secondary storage but loses its contents when the power is turned off.

- **Hard Disk Drive (HDD):** The HDD is a non-volatile storage device that stores data on magnetic disks. It provides long-term storage for the operating system, applications, and user files.
- **Input Devices (e.g., Keyboard)**
- **Output Devices (e.g., Monitor)**



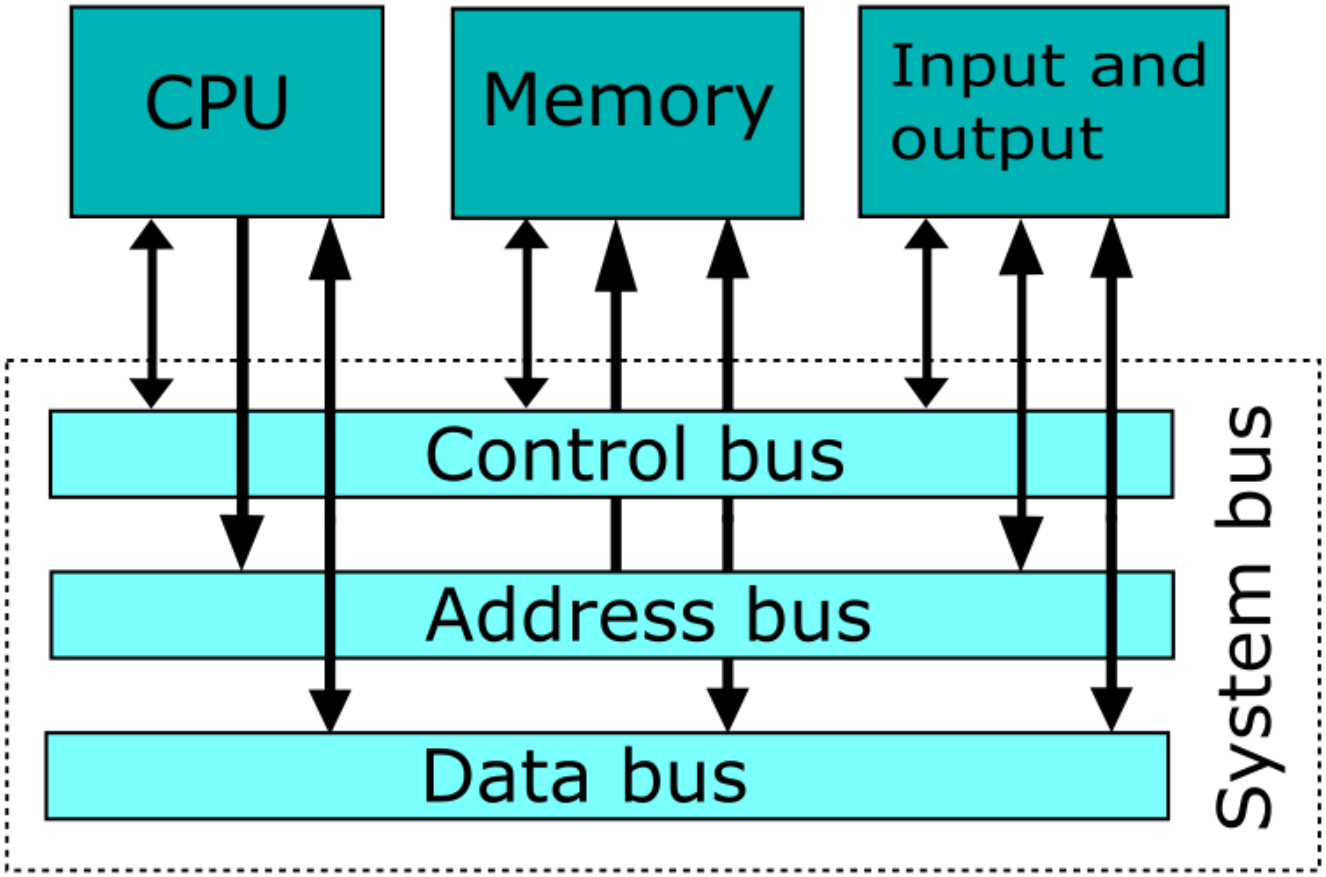
3. **Device-Specific Processors:**
   - Some devices in a computer system have dedicated processors to control their operations. For example, a graphics processing unit **(GPU)** is designed to handle graphics-related tasks, and a network interface card **(NIC)** may have its own processor for networking operations. Hard Disk Drive has **disk controller** which controls all disk operations.

The CPU acts as the central coordinator of the entire computer system. It communicates with dedicated processors for various devices and ensures the smooth functioning and coordination of different components, allowing them to work together seamlessly.

Buses

All communication between different components in a computer system occurs through conducting wires known as buses. These buses serve as communication pathways that allow the transfer of data, addresses, and

control signals.



Types of Buses:

- **Data Bus:** This bus is responsible for transferring actual data between the CPU, memory, and other peripheral devices. It carries the binary information that represents instructions, numbers, or any other data.
- **Address Bus:** The address bus is used to specify the memory locations for data transfer. It carries the address information that indicates where the data should be read from or written to in the memory.
- **Control Bus:** The control bus carries control signals that coordinate and manage the activities of different components. These signals include commands such as read, write, interrupt requests, and clock signals that synchronize operations.
- Since the data sent by one component can be received by any other component connected to the bus, the bus serves as a **shared communication pathway.** This shared nature allows for efficient and coordinated communication among different parts of the computer system.

Role of System Bus in Data Transfer:

- The major components of a computer system, such as the CPU, Cache Memory, and Main Memory, are interconnected by a common bus known as the **System Bus.**
- When a program is executed, instructions and data move between the CPU and memory. The system bus plays a vital role in facilitating this data transfer. The CPU uses the address bus to specify where in memory it wants to read or write data, and the data bus carries the actual information.

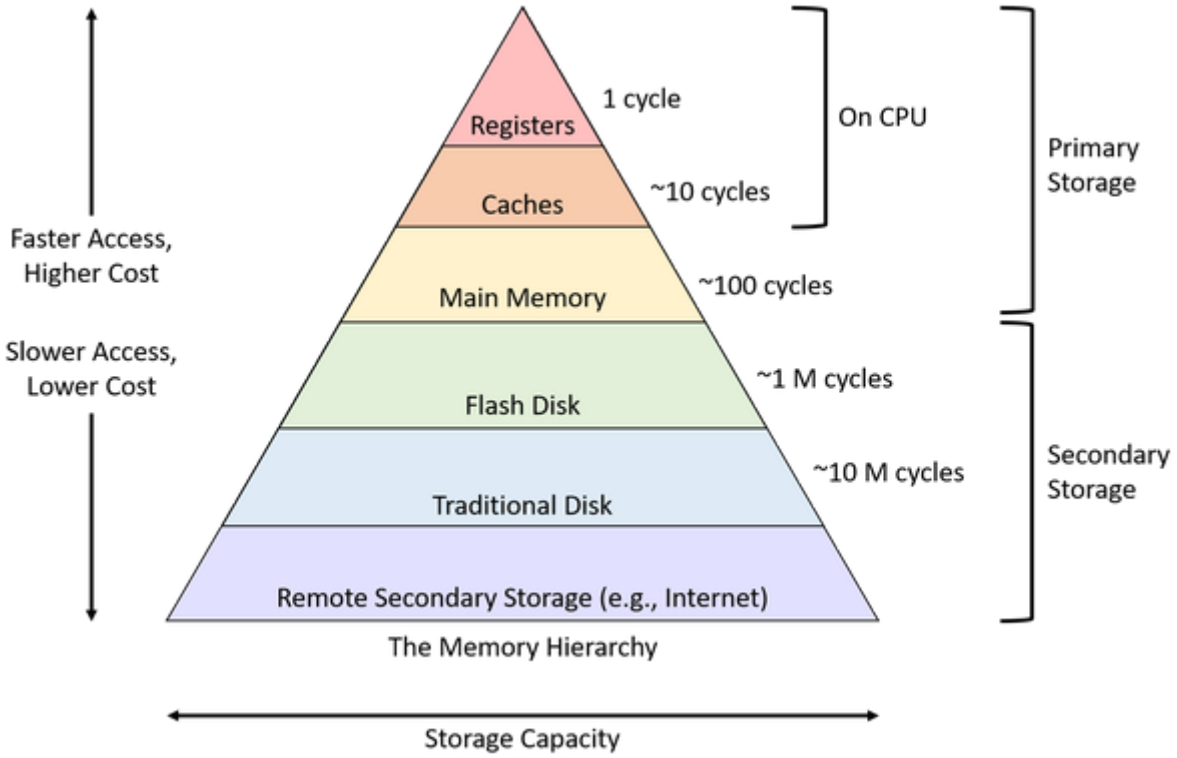
Expansion Slots and Peripheral Buses:

- In addition to the system bus, computers often have expansion slots for additional components like graphics cards, network cards, and other peripherals. These expansion slots are connected to peripheral buses, which allow for the integration of additional functionalities into the system.

Control signals sent by the CPU to other devices referred as commands:

1. **TEST:**
    - The TEST command is used by the CPU to check the status or condition of a particular device. This command allows the CPU to inquire about the current state of a device, such as whether it is ready to receive or send data, or if there are any errors or issues.
2. **WRITE:**
    - The WRITE command instructs a device to accept data from the CPU. It is used when the CPU wants to transfer information or send data to a specific device, such as writing data to the memory, storage devices, or output devices.
3. **READ:**
    - The READ command directs a device to transfer data to the CPU. It is employed when the CPU needs to retrieve information from a specific device, like reading data from memory, storage devices, or input devices.
4. **CONTROL:**
    - The CONTROL command is a general term that encompasses various signals used by the CPU to manage and coordinate the activities of different devices. This can include signals to initiate or halt operations, synchronize processes, or handle specific functions unique to the device.

CPU Memory Technologies



1. **CPU Registers:**
    - Registers are small, high-speed storage locations located within the CPU. They are at the top of the computer memory hierarchy and are used to temporarily store instructions and data that are currently being executed by the CPU.
    - Examples of CPU registers include
        - **MAR:** Memory Address Register
        - **MBR:** Memory Buffer Register
        - **IOAR:** Input/Output Address Register
        - **IOBR:** Input/Output Buffer Register
        - **PC:** Program Counter
        - **SP:** Stack Pointer, and Accumulator.
2. **Internal (Cache) & External Memory:**
    - **Internal memory** refers to the memory components that are part of the motherboard. This includes CPU registers, L1 and L2 cache, Cache memory, and RAM (Random Access Memory).
    - **External memory** is located outside the motherboard and includes storage devices such as magnetic disks, optical disks (CD/DVD), magnetic tapes, and other external storage media.
3. **Primary & Secondary Memory:**
    - **Primary memory** is directly accessible by the CPU. It includes memory components that can be accessed using the CPU's instruction set.
        - Examples include CPU registers, L1 and L2 Cache, Cache Memory, and RAM.
    - **Secondary memory** cannot be accessed directly by the CPU and requires data to be transferred to primary memory before the CPU can access it.

        - Examples include magnetic disks (Hard Disk Drives), CDs/DVDs, and portable storage devices (e.g., USB drives).

    - When the CPU needs to access data from a disk (secondary memory), it is first fetched into the RAM (primary memory). This is because accessing data from RAM is much faster than accessing it directly from secondary memory.

4. **RAM (Main Memory):**

    - RAM, or Random Access Memory, is a type of primary memory that is crucial for the execution of programs. It is volatile memory, meaning it loses its content when power is turned off. RAM is used to store data and instructions that the CPU actively needs during program execution.

    - RAM is often referred to as Main Memory because it is essential for the normal operation of a computer. It provides the working space for the CPU and allows for fast access to data and instructions required for program execution.

Methods of accessing data from computer memory:

1. **Sequential Access:**

    - **Example:** Magnetic Tape
    - **Characteristics:** Data is accessed in a linear, sequential manner. To reach a specific piece of data, the system must **traverse** through the preceding elements.

2. **Direct Access:**

    - **Example:** Magnetic Disk(HDD)
    - **Characteristics:** Allows direct access to any data element, without the need to traverse the preceding elements. Data is accessed through **disk block addresses.**

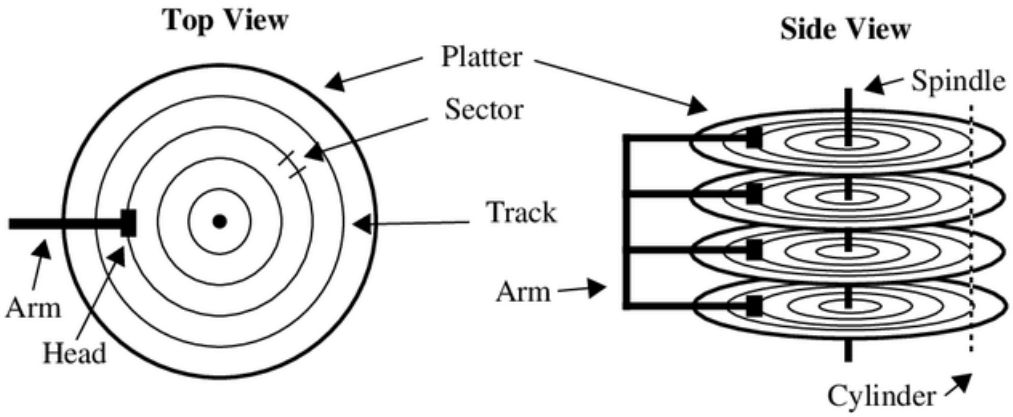
3. **Random Access:**

    - **Example:** RAM Memory
    - **Characteristics:** Provides direct access to any memory cell using its **address.** The time taken to access any location is consistent and not dependent on the location of the preceding data.

4. **Associative Access:**

    - **Example:** Cache Memory
    - **Characteristics:** Allows for quick data retrieval based on content, not specific addresses. **Data is stored associatively**, and searching is performed based on the content rather than a predefined address.

Magnetic Disk (HDD)



Structure of Magnetic Disk:

1. **Platters:** These are the circular disks inside the hard disk drive, and they are typically made of aluminum or an aluminum alloy. The platters are coated with a thin layer of magnetic material that allows data to be stored on them. Each platter surface represents a storage layer.

2. **Coating:** The magnetic coating on the platters can be applied to one or both sides. Single-sided platters store data only on one side, while double-sided platters store data on both sides. Double-sided platters increase the storage capacity of the hard disk.

3. **Tracks:** The surface of each platter is divided into concentric circles known as tracks. Each track is a separate, concentric data storage area. The tracks are used to organize and locate data on the platter.

4. **Sectors:** Each track is further divided into smaller, fixed-size units called sectors. Sectors are the smallest addressable storage units on a hard disk. The operating system and the hard disk controller work together to read and write data in terms of these sectors.

5. **Size of Sectors:** The standard size of a sector is typically 512 bytes. However, in modern hard drives, advanced format drives may use larger sector sizes, such as 4 KB (4096 bytes).

6. **Read/Write Head:** Above and below each platter, there is an actuator arm with a read/write head attached to it. The read/write heads move rapidly over the surfaces of the spinning platters to read or write data one at a time.

