AOS Assignment No. 1

❖ 2 Marks Questions

Q.1.) What is Operating System?

Ans :- An **Operating System (OS)** is software that manages hardware and software resources on a computer or other devices. It acts as an intermediary between users and the computer hardware, enabling users to interact with the machine and run programs. The OS provides essential services like file management, memory management, process scheduling, security, and input/output operations.

Common examples of operating systems include **Windows**, **macOS**, **Linux**, and **Android**. The OS ensures that the hardware is used efficiently and provides a user-friendly interface for interacting with applications.

Key functions of an operating system include:

- **Managing hardware** (CPU, memory, storage devices, etc.)
- **Managing software** (allowing multiple applications to run simultaneously)
- **Providing a user interface** (graphical user interface or command-line interface)
- **Security and access control** (keeping the system safe from unauthorized access)
- **Managing files** (creating, deleting, and accessing files)

Q.2.) List the top famous Operating Systems name.

Ans :- Here are some of the **top famous operating systems**:

1. **Microsoft Windows**
    - Widely used OS for personal computers and laptops.
    - Versions: Windows 10, Windows 11, etc.

2. **macOS**
    - Apple's operating system for Mac computers.
    - Known for its smooth user interface and integration with other Apple devices.
3. **Linux**
    - An open-source OS that comes in various distributions (distros).
    - Popular distros: Ubuntu, Fedora, Debian, CentOS, and Arch Linux.
4. **Android**
    - An open-source mobile operating system based on Linux, used primarily in smartphones and tablets.
5. **iOS**
    - Apple’s mobile operating system for iPhones and iPads.
6. **Chrome OS**
    - Google’s lightweight OS for Chromebooks, focused on cloud-based applications.
7. **Unix**
    - A powerful, multi-user OS widely used in servers, workstations, and high-performance computing.
8. **FreeBSD**
    - A Unix-like open-source OS, known for its reliability and security, often used in servers and embedded systems.
9. **IBM AIX**
    - A proprietary Unix operating system developed by IBM for enterprise servers and workstations.
10. **Solaris** (now Oracle Solaris)

- A Unix-based OS initially developed by Sun Microsystems, known for scalability and security, used in enterprise environments.

Q.3.) What is CPU Scheduling?

Ans :- **CPU Scheduling** is the process of determining which process or task gets to use the CPU at any given time in a multi-tasking operating system. Since only one process can use the CPU at a time, the OS uses scheduling algorithms to manage the execution of processes, ensuring efficient CPU utilization.

Key points about CPU scheduling:

1. **Context Switching**: The OS switches between processes, saving the state of one and loading the state of another to allow multitasking.
2. **Scheduling Algorithms**: Common algorithms include **First-Come-First-Served (FCFS)**, **Shortest Job Next (SJN)**, **Round Robin (RR)**, and **Priority Scheduling**.
3. **Objective**: The goal of CPU scheduling is to maximize CPU utilization, reduce waiting time, and provide fairness among processes.

Q.4.) Explain the Ubuntu OS.

Ans :- **Ubuntu** is a popular, open-source operating system based on **Linux**. It is known for its user-friendly interface, ease of use, and wide community support. Ubuntu is designed to be accessible for both beginners and advanced users.

Key points about Ubuntu:

1. **Open-source**: Ubuntu is freely available, with its source code accessible to the public for modification and distribution.
2. **User-friendly**: It comes with a graphical user interface (GUI) that is intuitive and easy to use, making it a great choice for Linux beginners.
3. **Regular Updates**: Ubuntu is known for its regular release cycle, offering both long-term support (LTS) and regular updates. The LTS versions are supported for five years.

Q.5.) What is Process?

Ans :- A **process** is an instance of a program that is being executed by the operating system. It consists of the program code, its current activity (including the program counter), and all the resources required for execution, such as memory and input/output devices.

Key points about a process:

1. **Program in Execution**: A process is not just a program but the actual running of the program in the CPU.
2. **Process Control Block (PCB)**: Each process is managed by the OS using a data structure called the PCB, which contains information such as the process state, CPU registers, memory management, and scheduling information.
3. **States of a Process**: A process can be in various states, such as **New**, **Ready**, **Running**, **Waiting**, and **Terminated**.

Q.6.) What is PCB?

Ans :- **PCB (Process Control Block)** is a data structure used by the operating system to store information about a process. It is created when a process is initiated and is used to manage and track the process's state during its lifecycle.

Key points about PCB:

1. **Process Information**: It contains key details about the process, such as the **process ID (PID)**, **process state**, **program counter**, **CPU registers**, and **memory management information**.
2. **Scheduling Information**: The PCB also stores information for scheduling, including priority, CPU scheduling information, and pointers to the next PCB in a queue.
3. **State Management**: The PCB helps the OS in managing the transitions of a process between different states (e.g., Ready, Running, Waiting).

Q.7.) What are block and wait state in the Process?

Ans :- In the context of process management, the **Block** and **Wait** states refer to situations where a process is not able to continue its execution immediately due to external factors or resource availability.

1. **Blocked State**:
    - A process is in the **Blocked** state (also called **Waiting** or **Blocked on I/O**) when it cannot proceed because it is waiting for some event or resource, such as I/O operations or user input, to complete.
    - For example, if a process requests data from a disk and the disk is not ready, the process will be blocked until the data is available.
2. **Wait State**:
    - The **Wait** state is similar to the Blocked state, where a process is waiting for an event to occur or a resource to become available. It generally refers to a state in which a process is paused, awaiting a particular condition, such as a signal or inter-process communication.
    - Processes in the **Wait** state cannot be scheduled to run until the waiting condition is resolved.

Q.8.) List the types of Operating Systems.

Ans :- Here are the **types of Operating Systems**:

1. **Batch Operating System**:
    - Processes are executed in batches without user interaction.
    - Tasks are grouped together and processed sequentially without real-time interaction.
    - Example: Early mainframe OSs like IBM's OS/360.
2. **Time-Sharing Operating System**:
    - Multiple users can interact with the system simultaneously by giving each user a small time slice of CPU time.

- Example: UNIX, Linux, and MULTICS.

3. **Distributed Operating System**:
    - Manages a group of independent computers to appear as a single cohesive system to users.
    - These systems coordinate resources across multiple machines in a network.
    - Example: Google's Android, Windows Server.
4. **Network Operating System**:
    - Provides services and manages resources for computers connected in a network.
    - Supports file sharing, printers, and other network services.
    - Example: Microsoft Windows Server, Novell NetWare.
5. **Real-Time Operating System (RTOS)**:
    - Designed for systems that require immediate processing and quick responses to external events.
    - Used in embedded systems, industrial robots, and safety-critical applications.
    - Example: VxWorks, FreeRTOS.
6. **Mobile Operating System**:
    - Operating systems designed specifically for mobile devices like smartphones and tablets.
    - Example: Android, iOS.

Q.9.) What is Multiprogramming?

Ans :- **Multiprogramming** is a technique used by operating systems to run multiple programs or processes simultaneously on a single CPU. The idea is to maximize CPU utilization by allowing the CPU to switch between different

programs while one is waiting for I/O operations to complete, keeping the CPU busy.

Key points about Multiprogramming:

1. **Maximizing CPU Utilization**: While one process is waiting for I/O operations, the CPU can be used to execute another process.
2. **Process Scheduling**: The operating system manages the scheduling of multiple processes, allowing them to share the CPU time effectively.
3. **Increased Efficiency**: Multiprogramming increases the system's overall throughput by reducing idle CPU time, allowing more tasks to be completed in less time.

Q.10.) How to describe Network Operating System?

Ans :- A **Network Operating System (NOS)** is an operating system designed to manage and coordinate the resources and services of computers connected in a network. It provides features for sharing files, printers, and other resources among multiple computers, allowing them to communicate and work together efficiently.

Key points about Network Operating System:

1. **Resource Sharing**: NOS allows sharing of resources such as files, printers, and applications across multiple computers within a network.
2. **Centralized Management**: It often includes centralized management tools for managing network devices, users, and security policies.
3. **Communication and Networking**: It supports networking protocols that enable communication between different devices and provides services like email, remote access, and network security.