    - Head writes and read data sector by sector i.e. block by block, and magnetic disk is also called as **block device.**

7. **Spindle Motor:** The platters are attached to a spindle, and a spindle motor spins the platters at a high speed. The speed is measured in revolutions per minute (RPM), with common speeds being 5400, 7200, or 10,000 RPM.

8. **Cylinder:** Cylinder refers to a group of concentric tracks on the disk surfaces that are vertically aligned. Each track within a cylinder contains a specific portion of data, and by aligning them in a cylinder, the read/write head of the disk drive can access them more efficiently.

9. **Disk Controller:** The disk controller plays a central role in managing various operations within the HDD. It controls read, write, and other operations on the disk, including the movement of the head.

Seek Time:

- Seek time is the time required for the disk controller to move the head from its current position to the desired track where the data is located.

Rotational Latency:

- After the head reaches the desired track, the circular platter rotates until the head aligns with the desired sector. The time required for this rotation is known as rotational latency.

Access Time:

- Access time, in the context of a hard disk drive, is the total time it takes to locate and retrieve data.
- **Access Time = Seek Time + Rotational Latency.**
- Minimizing access time is essential for optimizing the performance of the hard disk.

RAM

RAM (Random Access Memory) is also referred to as **"Main Memory."**

- RAM is crucial for normal computer operation, storing data actively used by the CPU during program execution.
- RAM allows **direct access** to any memory cell, facilitating quick retrieval and storage of data.
- RAM is **volatile**; its contents are retained only while the power supply is active.
- Serves as a **temporary storage** space for the operating system, applications, and actively used data.
- Main memory emphasizes the central role of RAM as the **primary working memory** for the CPU.
- Offers **fast read and write speeds**, contributing to system speed and responsiveness.
- RAM is considered the **"main"** memory due to its indispensable role in program execution and data manipulation.

There are two types of RAM:

1. **DRAM (Dynamic RAM):**
    - **Cell Composition:** DRAM cells are made up of capacitors, which are used to store charge. The presence or absence of charge in the capacitor represents the binary values 0 or 1.
    - **Refresh Requirement:** Dynamic RAM requires periodic refreshing of its memory cells, as the charge stored in the capacitors tends to leak over time. This refreshing process is essential to maintain the integrity of the stored data.
    - **Density:** DRAM is typically more compact in terms of storage density compared to SRAM, making it suitable for use as main memory in computer systems.
    - **Example:** Main memory (RAM) in a computer system is a common example of DRAM. It provides the high storage capacity required for running applications and storing data during computer operations.
2. **SRAM (Static RAM):**
    - **Cell Composition:** SRAM cells are constructed using flip-flop gates. These flip-flops can store data without the need for constant refreshing, as in the case of DRAM.

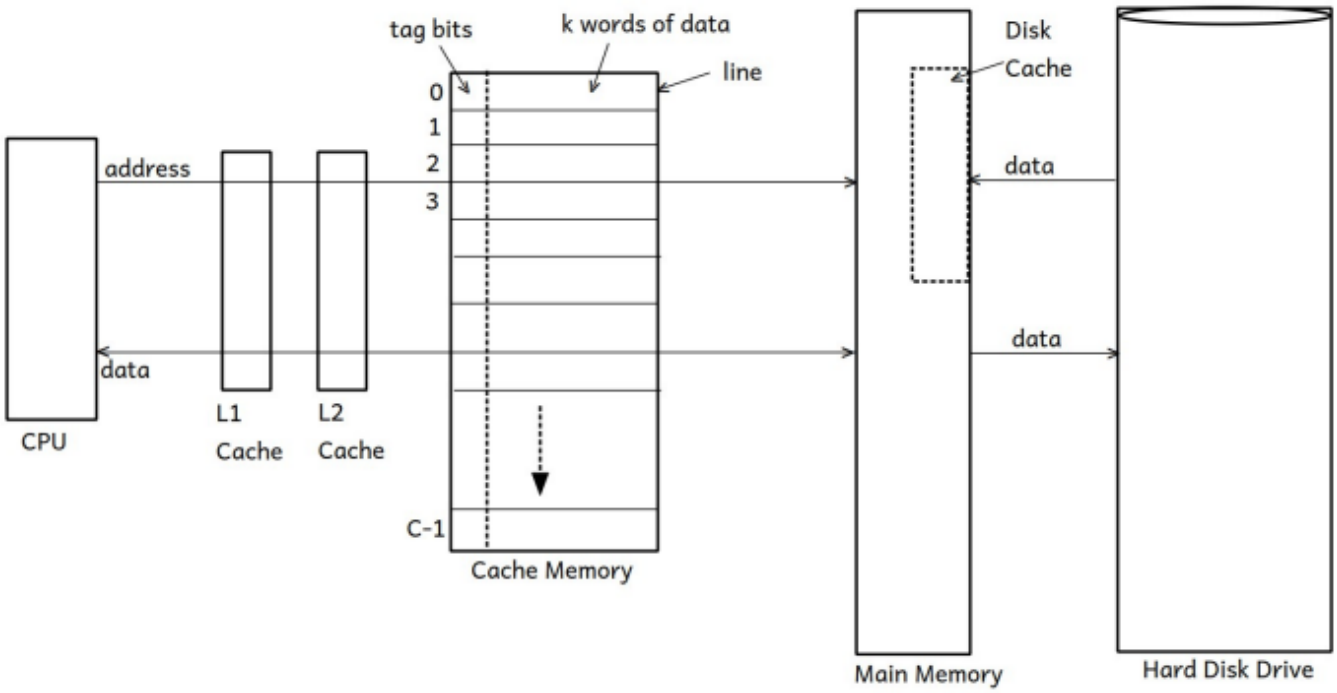
- **Stability:** SRAM is faster and more stable than DRAM, providing quicker access to stored data. However, it is also more expensive and less dense in terms of storage capacity.
- **Associative Storage:** SRAM is often used in cache memory due to its fast access times. In cache memory, recently accessed contents from the main memory are stored in an associative manner, allowing for rapid retrieval.
- **Example:** Cache memory is a common example of SRAM. It serves as a high-speed buffer between the CPU and main memory, storing frequently accessed data and instructions to reduce the time the CPU spends waiting for information.

Cache

Cache memory plays a crucial role in computer systems due to the inherent speed mismatch between the CPU and the main memory. Here's a detailed explanation of why cache memory is necessary:

1. **Speed Mismatch:**
    - CPUs can execute instructions at a much faster rate than the speed at which data can be accessed from the main memory (RAM). This is due to the different technologies and speeds at which CPUs and RAM operate.
2. **Processor Speed vs. Memory Access Time:**
    - Modern processors are designed to execute instructions at very high clock speeds. However, accessing data from the main memory takes longer compared to the speed at which the CPU can process instructions. This creates a bottleneck in performance, as the CPU has to wait for data to be fetched from the slower main memory.
3. **Role of Cache Memory:**
    - Cache memory is a small-sized, high-speed type of volatile computer memory that provides high-speed data access to the processor and stores frequently used computer programs, applications, and data.
    - The primary purpose of cache memory is to bridge the speed gap between the fast CPU and the slower main memory. By storing frequently accessed data and instructions closer to the CPU, cache memory reduces the time it takes for the CPU to fetch the required information.
4. **Levels of Cache:**
    - Modern computer systems typically have multiple levels of cache, including L1 (Level 1) and L2 (Level 2) caches. L1 cache is the smallest and fastest, located directly on the CPU chip, while L2 cache is larger and slightly slower but still faster than main memory.
5. **Improving System Performance:**
    - Cache memory significantly improves system performance by reducing the time the CPU spends waiting for data from the main memory. This results in more efficient utilization of the CPU's processing power, leading to overall better system responsiveness.

Cache memory is faster memory, which is implemented using static RAM (SRAM) technology, in which most recently accessed main memory contents can be stored in an associative manner i.e. in a key-value pairs(Tags & Data).



- Cache Memory has **C no. of lines**, whereas each line is divided into two parts, each line contains k words of data (most recently accessed main memory contents) and its main memory addresses can be kept in few tag bits.
    1. First part of a line : Few tag bits contains **main memory addresses** of k words of data in that line
    2. Second part of a line contains **k words** of data
        - "word": it is a unit of memory from system point view.
        - word length = 8 bits/16 bits/32 bits/64 bits
        - on few processor, word lenght may vary or on few process it is fixed.

Cache Hit and Cache Miss:

- A **"cache hit"** occurs when the required data or instructions are found in the cache, allowing the CPU to access them quickly without waiting for the slower main memory. Conversely, a **"cache miss"** occurs when the required data is not in the cache, leading to a retrieval from the main memory.

Even after adding cache memory between the CPU and main memory, the rate at which the CPU can execute instructions is faster than the rate at which at which data can be accessed from cache memory, and hence to reduce speed mismatch between the CPU and cache memory one or more levels of cache memory i.e. **L1 cache** & **L2 cache** can be added between them.

Disk Cache: it is purely a **software technique** used to reduce speed mismatch between the CPU and Secondary memory, in which portion of the main memory can be used as a cache memory in which most recently accessed disk contents can be kept in an *associative manner*, so whenever the CPU want to access data from hard disk drive it first gets searched into the disk cache.

IO Devices/External Device/Peripheral Devices

Deices which are connected to the motherboard externally through ports are known as peripheral devices

- Devices which are connected to the motherboard externally through ports reffered as peripheral devices or peripherals.
- An IO Devices are also reffered as an external devices.

Examples of peripheral devices include:

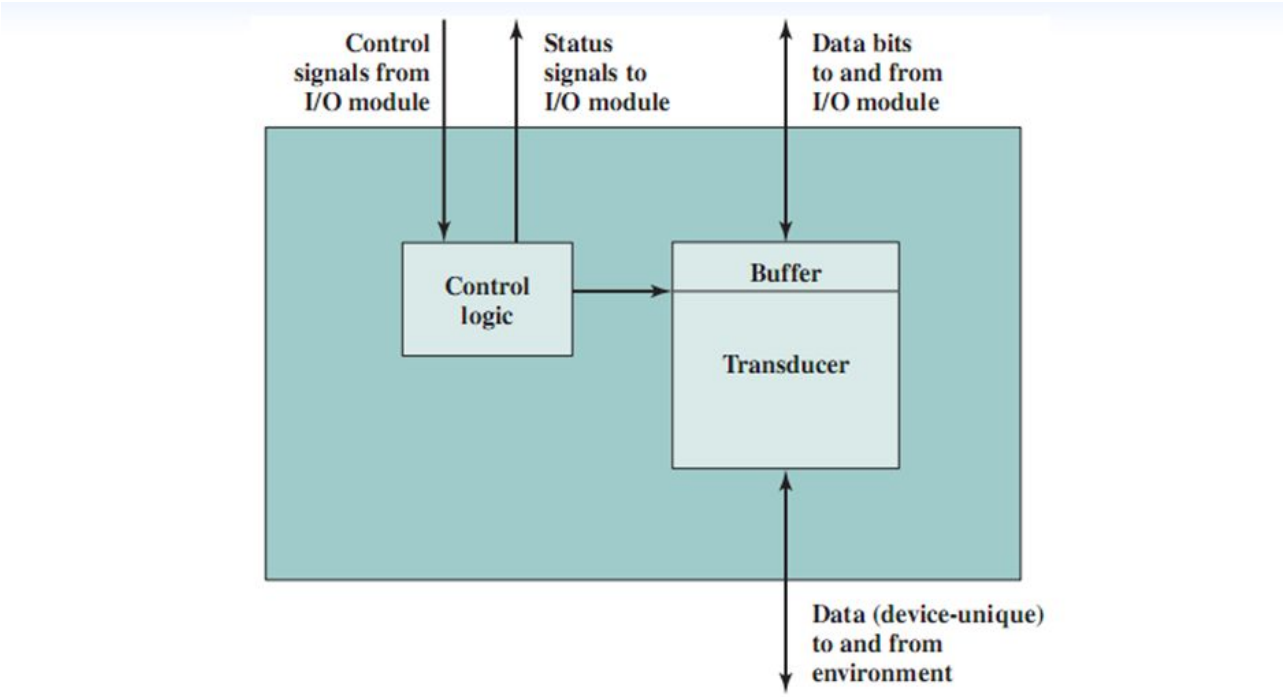
- **Input Devices:**
    - Keyboards
    - Mouse
    - Scanners
    - Webcams
    - Microphones
- **Output Devices:**
    - Monitors (Displays)
    - Printers
    - Speakers
    - Projectors
- **Storage Devices:**
    - External Hard Drives
    - USB Flash Drives
    - External SSDs
- **Communication Devices:**
    - Modems
    - Network Adapters
    - Wireless Adapters
- **Other Devices:**
    - External Card Readers
    - USB Hubs
    - External DVD/CD Drives

Peripheral devices are essential for expanding the functionality of a computer system, providing ways to interact with the computer and enhancing its capabilities. They typically connect to the motherboard through various types of ports, such as **USB, HDMI, Ethernet, audio jacks,** and more.

Structure of an External Device

External Device has three major blocks:

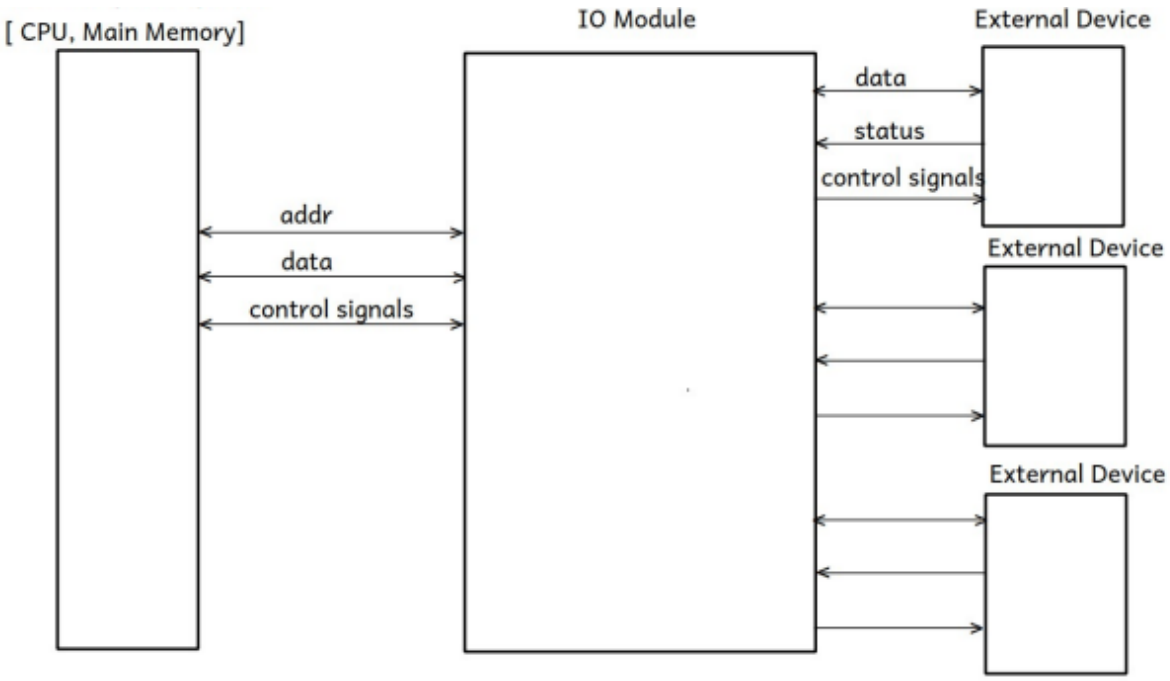
1. **Control Logic Block (Controller):**
    - It is responsible for overseeing and controlling all operations and functions of the device.
    - The controller interprets commands from the computer or user and manages the flow of data between the device and the computer.