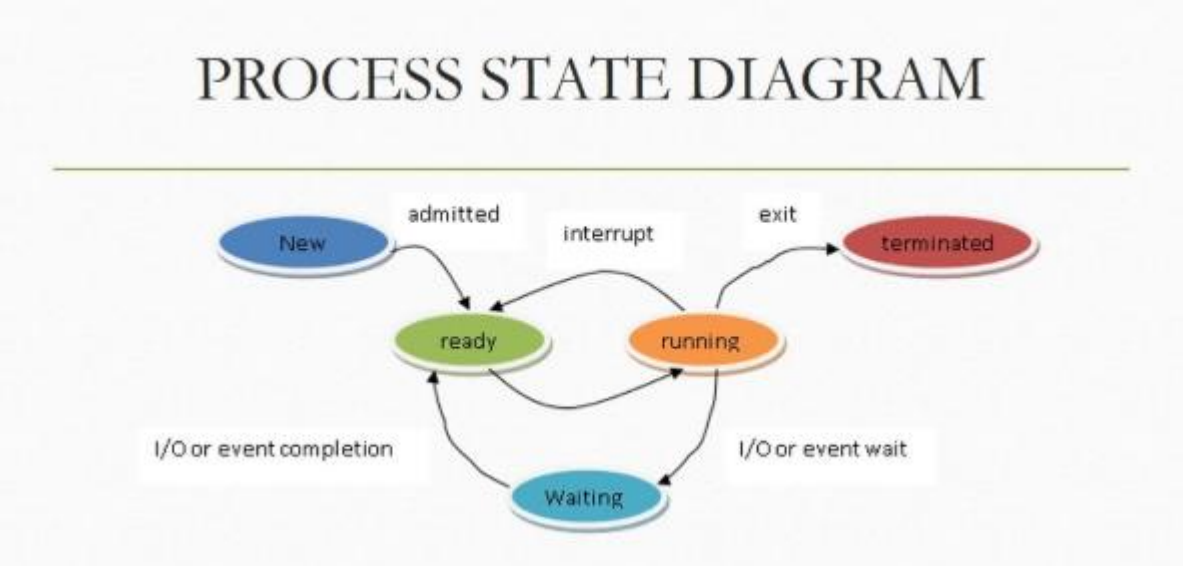
❖ 10 Marks Questions

Q.1.) Explain the different types of process state with neat diagram.

Ans :- A process goes through various states during its lifetime in an operating system. The different **process states** help the operating system manage the execution and scheduling of processes. Here are the main types of process states:

1. **New**:
   The process is being created but is not yet ready for execution. It's in the "New" state while the OS sets up resources for the process.
2. **Ready**:
   The process is prepared and ready to execute but is waiting for CPU time. It is placed in the **Ready Queue**.
3. **Running**:
   The process is currently being executed by the CPU. Only one process can be in the **Running** state at any given time on a single-core CPU.
4. **Waiting** (Blocked):
   The process is waiting for some external event to occur, such as I/O operations or receiving a signal. It cannot proceed until the event happens.
5. **Terminated** (Exit):
   The process has completed its execution or has been terminated due to an error or request. It is then removed from the process table.

Diagram of Process States:



Explanation of transitions:

- A process starts in the **New** state, then moves to **Ready** when it's ready to run.
- From **Ready**, the process is moved to **Running** when the CPU scheduler assigns the CPU to it.
- If the process requires an I/O operation or is waiting for an event, it moves to the **Waiting** state.
- Once the waiting condition is met, the process goes back to the **Ready** state.
- When the process finishes its execution or is terminated, it enters the **Terminated** state.

Q.2.) What is the different purpose of Operating system? Explain.

Ans :- The **Operating System (OS)** serves several key purposes that are essential for the effective operation of a computer system. These purposes ensure that the system runs efficiently, securely, and smoothly. Below are the main purposes of an OS:

1. Process Management:

- The OS manages processes, which are instances of programs in execution. It handles tasks such as scheduling, process creation, and termination.
- It ensures processes are executed in an orderly manner, prevents conflicts, and allocates resources to processes effectively.

2. Memory Management:

- The OS controls the computer's memory, allocating space for processes and ensuring that they do not interfere with each other's memory.
- It handles tasks like tracking which parts of memory are in use, swapping data in and out of memory (paging), and managing virtual memory.

3. File System Management:

- The OS provides a way to store, organize, and retrieve files on storage devices (like hard drives).
- It includes creating, deleting, and accessing files, as well as managing file permissions and ensuring that data is stored securely.

4. Device Management:

- The OS controls hardware devices such as printers, hard drives, and input/output devices (keyboards, mice).
- It uses **device drivers** to facilitate communication between hardware and software, ensuring that devices are used efficiently and without conflict.

5. Security and Access Control:

- The OS enforces security measures to protect against unauthorized access to resources, files, and sensitive information.
- It includes user authentication (like passwords) and access control mechanisms (permissions, encryption) to ensure only authorized users can perform specific actions.

6. User Interface:

- The OS provides a way for users to interact with the computer through a **User Interface (UI)**, which could be a **Graphical User Interface (GUI)** or a **Command-Line Interface (CLI)**.
- This makes the system user-friendly and allows users to execute commands and launch applications.

7. Multitasking and Resource Management:

- The OS allows multiple applications or processes to run concurrently (multitasking), efficiently managing CPU time, memory, and other resources to ensure smooth execution.
- It implements various scheduling algorithms to ensure fair resource allocation and prevent starvation of processes.

8. Networking:

- In modern OS, networking features allow computers to connect with each other over local or wide area networks (LAN/WAN).

- It manages communication protocols, file sharing, and network security to enable efficient data exchange across the network.

9. Error Detection and Handling:

- The OS monitors the system for potential errors and takes corrective actions to prevent or resolve issues, such as system crashes, hardware malfunctions, or resource conflicts.
- It logs errors and provides debugging tools for developers to troubleshoot.

10. Job Scheduling:

- The OS decides the order in which processes or jobs are executed. This scheduling helps in balancing CPU load and maximizing efficiency, particularly in systems with multiple users or processes.

Q.3.) Explain the priority CPU scheduling with an example.

Ans :- **Priority CPU Scheduling** is a scheduling algorithm used by operating systems to assign CPU time to processes based on their **priority levels**. Each process is assigned a priority value, and the CPU is allocated to the process with the highest priority. If two processes have the same priority, the OS uses another scheduling method (such as **FCFS** or **Round Robin**) to decide which process gets the CPU.

Key points of Priority Scheduling:

1. **Priority Value**: Each process has a priority value, where higher values represent higher priority.
2. **Preemptive vs Non-preemptive**:
    - In **preemptive** priority scheduling, a running process can be interrupted if a higher-priority process arrives.
    - In **non-preemptive** priority scheduling, a process continues to run until it finishes or voluntarily gives up the CPU.
3. **Priority Inversion**: A lower-priority process might prevent a higher-priority process from executing, leading to inefficiency, but this can be mitigated using techniques like priority inheritance.

Example of Priority Scheduling:

Let's consider four processes with the following attributes:

Process	Arrival Time	Burst Time	Priority
P1	0	4	3
P2	1	3	2
P3	2	2	4
P4	3	1	1

Here, the priority values range from 1 (lowest priority) to 4 (highest priority).

Step-by-step Execution:

1. **At time 0**:
    - Process P1 arrives with priority 3. Since it is the only process, it starts execution.
2. **At time 1**:
    - Process P2 arrives with priority 2. But since P1 has higher priority (3), P1 continues execution.
3. **At time 2**:
    - Process P3 arrives with priority 4 (highest priority). Since P3 has the highest priority, it preempts P1, and P3 starts executing.
4. **At time 3**:
    - Process P4 arrives with priority 1 (lowest priority). Since it has the lowest priority, it will wait.
5. **Process Completion**:
    - P3 runs first since it has the highest priority. After P3 finishes, the next process with the highest priority (P1) resumes execution and completes.
    - P2 executes next because its priority (2) is higher than P4's priority (1).
    - Finally, P4 executes, as it has the lowest priority.