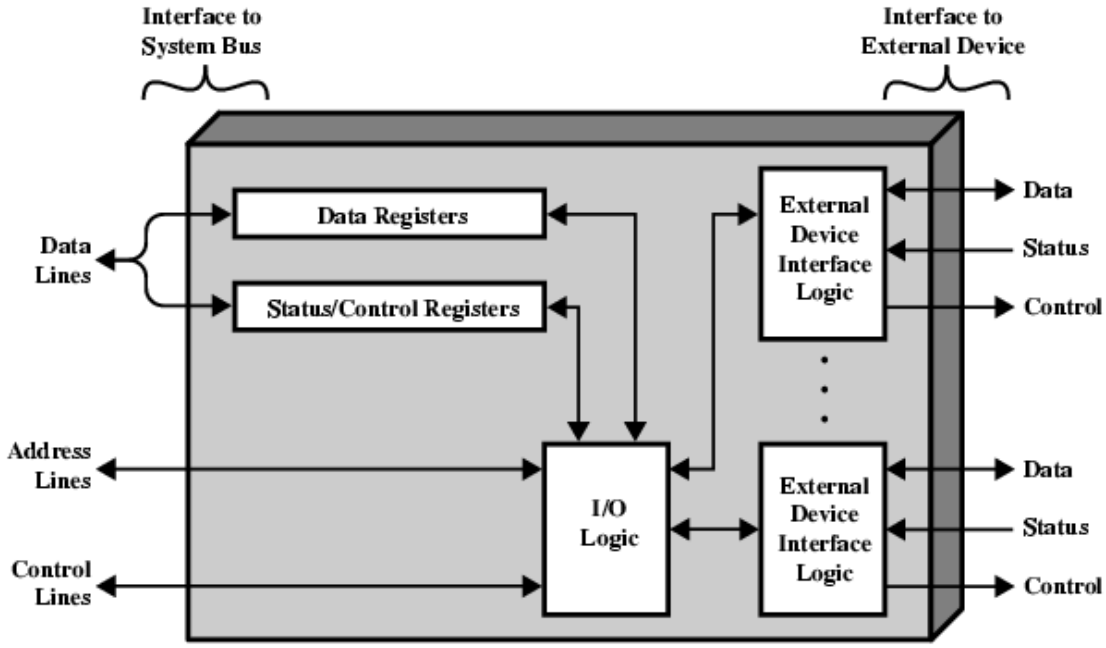
2. **Buffer:**
    - The buffer is a dedicated memory area within the external device.
    - It serves as temporary storage for data during input or output operations.
    - Buffers help to manage the flow of data between the external device and the computer, preventing potential bottlenecks.
3. **Transducer:**
    - It can convert external signals or energy into electrical signals that the device's controller can understand.
    - Conversely, the transducer can also convert electrical signals generated by the controller into the appropriate form of energy for communication with the external world.

IO Modules/IO Ports

- Core Computer system is not able to communicates directly with any external device and hence IO modules acts as an interface between core computer system and an io devices.
- IO modules contain the necessary logic and circuitry to enable communication with external devices. This logic may include protocols for data transfer, error handling mechanisms, and other functionalities required for effective communication.
- Single IO module can be used for communication between one device or with one or more devices as well.



IO modules abstract the details of device-specific communication from the core computer system. This abstraction allows the core system to remain device-independent, simplifying the integration of various external devices.



Whenever there is a transfer of data, either from the core computer system (via the bus) to an IO device or vice versa, it is referred to as an IO operation.

There are three IO techniques:

1. Program driven IO
2. Interrupt IO
3. DMA i.e. Direct Memory Access

Your statement is correct. These three IO (Input/Output) techniques are common methods used in computer systems to manage the transfer of data between the central processing unit (CPU) and external devices:

1. **Programmed IO (Program Driven IO):**
    - In programmed IO, the CPU manages data transfer to and from the IO devices through its own instructions.
    - The CPU is actively involved in controlling each data transfer operation.
    - **Advantage:** Programmed IO is simple
    - **Disadvantage:** The CPU has to wait for the IO operation to complete before moving on to other tasks.
2. **Interrupt Driven IO:**
    - In interrupt-driven IO, the CPU is notified (interrupted) when an IO operation is complete or requires attention.
    - The IO device generates an interrupt signal to the CPU, indicating that it needs attention.
    - **Advantage:** This allows the CPU to continue with other tasks while waiting for the IO operation to complete.
    - **Disadvantages:** When there is a data transfer between main memory & secondary memory unneccessary involvement of the CPU is there.
3. **DMA (Direct Memory Access):**
    - DMA is a technique where a separate DMA controller is used to manage data transfers between IO devices and memory without involving the CPU.
    - The DMA controller takes control of the system bus temporarily, allowing direct communication between the IO device and the memory.
    - This frees up the CPU to perform other tasks while the data transfer occurs in the background.
    - **Advantage:** DMA is efficient for large data transfers and reduces CPU involvement in IO operations.
    - **Disadvantage:** Increased system complexity, potential for bus contention, and limited support in some devices, impacting overall system performance and flexibility. **e.g. 8237 DMA controller.**

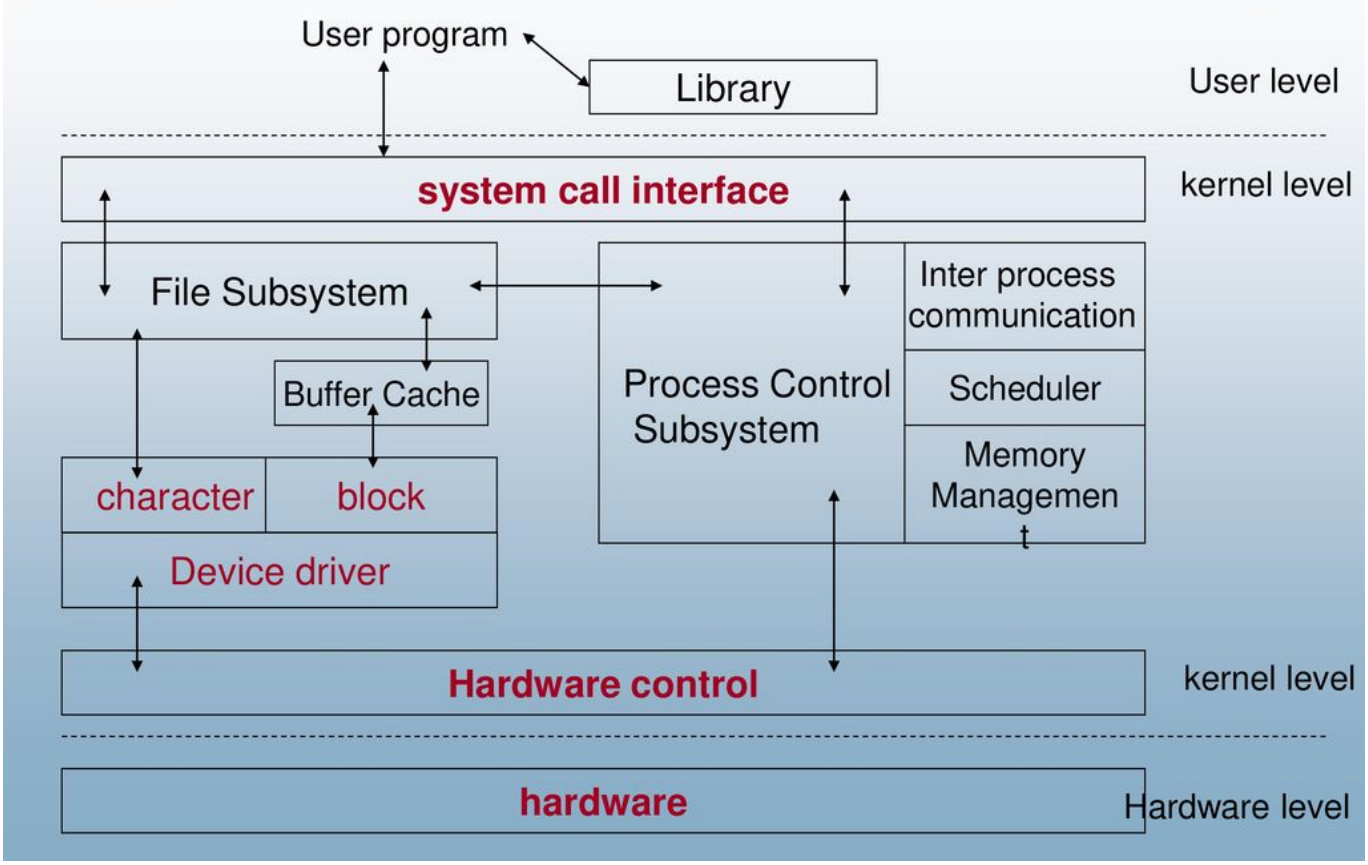
Types of Operating Systems

1. **Single-User, Single-Tasking:**
    - Designed for individual users and can handle only one task at a time.
    - Examples: MS-DOS.
2. **Single-User, Multi-Tasking:**
    - Supports multiple applications running concurrently for a single user.
    - Examples: Windows, macOS.
3. **Multi-User:**
    - Supports multiple users accessing the system simultaneously.
    - Examples: Unix, Linux.
4. **Real-Time Operating System (RTOS):**
    - Designed for real-time applications with strict timing requirements.

- Examples: VxWorks, QNX.

UNIX OS

- UNIX was developed at AT&T Bell Labs in US, in the decade of **1970's** by Ken Thompson, Denies Ritchie and team.
- It was first run on a machine **DEC-PDP-7** (Digital Equipment Corporation – Programmable Data Processing - 7).
- UNIX is the **first multi-user, multi-programming & multi-tasking** operating system.
- UNIX was specially designed for developers by developers, emphasizing a powerful and flexible environment for programming.
- System architecture design of UNIX is followed by all modern OS's like Windows, Linux, MAC OS, Android etc..., and hence UNIX is referred as mother of all modern operating systems.



- Kernel acts as an interface between programs and hardware.
- Operating System has subsystems like **System Call Interface, File subsystem, Process Control Subsystem(IPC, Memory Management & CPU Scheduling), Device Driver, Hardware Control/Hardware Abstraction Layer.**
- There are two major subsystems:

1. **Process Control Subsystem:** It manages processes, handles IPC, and oversees memory management and CPU scheduling.
2. **File Subsystem:** It deals with file-related operations, treating everything that can be stored as a file in the UNIX philosophy.

- In UNIX, whatever is that can be stored is considered as a **file** and whatever is active is reffered as a **process.**
- File has space & Process has life.

UNIX File and Process Model:

In UNIX, the philosophy is that **"everything is a file."** This means that not only regular files but also devices, directories, sockets, and other entities are treated as files. The concept that **"whatever is active is referred to as a process"** aligns with the idea that processes are active entities in the system with a specific life cycle.

In UNIX, devices are catagorises into two catagories:

1. **Character Devices:**
    - Character devices transfer data character by character.
    - These devices use character special device files for interaction.
    - Examples of character devices include keyboards, mice, printers, and monitors.
    - They are often used for devices where the data is transmitted as a stream of characters without being grouped into blocks.
2. **Block Devices:**
    - Block devices transfer data in fixed-size blocks or chunks.
    - These devices use block special device files for interaction.
    - Examples of block devices include storage devices such as hard drives, solid-state drives, and USB drives.
    - Block devices are well-suited for devices where data is managed in blocks, providing efficiency in handling large amounts of data.

Device Driver:

- A device driver is a software component or set of programs that facilitates communication between a specific hardware device and the computer's operating system.
- Device drivers act as intermediaries, translating high-level operating system commands into commands that the hardware device can understand and vice versa.
- They allow the operating system to control and utilize various hardware components, such as printers, graphics cards, network adapters, etc.

Hardware Control Layer/Block:

- The hardware control layer (or hardware abstraction layer) is a component that interfaces with the control logic block or controller of a hardware device.
- It acts as a bridge between the higher-level software (such as the operating system or device driver) and the low-level control logic embedded in the hardware.
- The hardware control layer abstracts the details of hardware-specific operations, providing a standardized interface for software components.

System Calls

System calls are functions defined in programming languages like C, C++, and Assembly, serving as an interface for user programs to access services provided by the kernel.

Programmers can use kernel services directly through system calls or indirectly through sets of library functions provided by the programming language.

Categories of System Calls:

1. **Process Control System Calls:** Manage processes, e.g., `fork()`, `_exit()`, `wait()`.
2. **File Operations System Calls:** Handle file-related operations, e.g., `open()`, `read()`, `write()`, `close()`.
3. **Device Control System Calls:** Control and interact with devices, e.g., `open()`, `read()`, `write()`, `ioctl()`.
4. **Accounting Information System Calls:** Retrieve accounting information, e.g., `getpid()`, `getppid()`, `stat()`.
5. **Protection & Security System Calls:** Manage security aspects, e.g., `chmod()`, `chown()`.
6. **Inter-Process Communication System Calls:** Facilitate communication between processes, e.g., `pipe()`, `signal()`, `msgget()`.

- **Purpose of System Calls:**
    - System calls provide a standardized way for user programs to interact with the underlying operating system kernel.
    - They enable essential functionalities related to process control, file handling, device management, security, and inter-process communication.
- In *UNIX 64* system calls are there.
- In *Linux more than 300* system calls are there
- In *Windows more than 3000* system calls are there
- **System Call Mechanism:**
    - When a program needs to request a service from the operating system, it invokes a system call.
    - Upon a system call, the CPU switches from user mode (user-defined code) to kernel mode (system-defined code).
    - System calls are often referred to as **"software interrupts"** or **"traps"** because they interrupt the normal flow of the program to transfer control to the kernel.

```c
//user defined code
#include<stdio.h>
//entry point function
int main(void)
{
    int n1, n2, res;
    printf("enter the values of n1 & n2: ");//write() - system defined
code
    scanf("%d %d", &n1, &n2);//read() - system defined code
    res = n1 + n2;
    printf("res: %d\n", res);//write() - system defined code
```

```
    return 0;//successful termination
}
```

Dual Mode Operation

1. System Mode (Kernel Mode):

- When the CPU executes instructions designated as system-defined code, the system operates in system mode.
- Kernel mode provides unrestricted access to system resources and privileged instructions.
- System mode is also known as kernel mode, monitor mode, supervisor mode, or privileged mode.

2. User Mode (Non-privileged Mode):

- When the CPU executes instructions from user-defined code, the system operates in user mode.
- User mode restricts access to certain privileged instructions and resources to ensure security and stability.
- User mode is also referred to as non-privileged mode.