Gantt Chart:

```
| P3 | P1 | P2 | P4 |
0    2    6    9    10
```

Completion Order:

- **P3** runs from time 0 to 2.
- **P1** runs from time 2 to 6.
- **P2** runs from time 6 to 9.
- **P4** runs from time 9 to 10.

Advantages of Priority Scheduling:

1. **Efficient for critical tasks**: High-priority processes can be executed first, ensuring important tasks are completed quickly.
2. **Customizable**: The priority values can be adjusted according to system requirements (e.g., high-priority system processes, low-priority background tasks).

Disadvantages of Priority Scheduling:

1. **Starvation**: Low-priority processes may never get a chance to execute if high-priority processes keep arriving.
2. **Complexity**: It requires a mechanism to adjust priorities and manage process queuing.

Q.4.) Describe the different operations on the process.

Ans :- The operating system performs various **operations on processes** to manage their lifecycle from creation to termination. Here are the key operations involved in managing processes:

1. Process Creation:

- **Forking**: When a new process is created, it is typically a child of an existing (parent) process. The OS creates a new process control block (PCB) for the new process and allocates necessary resources such as memory, CPU time, and I/O devices.

- **System Call**: A process can create another process through system calls like **fork()** (in UNIX/Linux) or **CreateProcess()** (in Windows).
- The child process can be an identical copy of the parent or have different attributes.

2. Process Scheduling:

- The OS decides which process gets to execute based on its scheduling policies. This involves moving processes from the **Ready Queue** to the **Running State** and allocating CPU time.
- Scheduling algorithms like **FCFS (First Come, First Served)**, **Round Robin**, **Priority Scheduling**, and **Shortest Job First** are used to manage process execution.

3. Context Switching:

- **Context switching** occurs when the OS saves the state (context) of a currently running process and loads the state of a different process. This allows multiple processes to run "concurrently" on a single CPU.
- The process control block (PCB) stores the necessary information about the process's state (e.g., program counter, CPU registers) so that execution can resume from where it was interrupted.

4. Process Termination:

- When a process finishes its execution or is aborted, the OS terminates it. This involves releasing all resources (memory, I/O devices, etc.) associated with the process.
- The process control block (PCB) is removed from the process table, and the operating system may send signals to the parent process or perform other cleanup operations.

5. Process Blocking and Waiting:

- A process may enter the **Blocked (Waiting) State** when it is waiting for some event, such as input/output (I/O) operations to complete or resources to become available.
- The OS moves the process from the **Running** or **Ready** state to the **Blocked** state, where it remains until the event is completed or the required resource becomes available.

6. Process Synchronization:

- Synchronization ensures that multiple processes or threads do not interfere with each other while accessing shared resources. It is especially important in concurrent processing to prevent data corruption.
- **Locks**, **semaphores**, **mutexes**, and **monitors** are commonly used mechanisms to achieve synchronization.

7. Process Communication:

- **Inter-process communication (IPC)** is a method used by processes to communicate with each other. This can be done using techniques like:
    - **Message Passing**: Processes send and receive messages to exchange information.
    - **Shared Memory**: Multiple processes share a portion of memory to read/write data.
- IPC ensures that processes can work together and exchange data in a controlled manner.

8. Process Suspension:

- A process may be suspended temporarily to free up resources for other tasks. The process may be suspended and later resumed (moved back to the **Ready Queue**).
- This operation helps in optimizing system performance, especially in systems with multiple processes competing for limited resources.

9. Priority Adjustment:

- The priority of a process can be changed dynamically, for example, in **priority-based scheduling** algorithms.
- The OS can adjust the priority of a process to ensure fairness and efficiency, or based on factors like process urgency (e.g., interactive vs. background processes).

10. Memory Allocation:

- The OS allocates and deallocates memory for processes. It ensures that processes get enough memory to execute and prevents one process from accessing the memory of another.

- Memory management techniques like **paging**, **segmentation**, and **virtual memory** help manage memory resources efficiently.