Throughout the execution of programs, the CPU dynamically switches between kernel mode and user mode based on the nature of the code being executed. This Siwtching is also termed as **Dual Mode Operation.**

Your explanation further delves into the mechanism of differentiating between user mode and kernel mode using a mode bit. Here's a summary:

- **Mode Bit:**
  - A single bit, often referred to as the "mode bit," is maintained by the operating system onto the CPU.
  - The mode bit helps the CPU identify whether the currently executing instruction is part of system-defined code or user-defined code.
    - In kernel mode, the value of mode bit is set to **0.**
    - In user mode, the value of mode bit is set to **1.**

Functions of OS

1. Process Management: A process is an instance of a program in execution. It consists of an address space, code, data, and system resources.

- **Process Control Block (PCB):**
  - PCB is a data structure that stores information about a process, including its state, program counter, registers, etc.
- **States of a Process:**
  - Processes can be in states such as New, Ready, Running, Blocked, or Terminated.
- **CPU Scheduling:**
  - Allocating CPU time to processes using scheduling algorithms like FCFS, SJF, Round Robin, etc.
- **Inter-Process Communication (IPC):**
  - **Shared Memory Model:**
    - Processes communicate by sharing a portion of memory.

    - **Message Passing Model:**
        - Processes exchange messages through an IPC mechanism.

2. Memory Management:

- **Swapping:**
    - Moving processes between main memory and disk to support multitasking.
- **Memory Allocation Methods:**
    - Contiguous Allocation, Paging, Segmentation.
- **Fragmentation:**
    - Internal Fragmentation (within allocated blocks) and External Fragmentation (free memory scattered).
- **Segmentation:**
    - Dividing a program into logical segments for better organization.

3. Hardware Abstraction: Hardware abstraction provides a layer between application software and hardware, allowing software to run on various hardware architectures.

- **Abstraction Layers:**
    - Interface layers that hide hardware complexity (e.g., HAL - Hardware Abstraction Layer).
- **Device Drivers:**
    - Software components that facilitate communication between the operating system and hardware devices.

4. CPU Scheduling:

- **Purpose:**
    - Efficiently allocate CPU time among competing processes.
- **Scheduling Algorithms:**
    - First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin, Priority Scheduling, etc.
- **Context Switching:**
    - Switching between different processes, involving saving and restoring context.

5. File & IO Management:

- **File Management:**
    - A file is a collection of related information. File systems organize and store files.
- **Disk Space Allocation:**
    - Contiguous Allocation, Linked Allocation, Indexed Allocation.
- **IO Techniques:**
    - Programmed IO, Interrupt-driven IO, Direct Memory Access (DMA).
- **Disk Scheduling Algorithms:**
    - Strategies to optimize disk access (e.g., FCFS, SSTF, SCAN).

6. Protection & Security:

- **User Authentication:**
    - Verify the identity of users before granting access.
- **Authorization:**
    - Assign specific privileges to users based on roles.

- **Encryption:**
  - Protect data by converting it into a code that can only be decoded by authorized parties.

7. User Interfacing:

- **Graphical User Interface (GUI):**
  - Visual representation of system functions for user interaction.
- **Command-Line Interface (CLI):**
  - Users interact with the system by typing commands.
- **Shell:**
  - Interface between users and the operating system.

8. Networking:

- **Network Protocols:**
  - Rules for communication between devices in a network (e.g., TCP/IP).
- **Device Communication:**
  - Managing communication between devices in a network.
- **Distributed Systems:**
  - Operating systems that run on multiple machines and communicate over a network.

Fundamental Features of an OS

1. **Multi-Programming:** A system in which more than one process can be submitted or more than one program can be started for execution at a time.
   - *Degree of Multi-Programming:* Refers to the number of programs that can be submitted into the system simultaneously.
2. **Multi-Tasking:** A system in which it appears that the CPU can execute more than one program simultaneously or concurrently.
   - *Time-Sharing:* Involves sharing the CPU time among all running programs to give the illusion of parallel execution.
3. **Multi-Threading:**
   - Thread is the smallest execution unit of a process.
   - Thread is the smallest indivisible part of a process.
   - Thread is a lightweight process.

   A system in which it appears that the CPU can execute more than one thread simultaneously. Threads can belong to the same process or different processes.
4. **Multi-Processor:** A system that can run on a machine in which more than one CPUs are connected in a closed circuit.
5. **Multi-User:** A system in which multiple users can log in and interact with the system concurrently.

Process Management

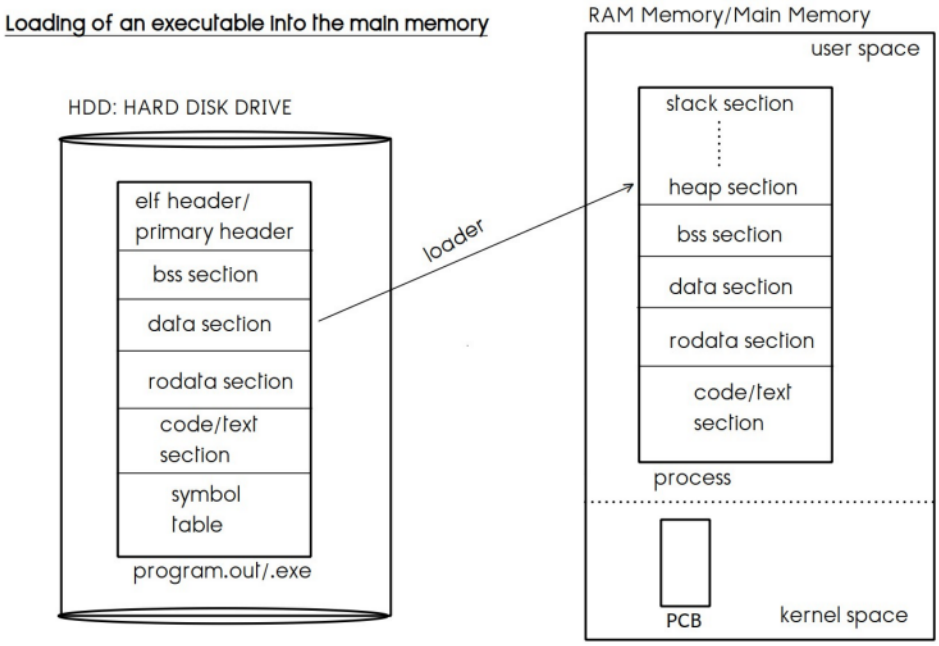
- When we say an OS does process management it means an OS is responsible for:

- Process Creation
- To provide environment for an execution of a process
- Resource Allocation
- Scheduling
- Resources Management
- Inter Process Comminication
- Process Coordination
- Terminating the Process.

Program:

- **User View:** From a user's perspective, a program is indeed a set of instructions given to the computer to perform a specific task. Users interact with programs to accomplish various activities, such as word processing, web browsing, or playing games.
- **System View:** From the system's perspective, a program is an executable file that contains machine-readable instructions for the computer's central processing unit (CPU). The program is typically divided into several sections, each serving a specific purpose:

    1. **Executable (exe) Header:** This part of the program contains information that the operating system needs to execute the program properly. It includes details such as the entry point of the program and the size of various sections.
    2. **BSS Section:** This section contains *uninitialized global and static variables.* The name "BSS" stands for "Block Started by Symbol."
    3. **Data Section:** This section holds *initialized global and static variables.* It stores data that has a predefined value before the program starts running.
    4. **Rodata Section:** The term "rodata" stands for "read-only data." This section contains *constant or read-only data* that is used by the program but cannot be modified during its execution.
    5. **Code Section:** This is the core of the program, containing the *actual executable instructions* that the CPU will execute.
    6. **Symbol Table:** The symbol table keeps track of all the symbols used in the program, including *variables, functions, and other entities.* It helps the linker and debugger in the compilation and debugging processes.



Process:

- **User View:**

  1. Running instance of a program is reffered as a process. It represents the active state of a program while it's running.
  2. The running instance of a program is also called a process. This emphasizes the dynamic and active nature of the program at runtime.
  3. When a program is loaded into the main memory for execution, it is considered a process. This loading involves allocating memory and resources necessary for the program to run.

- **System View:**

  1. A process is essentially a file loaded into the main memory. This file consists of various sections, including:
     - **BSS Section:** Uninitialized global and static variables.
     - **Rodata Section:** Read-only data, constant throughout the program's execution.
     - **Code Section:** Contains the executable instructions.
     - **Stack Section:** Holds function activation records and local variables. It's used for managing function calls and returns.
     - **Heap Section:** Dynamically allocated memory during the program's execution for data structures like arrays and objects.

Kernel Space and User Space:

- The core program of an operating system (kernel) runs continuously in the main memory.
- The main memory is logically divided into two parts:
  - **Kernel Space:** Occupied by the operating system's core (kernel).
  - **User Space:** Allocated for user programs.
  - User programs are loaded into the user space, the part of memory reserved for application execution.

Process Control Block (PCB):

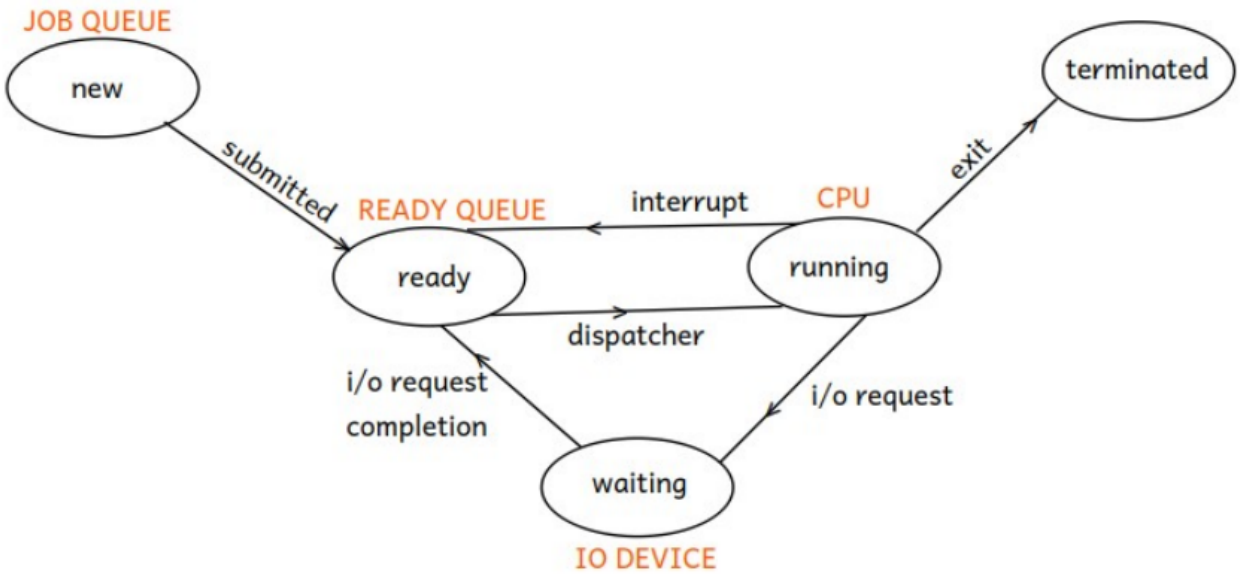
- Upon the execution or submission of a process, the operating system creates a structure in the main memory inside the kernel space to store information needed to control the execution of that process. This structure is called the **Process Control Block (PCB)** or **Process Descriptor.**
- Each process has its own PCB, and the PCB remains in the main memory throughout the execution of the program.
- The PCB is destroyed when the process exits.

Contents of PCB:

- The PCB contains various pieces of information necessary for managing and controlling the process. This information includes:
  - **PID (Process ID):** A unique identifier for the process.
  - **PPID (Parent Process ID):** ID of the process that created the current process.
  - **PC (Program Counter):** Points to the address of the next instruction to be executed.
  - **CPU Scheduling Information:** Details about the process's scheduling, such as priority.
  - **Memory Management Information:** Information about the memory allocated to the process.
  - **Resource Allocation Information:** Details about resources allocated to the process.
  - **Execution Context:** Information about the register values and other CPU-related state.

Process States

1. **New State:**
   - The process is in the new state upon submission, or when a Process Control Block (PCB) for a process is created in the main memory.
   - At this stage, the process is just being initialized, and resources are being allocated.
2. **Ready State:**
   - After creation, if the process is in the main memory and waiting for CPU time, it is in the ready state.
   - Processes in the ready state are prepared to execute but are waiting for their turn to be scheduled on the CPU.
3. **Running State:**
   - When the CPU is actively executing a process, the process is in the running state.
   - During this state, the process is actively performing its tasks and executing instructions.
4. **Waiting State:**
   - If a process needs to wait for some external event, such as input/output (I/O) operations or the availability of a resource, it enters the waiting state.
   - Processes in the waiting state are temporarily not using the CPU and are waiting for a specific event to occur.
5. **Terminated State:**
   - When a process completes its execution or is explicitly terminated, it enters the terminated state.
   - In this state, the process is considered finished, and its PCB is typically removed or deallocated from the main memory.



PROCESS STATE DIAGRAM

Important Points:

- A process can be in only one state at any given time.
- Transitions between states occur based on events and the execution of the process.
- The New state is the initial state, and Terminated is the final state.

Kernel Data Structures for Process Management

These kernel data structures are essential for the operating system to efficiently manage and schedule processes.

1. **Job Queue:**
    - *Definition:* Contains a list of PCBs (Process Control Blocks) for all submitted processes.
    - The job queue represents all processes that have been submitted but may not yet be actively running.
    - *Purpose:* It serves as a repository for all processes that have been submitted to the operating system. Processes in the job queue may not be actively running yet.
2. **Ready Queue:**
    - *Definition:* Contains a list of PCBs for processes that are in the main memory and waiting for CPU time.
    - The ready queue contains processes that are prepared to execute, waiting for CPU time.
    - *Purpose:* Processes in the ready queue are prepared to execute but are waiting for their turn to be scheduled on the CPU.
3. **Waiting Queue:**
    - *Definition:* Contains a list of PCBs for processes that are currently in a waiting state, typically waiting for a specific device.
    - The waiting queue is specific to processes waiting for a particular device or event.

- *Purpose:* Processes in the waiting queue are not actively using the CPU and are waiting for a particular event, such as I/O completion.

Scheduling

1. **Job Scheduler/Long Term Scheduler:**
    - *Definition:* A system program responsible for selecting and scheduling jobs or processes from the job queue to load them onto the ready queue.
    - *Purpose:* Determines which processes from the job queue should be brought into the system for execution. This scheduler is responsible for managing the overall job mix and system load.
2. **CPU Scheduler/Short Term Scheduler:**
    - *Definition:* A system program that selects and schedules jobs or processes from the ready queue to load them onto the CPU for execution.
    - *Purpose:* Manages the execution of processes in the main memory, deciding which process should be given CPU time. This scheduler operates more frequently than the long-term scheduler.

- The job scheduler and CPU scheduler work together to manage the loading and execution of processes in the system.

Dispatcher:

- *Definition:* A system program responsible for actual loading a process onto the CPU, which has been scheduled by the CPU scheduler.
- *Dispatcher Latency:* The time required for the dispatcher to stop the execution of one process and start the execution of another process. Dispatcher latency contributes to the overall context-switching time.

Context Switch

Context switch refers to the process of switching the CPU from the execution context of one process to the execution context of another process.

Context switching is essential for multitasking and allows the operating system to efficiently manage and execute multiple processes.

Components of Context Switch:

- **State-Save:** Saving the execution context of the suspended process. This information is typically stored in the Process Control Block (PCB) of the process.
- **State-Restore:** Loading the execution context of the process scheduled by the CPU scheduler onto the CPU. This is performed by the dispatcher, which copies the relevant information from the PCB to the CPU registers.

Handling Process Priorities:

- When a high-priority process arrives in the ready queue, it can preempt a low-priority process.
- The low-priority process is suspended through an interrupt, and the CPU's control is allocated to the high-priority process.

- Once the high-priority process completes its execution, the low-priority process can be resumed from the point at which it was suspended.

CPU Scheduling

- **CPU Scheduler**: This is a component of the operating system that decides which process should be executed by the CPU at any given time. It gets called in four scenarios:

    1. When a process is **Running** and then **Terminated**.
    2. When a process is **Running** and then goes into a **Waiting** state.
    3. When a process is **Running** and then becomes **Ready** for execution.
    4. When a process is **Waiting** and then becomes **Ready** for execution.

- There are two types of CPU scheduling:

    1. **Non-preemptive scheduling**: In this type, a process voluntarily releases control of the CPU. For example, in above case 1 & case 2
    2. **Preemptive scheduling**: In this type, control of the CPU is taken away forcefully from a process. For example, in above case 3 & case 4

Following algorithms are used for CPU Scheduling:

1. FCFS (First Come First Served) CPU Scheduling
2. SJF (Shortest Job First) CPU Scheduling
3. Priority CPU Scheduling
4. Round Robin CPU Scheduling
5. Multilevel Queue CPU Scheduling
6. Multilevel Feedback Queue CPU Scheduling

As multiple algorithms are there for CPU scheduling and hence there are certain criterias which algorithm is best suited in which situation and which one efficient are reffered as CPU scheduling criterias.

CPU Scheduling Criterias

1. **CPU Utilization:**
    - CPU utilization is the percentage of time the CPU is actively executing processes.
    - One need to select such an algorithm in which utilization of the CPU must be as **max** as possible.
2. **Throughput:**
    - Throughput is the total amount of work done per unit of time.
    - One need to select such an algorithm in which throughput must be as **max** as possible.
3. **Waiting Time:**
    - Waiting time is the total time a process spends waiting in the ready queue before getting CPU time.
    - One need to select such an algorithm in which waiting time must be as **min** as possible.

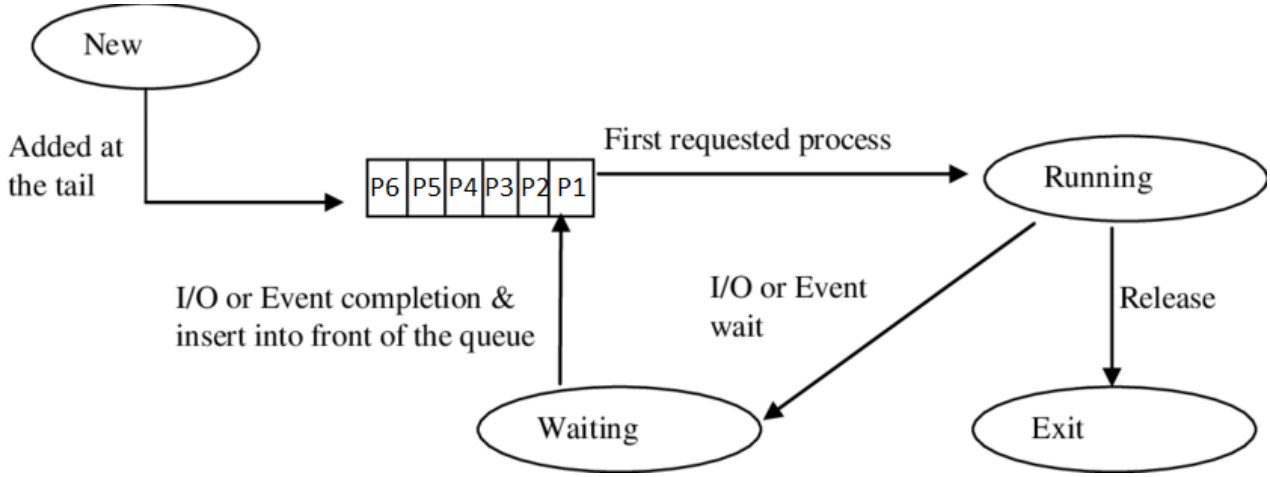
4. **Response Time:**
    - Response time is the time taken for a process to receive the first response from the CPU.
    - One need to select such an algorithm in which response time must be as **min** as possible.
5. **Turn-Around-Time:**
    - Turn-around-time is the total time taken for a process to complete its execution from the time of submission.
    - One need to select such an algorithm in which turn-around-time must be as min as possible.
    - **Execution time** is the total time a process spends on the CPU to complete its execution.
    - **CPU Burst Time** is the number of CPU cycles required for a process to complete its execution.
    - **Turn-around-time = Waiting Time + Execution Time.**

Algorithms for CPU Scheduling

- "Gant Chart": Bar chart presentation of CPU allocation for processes in terms of CPU cycle numbers.

1. First Come First Served (FCFS)

- In FCFS, the process that arrives first in the ready queue is given control of the CPU first. It operates on the principle of serving processes in the order of their arrival.
- This algorithm is simple to implement and can be managed using a First-In-First-Out (FIFO) queue, reflecting the order in which processes enter the ready queue.
- **FCFS is a non-preemptive** scheduling algorithm, meaning that once a process starts its execution, it continues until it completes without interruption.



Convoy Effect:

- The convoy effect is a drawback of the FCFS algorithm. It occurs when longer processes arrive before shorter processes.
- Due to the non-preemptive nature of FCFS, shorter processes must wait for the completion of longer processes before they can execute. This leads to an increase in the average waiting time for shorter processes.
- The increase in average waiting time, in turn, contributes to higher average turn-around-time and can negatively impact the overall system performance.

Jobs/ Processes	CPU Burst Time
P1	24
P2	3
P3	3

P1	P2	P3
0	24	27	30

Submission Time of P1, P2 & P3 = 0 ms
Order of an arrivals: P1, P2, P3

W.T. of P1 = 0 ms
W.T. of P2 = 24 ms
W.T. of P3 = 27 ms
A.W.T = (0+24+27)/3 = 51/3 = 17 ms

Waiting Time

R.T. of P1 = 0 ms
R.T. of P2 = 24 ms
R.T. of P3 = 27 ms
A.R.T = (0+24+27)/3 = 51/3 = 17 ms

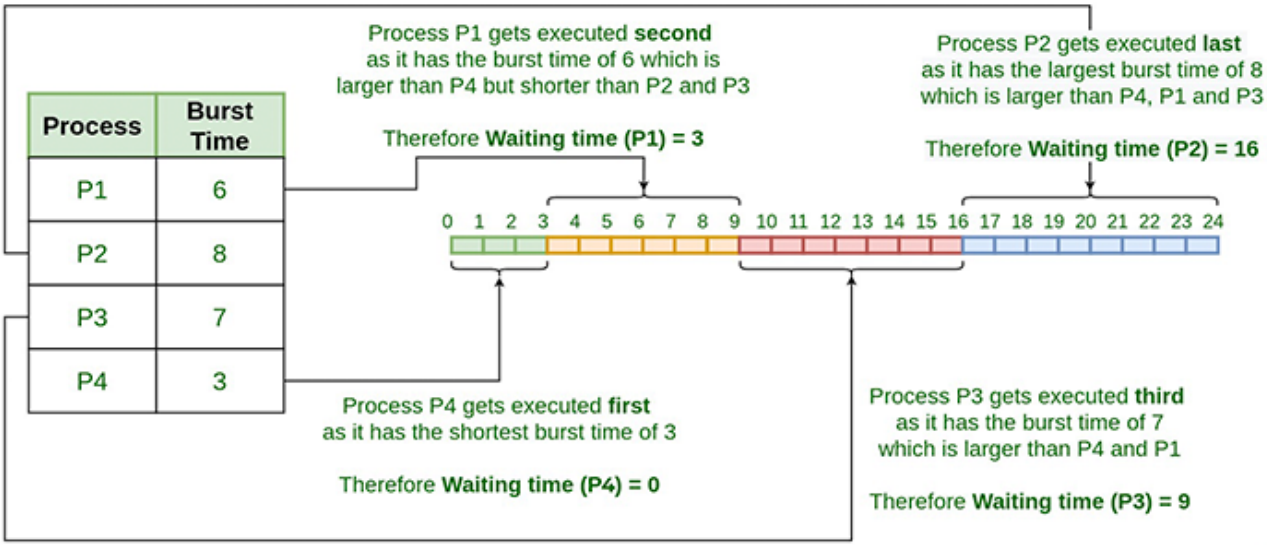
Response Time

T.A.T of P1 = 0 + 24 = 24 ms
T.A.T. of P2 = 24 +3 = 27 ms
T.A.T. of P3 = 27 + 3 = 30 ms
A.T.A.T = (24+27+30)/3 = 81/3 = 27 ms

Turn Around Time

2. Shortest Job First (SJF)

- This algorithm assigns the CPU to the process with the smallest CPU burst time first. The aim is to minimize waiting time.
- Under non-preemptive SJF, algorithm fails as the submission time of all processes are not same, and hence it can be implemented as preemptive as well.



- **Non-preemptive SJF**: In this mode, once a process starts execution, it runs to completion. It can't be interrupted. This mode can fail if the arrival times of the processes are not the same. This is also known as **Shortest-Next-Time-First (SNTF).**
- **Preemptive SJF**: In this mode, if a new process arrives with a CPU burst time less than the remaining time of the current executing process, the CPU is taken from the current process and given to the new

process. This is also known as **Shortest-Remaining-Time-First (SRTF).**

Jobs/ Processes	CPU Burst Time
P1	24
P2	3
P3	3

P2	P3	P1

0 3 6 30

Submission Time of P1, P2 & P3 = 0 ms
Order of an arrivals: P1, P2, P3

W.T. of P1 = 6 ms
W.T. of P2 = 0 ms
W.T. of P3 = 3 ms
A.W.T = (6+0+3)/3 = 9/3 = 3 ms

R.T. of P1 = 6 ms
R.T. of P2 = 0 ms
R.T. of P3 = 3 ms
A.R.T = (6+0+3)/3 = 9/3 = 3 ms

T.A.T of P1 = 6 + 24 = 30 ms
T.A.T. of P2 = 0 + 3 = 3 ms
T.A.T. of P3 = 3 + 3 = 6 ms
A.T.A.T = (30+3+6)/3 = 39/3 = 13 ms

3. Priority Scheduling

- Each process is assigned a priority value in its PCB. Lower numerical values typically indicate higher priority. The process with the highest priority is given control of the CPU first.
- **Priority Scheduling is a preemptive** scheduling algorithm, meaning that the operating system can interrupt a currently running process to start or resume another process with a higher priority.

Process	Burst Time	Priority
P1	10	2
P2	5	0
P3	8	1

P1	P3	P2

0 10 18 23

Blocking and Starvation:

- If a process with very low priority is continually moved to the back of the ready queue, it may never get a chance to execute. This situation is called **"starvation"** or **"indefinite blocking,"** where a low-priority process is blocked indefinitely from accessing the CPU.

Aging Technique:

- To prevent starvation, an aging technique is introduced. The OS gradually increases the priority of processes that have been waiting in the ready queue for an extended period. This ensures that even low-priority processes eventually get a chance to execute, preventing indefinite blocking.
- The aging process involves incrementing the priority of a blocked process after a fixed time interval. As a result, the priority of a blocked process gradually increases over time, making it more likely to be selected for execution.

4. Round Robin

- This algorithm allocates a fixed **time slice** (or **time quantum**) to each process in the system. The CPU executes any process maximum for dedicated time slice. Once the given time slice is finished process gets suspended and control of the CPU gets allocated to the next process for maximum the decided time slice and so on each process gets control of the CPU in a round robin manner.

Round Robin

time quantum/time slice it should not be so less or should not more
time quantum = 4 ms

Jobs/ Processes	CPU Burst Time
P1	12,8,4
P2	5,1,#
P3	3,#

P1	P2	P3	P1	P2	P1
0	4	8	11	15	16	20

R.T. of P1 = 0 ms
R.T. of P2 = 4 ms
R.T. of P3 = 8 ms
A.R.T. = (0+4+8)/2 = 12/3 = 4 ms

there is no starvation in round robin scheduling

- **RR is Preemptive Algorithm.** If a process completes its execution before its allocated time slice is up, then control of the CPU is immediately passed to the next process. This ensures maximum utilization of the CPU.
- The algorithm aims to ensure a minimum response time for all processes. By giving each process a chance to execute in a round-robin manner, it prevents any single process from monopolizing the CPU.

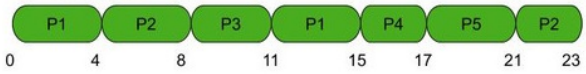
TIME SLICE = 4

PROCESS	ARRIVAL TIME	BURST TIME	
		TOTAL	REMAINING
P1	0	8	8
P2	1	6	6
P3	3	3	3
P4	5	2	2
P5	6	4	4

READY QUEUE:

P1	P2	P3	P1	P4	P5	P2

GANTT CHART:



0 4 8 11 15 17 21 23

Inter Process Communication

IPC is a mechanism that allows processes to communicate and share data with each other. The processes can be categorized into two types:

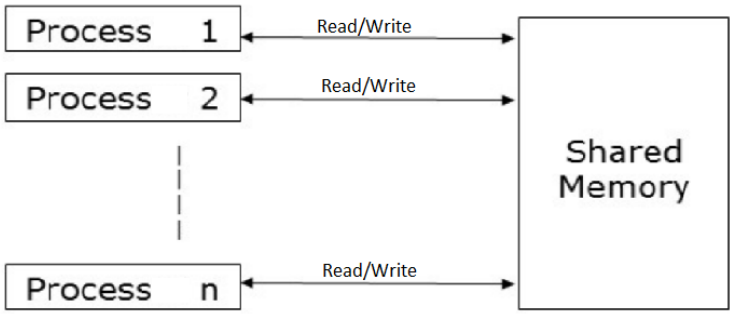
1. **Independent Processes**: These are processes that do not share data with any other process. They do not affect or get affected by the execution of other processes. Essentially, they operate in isolation.

2. **Cooperative Processes**: These are processes that share data with other processes. They can affect or get affected by the execution of other processes. This interaction allows them to work together to complete tasks.