11. Process Migration (in Distributed Systems):

- In distributed systems, processes can be migrated from one machine to another for load balancing or fault tolerance.
- The OS supports process migration by saving the process state and transferring it to the target system.

12. Process Control:

- The OS provides mechanisms to control the execution of processes. This includes the ability to pause, resume, or kill processes as needed. System calls like **kill()**, **pause()**, or **wait()** are used to control processes.

Q.5.) Define the Round Robin CPU scheduling with an example.

Ans :- **Round Robin (RR) CPU Scheduling** is one of the simplest and most widely used CPU scheduling algorithms in time-sharing systems. It is a **preemptive** scheduling algorithm where each process is assigned a fixed time slice or **quantum**. The CPU is allocated to each process for the duration of its time slice, and if the process does not complete within that time, it is preempted and moved to the back of the ready queue. The CPU then allocates the next time slice to the next process in the ready queue. This process continues in a circular manner until all processes are finished.

Key Features of Round Robin Scheduling:

1. **Time Quantum**: The fixed amount of time allocated to each process in the ready queue. If a process does not complete during this time, it is preempted.
2. **Preemptive**: If a process doesn't finish its execution within the time quantum, it is placed back in the ready queue, and the next process is given a chance to execute.
3. **Fairness**: Every process gets an equal share of CPU time, which makes it a fair scheduling algorithm, especially in time-sharing environments.

Steps of Round Robin Scheduling:

1. The operating system maintains a **Ready Queue**, where processes are placed waiting for CPU time.
2. The first process in the queue is allocated the CPU for a time quantum.
3. If the process completes execution within its time quantum, it is removed from the queue.
4. If the process does not complete execution within its time quantum, it is preempted and placed back at the end of the ready queue.
5. The CPU scheduler then moves on to the next process in the ready queue, repeating the cycle.

Example of Round Robin Scheduling:

Consider the following processes with their arrival times and burst times (time required to complete execution):

Process	Arrival Time	Burst Time
P1	0	4
P2	1	3
P3	2	5
P4	3	2

Assume the time quantum (time slice) is **3 units**.

Execution Process:

1. **At time 0**: Process P1 starts execution and is given 3 units of time (time quantum). P1 completes 3 units of its 4-unit burst time and is preempted. It goes back to the ready queue, and the remaining burst time of P1 is now 1 unit.
2. **At time 3**: Process P2 starts execution and is allocated 3 units of time. P2 completes its 3 units of burst time and finishes execution at time 6. P2 terminates.
3. **At time 6**: Process P3 starts execution and is given 3 units of time. P3 completes 3 units of its 5-unit burst time and is preempted. It goes back to the ready queue, and the remaining burst time of P3 is now 2 units.

4. **At time 9**: Process P4 starts execution and is allocated 2 units of time, which is its full burst time. P4 finishes execution at time 11. P4 terminates.
5. **At time 11**: Now, process P1, which was preempted earlier, resumes execution. It has 1 unit of burst time remaining, so it completes its execution at time 12. P1 terminates.
6. **At time 12**: Process P3, which was preempted earlier, resumes execution. P3 has 2 units of burst time remaining, so it completes its execution at time 14. P3 terminates.

Gantt Chart for the Above Example:

```
| P1 | P2 | P3 | P4 | P1 | P3 |
0    3    6    9    11   12   14
```

Advantages of Round Robin Scheduling:

1. **Fairness**: Every process gets an equal share of the CPU.
2. **Simple to Implement**: The algorithm is easy to understand and implement.
3. **Preemptive**: Ensures that no process monopolizes the CPU, especially in time-sharing environments.

Disadvantages of Round Robin Scheduling:

1. **Context Switching Overhead**: If the time quantum is too small, there will be frequent context switches, leading to inefficiency.
2. **Non-Optimal for Varying Process Times**: Processes with short burst times may have to wait for longer processes to complete their time slices, leading to inefficiency for those shorter tasks.

Choosing Time Quantum:

- If the **time quantum** is too large, Round Robin behaves like **First-Come-First-Serve (FCFS)**.
- If the **time quantum** is too small, the system may experience too many **context switches**, causing overhead.