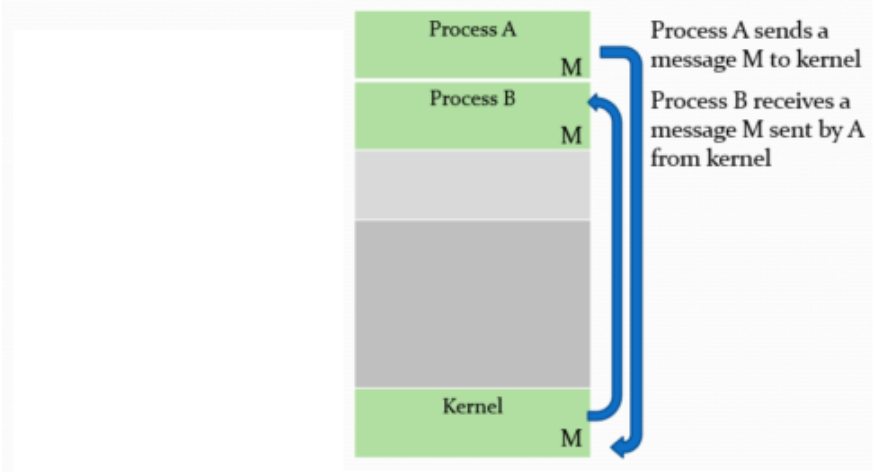
IPC can occur only between cooperative processes, i.e., processes that share data with each other. There are two primary techniques for IPC:

1. **Shared Memory Model**: In this model, processes communicate by reading from and writing to a **shared memory region.** This region is provided by the operating system upon request by the processes that want to communicate. The advantage of this model is that it's faster because processes can directly access the shared memory.

- By making **shmget()** system call shared memory is requested



1. **Message Passing Model**: In this model, processes communicate by sending and receiving messages. This model is more structured as it allows for direct communication between processes, but it might be slower than the Shared Memory Model because messages need to be passed around.

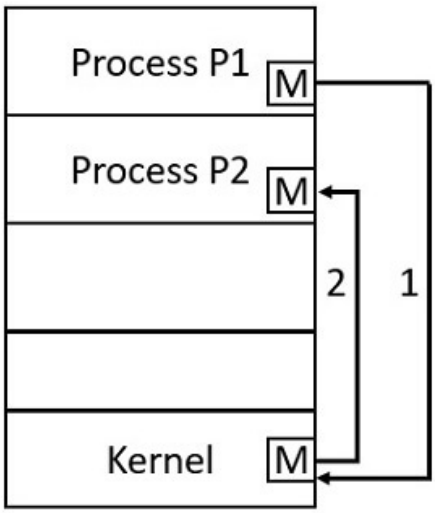


- Shared Memory Model is faster than Message Passing Model.

Message Passing Model can be further divided in 4 types

1. **Pipe:**
    - By using pipe mechanism one process can send message to another process, vice-versa is not possible and hence it is an **unidirectional communication** technique.

- By using Pipe only processes which are running on the same system can communicates.

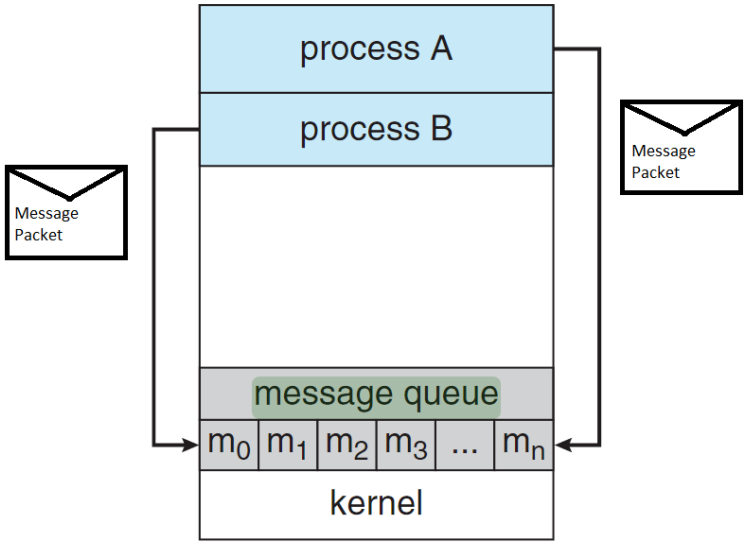


- There are two types of pipes:

    1. **Unnamed Pipe:** By using unnamed pipe, only related processes can communicates. e.g. pipe (|) command.
    2. **Named Pipe:** By using named pipe non-related processes also can communicates. e.g. pipe() system call.

2. **Message Queue:**

- By using message queue technique, processes can communicates by means of sending as well as recieving message packets to each other, and hence it is a **bidirectional communication.**
- **Message Packet = Message Header(Information about the message) + Actual Message.**
- Internally an OS maintains message queue in which message packets sent by one process are submitted and can be sent to reciever process and vice-versa.



3. **Signals:**

- Processes communicates by means of passing signals as well.
- One process can send signal to another process.
- An OS send signal to any process but any process cannot send signal to an OS.

- When we shutdown the machine an OS send SIGTERM signal to all processes due to which processes gets terminated normally, few processes can handle SIGTERM and even after recieving this signal from an OS continues execution, and hence to such processes an OS send SIGKILL signal due to which processes gets terminated forcefully. e.g. SIGTERM: Normal Termination SIGKILL: Forceful Termination SIGSEGV: Process gets terminated with printing message as "Segmentation Fault" SIGSTOP: To suspend a process SIGCONT: To resume suspended porcess etc...

4. **Socket:**

   - Limitation of above all IPC techniques is, only processes running on the same system can communicates, to overcome this limtation Socket IPC mechanism has been designed.
   - By using socket IPC mechanism, process which is running on one machine can communicates with process running on another machine whereas machines are at remote distance from each other provided they connected in a network (either LAN/ WAN/ Internet).
   - **Socket = IP Address + Port Number.** e.g. chat application

Process Coordination/Process Synchronization

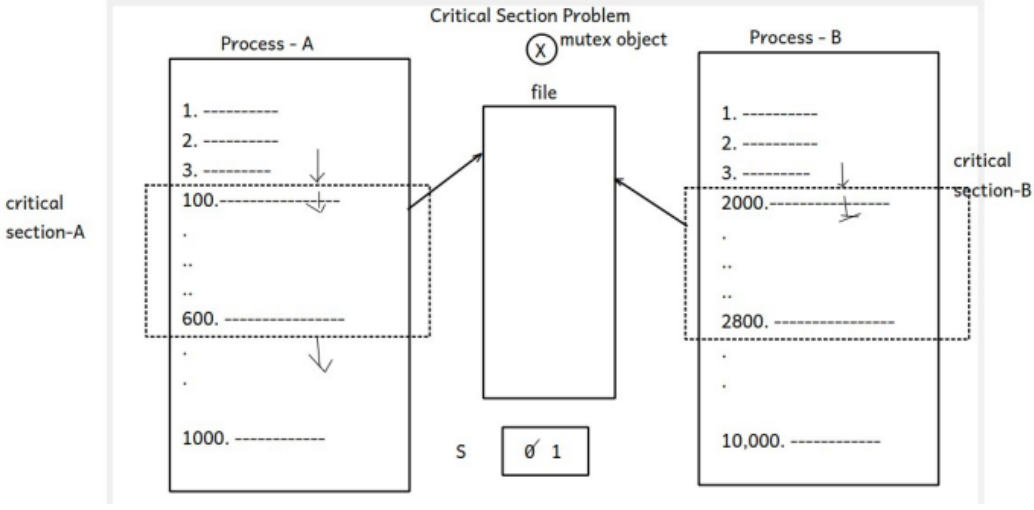
Race Conditions:

A race condition occurs when two or more processes attempt to access a shared resource simultaneously, leading to unpredictable behavior and potential data inconsistency. This is because the final result depends on the sequence or timing of the processes, which is not always guaranteed.

To avoid race conditions, an operating system can implement the following strategies:

1. **Order of Allocation**: The operating system can decide a specific order in which processes are allocated resources. This ensures that not more than one process accesses the resource at the same time.

2. **Final Changes**: The operating system ensures that the changes made by the last process to access the resource are the final changes. This prevents previous processes from overwriting the changes of later processes.



"Data inconsistency" problem occurs in above case only when both the sections of process A & B are running at a same time, and hence these sections are referred as critical section, and hence data inconsistency

problem may occur when two or more processes are running in their critical sections at a sometime, and this problem is also referred as **"critical section problem".**

Synchronization Tools

There are two Synchronization tools:

1. **Semaphore:** This is a variable that is used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system. There are two types of semaphores:
    - **Binary Semaphore**: This is a semaphore with a value of 0 or 1. It can be used when a resource can be acquired by only one process at a time.
    - **Classic Semaphore**: This is a semaphore that can have a value greater than 1. It can be used when a resource can be acquired by more than one processes at a time.
2. **Mutex Object**:
    - This is a synchronization tool that can be used when a resource can be acquired by only one process at a time. A mutex object has two states: **locked** and **unlocked.** At any given time, it can be in only one state.
    - When a process acquires a mutex, it becomes locked, and no other process can acquire it until the owning process releases it, at which point the mutex becomes unlocked. This ensures that the resource being protected by the mutex is accessed by only one process at a time, preventing race conditions.

Deadlock

Deadlock is a situation where a set of processes are unable to proceed because each is waiting for a resource that is held by another process in the set. There are four necessary and sufficient conditions for a deadlock to occur:

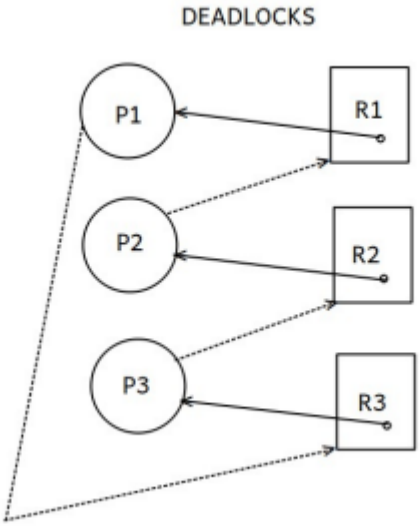
1. **Mutual Exclusion**: This condition states that at least one resource must be held in a non-sharable mode. In other words, only one process can use the resource at any given time. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2. **No Preemption**: This condition states that a resource can only be released voluntarily by the process holding it, after that process has completed its task.
3. **Hold and Wait**: This condition states that a process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
4. **Circular Wait**: This condition states that there exists a set {P0, P1, ..., Pn} of waiting processes such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

Methods for handling deadlocks in operating systems

1. **Deadlock Prevention**: This method aims to prevent deadlocks by ensuring that at least one of the four necessary conditions for a deadlock does not hold. By controlling how resources are allocated to

processes, it's possible to prevent a deadlock from occurring.

2. **Deadlock Detection & Avoidance**: This method involves detecting potential deadlocks before they occur and avoiding unsafe resource allocations. Two key algorithms used for deadlock detection are:
    - **Resource Allocation Graph Algorithm**: This algorithm uses a graph to represent the allocation of resources to processes. If the graph contains a cycle, then a deadlock could occur.



    - **Banker's Algorithm**: This algorithm avoids deadlock by denying or postponing the requests for resources from processes if fulfilling the requests could result in a deadlock.
3. **Deadlock Recovery**: If a deadlock occurs, the system can recover using one of two methods:
    - **Process Termination**: In this method, one of the processes involved in the deadlock (referred to as the "victim") is terminated to break the deadlock.
    - **Resource Preemption**: In this method, resources are forcibly taken away from some processes and given to others so that the deadlock can be broken.

Memory Management

- OS manages "main memory"

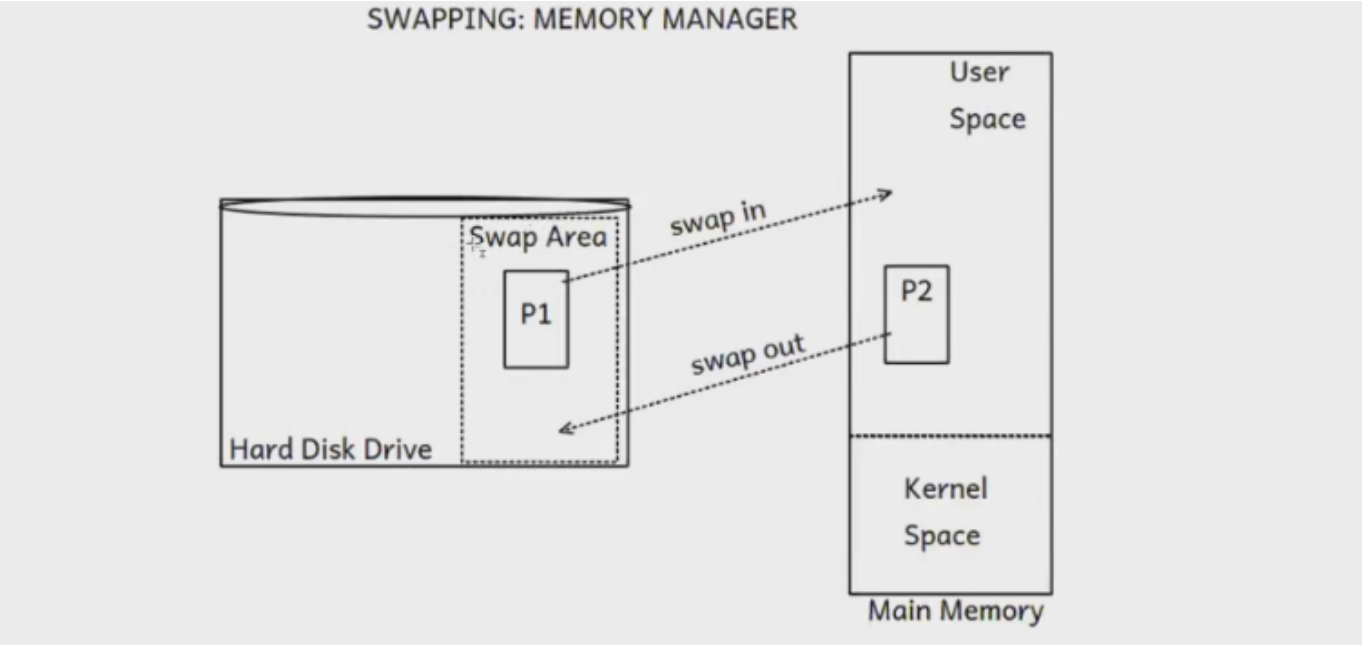
Q. why there is a need for memory (main memory) management

1. **Essential for Program Execution:**
    - Main memory is crucial for the execution of any program. It stores the currently running processes and their data.
2. **Limited Memory:**
    - Main memory is limited in size. The amount of RAM (Random Access Memory) available on a system is finite.
3. **Support for Multi-Tasking:**
    - To achieve maximum CPU utilization, an operating system must support multi-tasking. This means that multiple processes can run concurrently.

4. **Support for Multi-Programming:**
    - Multi-programming involves having multiple processes submitted into the system at the same time. However, since main memory is limited, the OS needs to manage it efficiently to allow the execution of multiple processes.
5. **Memory Protection:**
    - Memory space of one process should be protected from another process. Memory management ensures that each process has its own isolated memory space, preventing one process from interfering with the memory contents of another.

Swapping

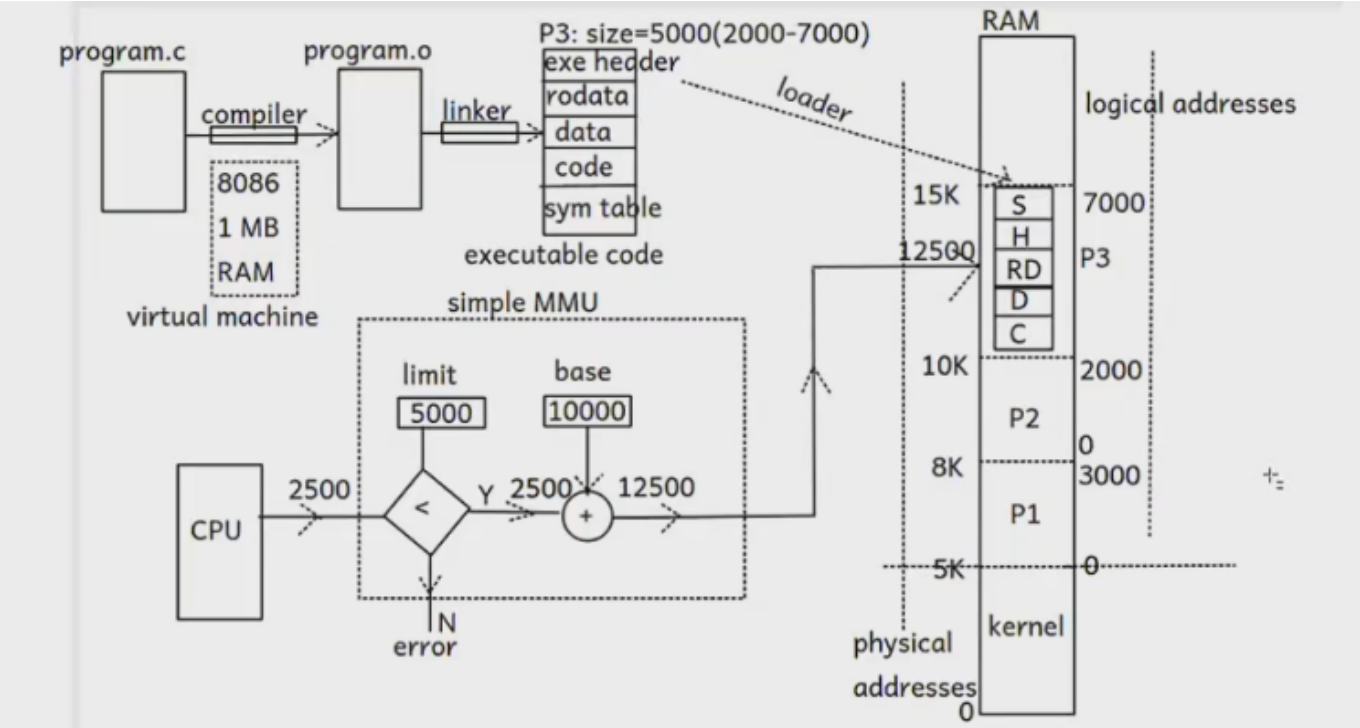
Swapping is done by program of an OS named as **Memory Manager**, it swapin active programs into the main memory from swap area and swapout inactive running programs from the main memory and store them temporarily into the swap area.



1. **Swap Area:**
    - A swap area is a designated portion of the hard disk drive that is reserved during the installation of an operating system.
    - It serves as an extension of the main memory, allowing the operating system to temporarily store inactive running programs or parts of programs that are not currently in use.
2. **Swapping In and Out:**
    - Swapping involves moving processes between the swap area and the main memory.
    - Inactive programs or parts of programs are swapped out from the main memory to the swap area to free up space for other active processes.
    - When a swapped-out process is needed again, it can be swapped back into the main memory.
3. **Linux vs. Windows:**
    - In Linux, a swap area is often implemented as a dedicated swap partition.

- In Windows, the swap area is typically maintained in the form of swap files.

4. **Swap Area Size:**
   - There's a convention, as you mentioned, that suggests the size of the swap area should be roughly double the size of the main memory.
   - For example, if the main memory is 2 GB, the swap area might be recommended to be 4 GB.



There are two methods by which memory gets allocated for processes:

1. Contiguous Memory Allocation
2. Non-Contiguous Memory Allocation

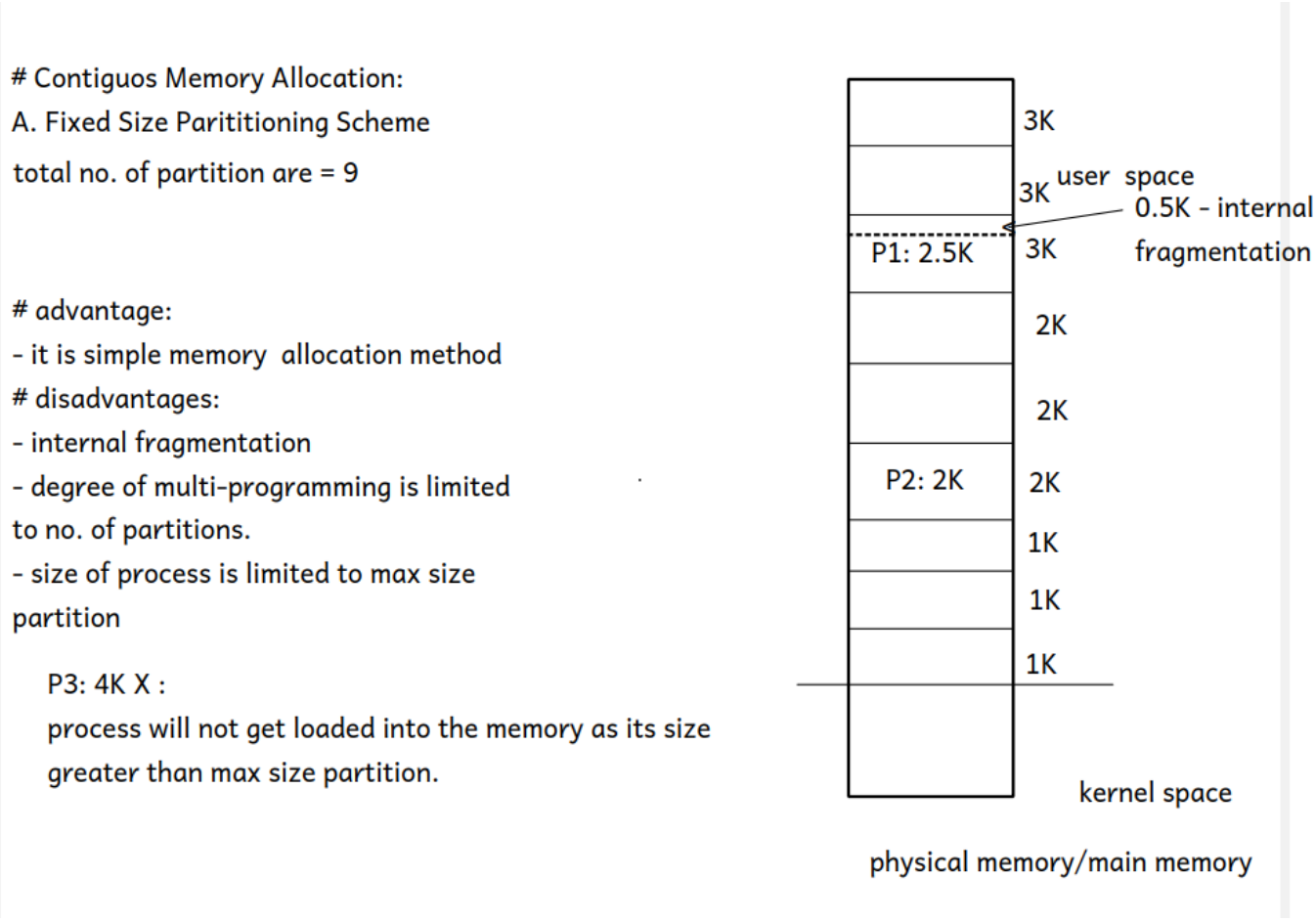
Contiguous Memory Allocation

Contiguous memory allocation is a memory management technique where the main memory (RAM) is divided into fixed-size partitions, and each process is loaded into a single contiguous block of memory. There are two main methods of contiguous memory allocation: fixed size partitioning and dynamic (variable) size partitioning.

1. Fixed Size Partitioning

In fixed size partitioning, the main memory is divided into fixed partitions, each with a predetermined size. When a process is loaded into memory, it is assigned to a partition based on its size. The advantages and

disadvantages of fixed size partitioning are:



- **Advantages:**
  - Simplicity: The allocation process is straightforward.
- **Disadvantages:**
  - Internal Fragmentation: Unused memory within a partition leads to wasted space.
  - Limited Multi-programming: The number of processes that can be accommodated simultaneously is restricted by the number of partitions.
  - Size Limitation: Process size is limited to the size of a partition.

2. Dynamic/Variable Size Partitioning

In dynamic size partitioning, the main memory is divided into partitions of varying sizes to accommodate processes with different memory requirements. External fragmentation is a common issue in dynamic size

partitioning, which can be addressed through compaction or non-contiguous memory allocation methods.

Contiguos Memory Allocation:
B. Variable/Dynamic Size Parititioning Scheme

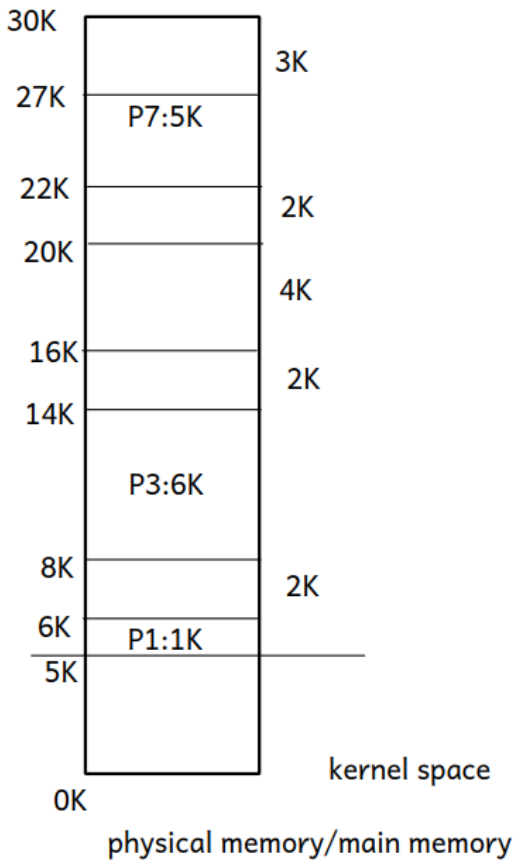
- in this method no. of partitions and size of partitions are not fixed in advanced, initially whole user space is considered as a single free partition, and as processes requests for a memory it gets loaded into it

advantage:
- there is no limit on degree of multi-programming
- there is no limit on size of the process

disadvantages:
- internal fragmentaion
- external fragmentation

P8: 10 K: will not gets loaded into the main memory due to an external fragmentation.

P9: 2K



- **Compaction:**
  - Shuffling memory to consolidate used partitions and free partitions separately.
  - Creates one large contiguous free partition for loading subsequent programs.
- **Non-contiguous Memory Allocation:**
  - Programs can execute even if memory is allocated non-contiguously.
  - Techniques include Segmentation and Paging.

Memory allocation strategies

An OS maintains list of free partitions and when any process is requesting for a memory then this list gets traversed and free partition gets allocated for that process by using any one of the following methods:

1. **First Fit:**
   - In the first-fit memory allocation strategy, the list of free partitions is traversed sequentially, and the first free partition that is large enough to accommodate the process is allocated.
   - This method is relatively fast because it stops searching as soon as a suitable free partition is found.
   - However, it may lead to fragmentation, as the allocated partition may leave smaller free spaces that cannot be efficiently utilized.
2. **Best Fit:**

- Best-fit memory allocation involves traversing the list of free partitions and selecting the smallest partition that is large enough to hold the process.
- This method aims for efficient memory utilization by choosing the partition that minimizes wasted space.
- Best-fit can be more time-consuming than first-fit since it requires finding the smallest suitable partition.

3. **Worst Fit:**

- In the worst-fit memory allocation strategy, the largest free partition in the list is allocated to the requesting process.
- This method attempts to minimize the creation of small free spaces after allocation.
- Like best-fit, worst-fit may require more time to find the largest suitable partition.

- first-fit and best-fit are commonly used in modern operating systems. These methods strike a balance between speed and memory utilization.

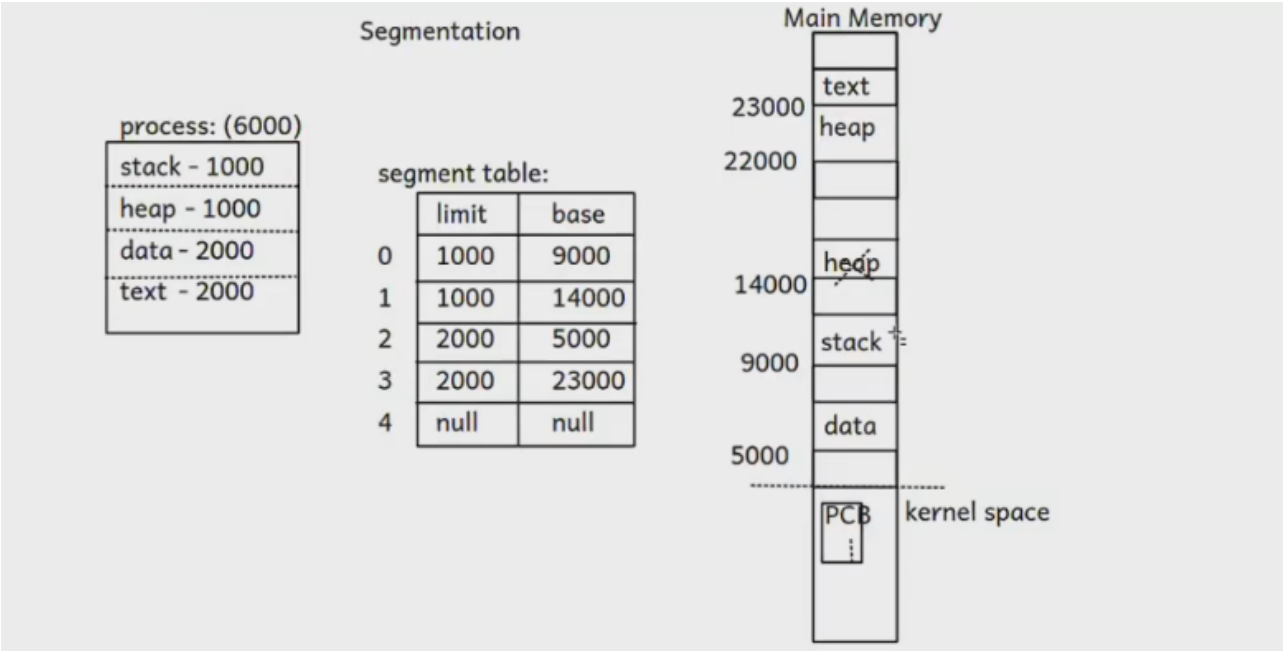
Non-Contiguous Memory Allocation

Non-contiguous memory allocation is a memory management technique where a process is not required to occupy a contiguous block of physical memory. This approach allows a program's various segments or pages to be scattered across different locations in the main memory. Two common methods of non-contiguous memory allocation are Segmentation and Paging.

1. Segmentation

In segmentation, a process is divided into logical segments, each representing a specific type of data or functionality. These segments can be loaded into non-contiguous blocks of physical memory. The key components of segmentation are:

- **Segments:** Logical divisions of a process, such as code segment, data segment, stack segment, etc.
- **Segmentation Table:** The operating system maintains a segmentation table for each process. This table contains information about the base address and length of each segment.

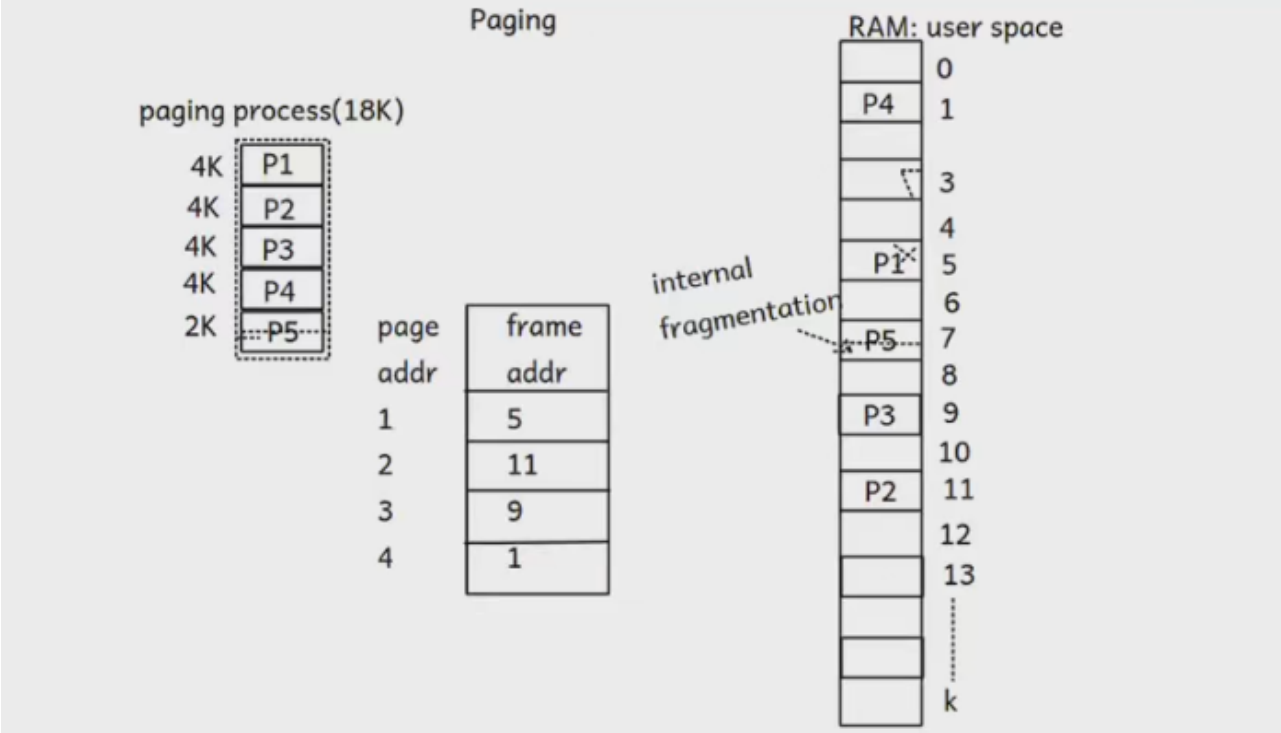


- **Advantages:**
    - Flexibility: Different segments can be loaded independently, allowing for more efficient memory usage.
    - Ease of Sharing: Segments can be shared among processes.
    - Reduced External Fragmentation: Since segments are independent, fragmentation is minimized.
- **Disadvantages:**
    - Internal Fragmentation: Unused memory within a segment may lead to wasted space.
    - Complex Management: Managing and tracking multiple segments for each process can be more complex.

2. Paging

Paging is another non-contiguous memory allocation technique that divides both the physical memory and the process into fixed-size blocks called pages. These pages are then loaded into frames in the physical memory. Key features of paging include:

- **Pages:** Fixed-size blocks into which a process is divided. These pages are loaded into frames in physical memory.
- **Page Table:** The operating system maintains a page table for each process, which maps logical pages to physical frames.



- **Advantages:**
    - No External Fragmentation: Since pages can be loaded into any available free frame, external fragmentation is eliminated.
    - Simplified Memory Management: Paging provides a simple and efficient way to manage memory.
- **Disadvantages:**

- Internal Fragmentation: A page may not be fully utilized, leading to internal fragmentation.
- Overhead: Maintaining the page table incurs some overhead.

"Virtual Memory Management": Paging + Swapping.

- Big size process is divided into pages: few pages of running process can be kept inside the RAM and few pages of the same process can be kept into the swap area
- Active pages of running programs are stored in main memory
- Inactive pages of running programs are stored in swap area
- **Demand Paging/ Lazy Swapper:** Pages are loaded into the main memory from the swap partition only when requested by the process. This approach aims to minimize unnecessary page transfers.
- **Page Fault:** Occurs when a process requests a page, and that page is not present in the main memory.

When all frames in main memory are occupied by pages and a page fault occurs, the operating system needs to bring in a new page from the swap area into a frame in main memory. To make space for the new page, a page replacement algorithm is used to select a page currently residing in a frame to be replaced.

- **Page Replacement Algorithms:** a. **FIFO (First-In-First-Out):**
  - The page that was inserted first into the main memory is the first to be removed.
  - **Advantage:** Simple implementation.
  - **Disadvantage:** May not always yield optimal results in terms of page hits.

  b. **Optimal Page Replacement Algorithm:**
  - Selects the page that will be used in the near future for removal.
  - **Advantage:** Provides the best possible page replacement strategy.
  - **Disadvantage:** Requires knowledge of future page accesses, which is usually not available.

  c. **LRU (Least Recently Used):**
  - Removes the least recently used page from the main memory.
  - **Advantage:** Tries to capture temporal locality.
  - **Disadvantage:** Requires maintaining a detailed record of page access history.

  d. **LFU (Least Frequently Used):**
  - Removes the page that has been used the least number of times.
  - **Advantage:** Considers frequency of use.
  - **Disadvantage:** May not work well in situations where a page is used heavily initially and then not at all.

  e. **MFU (Most Frequently Used):**
  - Removes the page that has been used the most frequently.
  - **Advantage:** Considers frequency of use.
  - **Disadvantage:** Similar to LFU, it may not adapt well to changing patterns of page access.

3. Thrashing

Thrashing occurs in a computer system when there is excessive paging activity, and the system spends more time swapping pages between the main memory and the secondary storage (usually disk) than executing actual processes.

Causes:

1. **Insufficient Memory:** When the available physical memory is not sufficient to accommodate the working set of active processes.
2. **High Degree of Multiprogramming:** Running too many processes simultaneously, leading to intense competition for limited physical memory.

Effects:

1. **Poor Performance:** The system spends a significant amount of time swapping pages, resulting in a decrease in overall system performance.
2. **Increased Overhead:** The CPU spends more time on page swapping than on actual process execution.
3. **Decreased Throughput:** Thrashing can lead to a severe reduction in the number of completed processes within a given time frame.

Mitigation:

1. **Add More Physical Memory:** Increasing the amount of RAM can help accommodate more pages in the main memory.
2. **Optimize Page Replacement Algorithms:** Use efficient page replacement algorithms to minimize unnecessary page swaps.
3. **Reduce Degree of Multiprogramming:** Limit the number of concurrently running processes to ensure that the working set can fit into available physical memory.

Monitoring: System administrators often monitor performance metrics, such as page fault rates and CPU utilization, to detect signs of thrashing and take corrective actions.

File Management

What is a File?

User View:

- A file is a named collection of logically related data or information.
- It serves as a basic storage unit.
- It acts as a container that holds logically related data.

System View:

- A file is a stream of bits or bytes.
- A file consists of two main components:
    1. **Data:** The actual content of the file, located inside the file.
    2. **Metadata:** Information about the file, typically stored in a File Control Block (FCB) or iNode.

File Control Block (FCB) / iNode

The FCB (File Control Block) or iNode is a data structure maintained by the file system for each file. It contains information about the file, both data and metadata.

- **Components of FCB/iNode:**

    1. **iNode Number:** A unique identifier for the file.
    2. **File Name:** The name of the file.
    3. **Parent Folder Location:** The directory where the file is located.
    4. **Time Stamps:** Information about file creation, modification, and access times.
    5. **Access Permissions:** Permissions specifying who can read, write, or execute the file.
    6. **Size of the File:** The size of the file in terms of bytes.
    7. **Other Information:** Various additional details about the file.

- **Purpose:** The FCB or iNode helps the file system keep track of and manage each individual file. It ensures that the file system can locate, organize, and retrieve information about files efficiently.

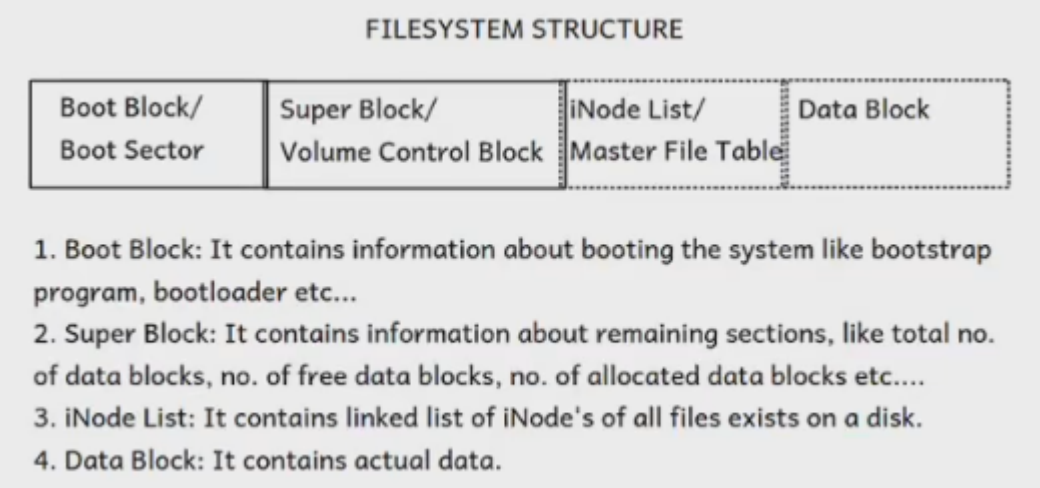
Filesystem Overview

A filesystem is a method of storing data on a disk in an organized manner to facilitate efficient data access.

FileSystems on different OS:

- Linux: "ext" - ext, ext2, ext3, ext4 (Extended FileSystem)
- Windows: NTFS (New Technology File System), FAT, FAT16, FAT32
- UNIX: UFS (UNIX FileSystem)
- MAC OS X: HFS (Hierchical FileSystem)
- FileSystem can be creates at the time installation of an OS "FORMAT": It means to create a new filesystem and not to erase data

Filesystem Components



1. **Boot Sector/Boot Block:**
    - Contains information about the booting system.

    - Typically the first sector of the disk.

2. **Volume Control Block/Super Block:**
    - Contains information about the remaining blocks on the disk.
    - Includes details such as the total number of data blocks, the number of used blocks, the number of free data blocks, etc.
3. **Master File Table/Inode List Block:**
    - Represents a linked list of inodes.
    - Inodes contain metadata about files, such as file size, permissions, timestamps, and pointers to data blocks.
4. **Data Block:**
    - Contains the actual data of files.

- **Efficient Access:** The filesystem is designed to facilitate efficient data access from the disk.
- **Logical Division:** Divides the disk into logically structured components for better organization and management.

Methods of disk space allocation for files

- When a file is requesting for free data blocks, then in which manner free data blocks gets allocated for that file and how its information can be kept inside inode of that file is referred as disk space allocation method.

1. Contiguous Allocation

- Data blocks for a file are allocated in a contiguous manner.
- The entire file is stored as a continuous block on the disk.
- **Advantages:**
    - Simple and efficient for sequential access.
    - Reduces disk head movement.
- **Disadvantages:**
    - Fragmentation: Free space fragmentation can occur, leading to inefficient use of space.
    - Difficulties with dynamic storage needs.

2. Linked Allocation

- A linked list of data blocks is maintained for each file.
- Each block contains a pointer to the next block in the sequence.
- **Advantages:**
    - Flexibility in handling dynamic storage requirements.

- No fragmentation issues.

- **Disadvantages:**
  - Extra space overhead for storing pointers.
  - Random access can be slower compared to contiguous allocation.

3. Index Allocation

- One data block is designated as an "index block."
- The index block contains pointers to all the data blocks of the file.
- **Advantages:**
  - Efficient for both sequential and random access.
  - Reduces the overhead associated with maintaining a linked list.
- **Disadvantages:**
  - Additional storage overhead for the index block.
  - May lead to increased disk head movement during sequential access.

Disk Scheduling Algorithms

- When system want to access data from a disk, request can send to disk controller and disk controller accepts one request at a time and complete it.
- There are chances that at a time more than one requests for accessing data from the disk can be made by the processes running in a system, in that case all the requests can be kept in a waiting queue of the disk maintained by an OS, and there is need to schedule/select only one request at a time and sent it to the disk controller, to do this there are certain algorithms referred as disk scheduling algorithms.

1. FCFS (First Come First Served)

- Requests are serviced in the order they arrive in the waiting queue.
- **Advantages:**
  - Simple and easy to understand.
- **Disadvantages:**
  - May lead to the "convoy effect" where shorter requests get stuck behind longer ones.

2. SSTF (Shortest Seek Time First)

- Selects the request with the shortest seek time (distance between current position and target).
- Aims to minimize disk arm movement.
- **Advantages:**

- Reduces overall seek time.

- **Disadvantages:**
  - May result in starvation for certain requests.

3. SCAN

- Disk arm moves in one direction, servicing requests along the way, until reaching the end, then reverses direction.
- Also known as the "elevator" algorithm.
- **Advantages:**
  - Helps reduce the wait time for requests.
- **Disadvantages:**
  - Requests at the extremes may experience longer wait times.

4. C-SCAN (Circular SCAN)

- Similar to SCAN, but the disk arm moves only in one direction, and when reaching the end, it jumps to the other end.
- **Advantages:**
  - Reduces the maximum wait time for requests.
- **Disadvantages:**
  - Requests at the ends may still experience longer wait times.

5. LOOK

- Similar to SCAN but doesn't go all the way to the end.
- The arm reverses direction when there are no more requests in the current direction.
- **Advantages:**
  - Reduces the arm movement compared to SCAN.
- **Disadvantages:**
  - Similar to SCAN, requests at the extremes may experience longer wait times.