OPERATING SYSTEM

UNIT 1

OPERATING SYSTEM

An operating system (OS) is a software program that manages computer hardware and software resources and provides common services for computer programs. The operating system is the most important type of system software in a computer system.

Some popular operating systems include:

- ***Microsoft Windows***: This is the most popular operating system for desktop and laptop computers.
- ***Mac OS***: This is the operating system for Apple Macintosh computers.
- ***Linux***: This is a free and open-source operating system that is popular among developers and server administrators.
- ***Android:*** This is the most popular operating system for smartphones and tablets.
- ***iOS:*** This is the operating system for Apple's iPhone and iPad devices.

Characteristics of Operating System

Memory Management:

The OS is responsible for managing the computer's memory, which is critical to ensure that multiple programs can run at the same time. Memory management is also essential to avoid conflicts between programs and to allocate memory resources efficiently.

Multitasking:

An Operating System is designed to support multitasking, which means that multiple programs can run simultaneously. The OS manages the allocation of resources to different programs, ensuring that each program gets enough resources to operate effectively.

User Interface:

The OS provides a user interface (UI) that enables users to interact with the computer. The UI can be graphical, text-based, or command-line-based, and it enables users to run programs, navigate files, and access various features of the system.

File Management:

An OS provides file management capabilities, which means that it enables users to create, store, retrieve, and manipulate files. The file management system is critical for ensuring that files are organized and can be accessed efficiently.

Device Management:

An Operating System is responsible for managing the computer's hardware devices, such as printers, scanners, and cameras. The OS provides a device driver, which is a software program that enables the computer to communicate with the device.

Security:

An OS is designed to provide security features to protect the computer system from malicious software and other threats. It provides authentication mechanisms, access controls, and other security features to ensure that only authorized users can access the system.

Virtual Memory:

An Operating System provides virtual memory, which enables the computer to use more memory than it physically has. The OS creates a virtual memory space that is larger than the physical memory and uses it to store data that is not currently in use.

Networking:

An OS provides networking capabilities, enabling the computer to connect to other computers or devices. The OS manages the network connection, allowing the computer to send and receive data over the network.

Fault Tolerance:

An Operating System is designed to be fault-tolerant, which means that it can continue to operate even if there is a hardware or software failure. The OS provides error detection and correction mechanisms to ensure that the system remains stable and operational.

Evolution of Operating System:

The evolution of operating systems started in the 1950s with the development of the first operating system for the UNIVAC I computer. In the 1960s, IBM developed the first mainframe operating system, OS/360, which enabled the sharing of computing resources between multiple users. The development of minicomputers and microcomputers led to the creation of smaller operating systems such as UNIX and DOS in the 1970s. The 1980s saw the introduction of graphical user interfaces (GUIs) with the release of Apple’s Macintosh operating system and Microsoft’s Windows. In the 1990s, networked operating systems such as Linux and Windows NT were developed, followed by mobile operating systems in the

2000s such as iOS and Android. Today's modern operating systems provide advanced features such as virtualization, cloud computing, and machine learning.

Types of Operating System:

Real-Time Operating Systems (RTOS):

A real-time operating system is designed to provide predictable and timely response to events or requests from other systems. It is used in systems that require high reliability, safety, and responsiveness. Examples of applications that use RTOS include aerospace systems, medical devices, and industrial automation systems.

Multi-User Operating Systems:

A multi-user operating system is designed to support multiple users simultaneously. It allows users to share the resources of the system and work on different tasks at the same time. Examples of multi-user operating systems include UNIX, Linux, and Windows Server.

Network Operating Systems (NOS):

A network operating system is designed to manage the resources of a network. It allows users to share files, printers, and other resources over a network. Examples of network operating systems include Novell Netware, Windows Server, and Linux.

Mobile Operating Systems:

A mobile operating system is designed to run on mobile devices such as smartphones and tablets. It provides features such as touch-based interfaces, sensors, and mobile network connectivity. Examples of mobile operating systems include Android, iOS, and Windows Mobile.

Single-User Operating Systems:

A single-user operating system is designed to support one user at a time. It is commonly used on personal computers and laptops. Examples of single-user operating systems include Windows, macOS, and Linux.

Embedded Operating Systems:

An embedded operating system is designed to run on small devices with limited resources such as routers, digital cameras, and gaming consoles. It provides a low-footprint operating system that can operate on limited hardware resources. Examples of embedded operating systems include VxWorks, Windows Embedded, and Linux.

Distributed Operating Systems:

A distributed operating system is designed to manage a network of independent computers that work together as a single system. It allows users to access resources from multiple computers and provides fault tolerance and load balancing. Examples of distributed operating systems include Amoeba, Windows NT, and Linux.

Functions of an operating system:

- ***Resource Management***: An operating system manages the computer's resources, such as the CPU, memory, and storage, and allocates them to different programs and processes. It ensures that each program gets its fair share of resources and avoids resource conflicts between different programs.
- ***Memory Management***: An operating system manages the memory of a computer system by allocating and deallocating memory to different programs and processes. It ensures that programs and processes do not interfere with each other's memory space and avoids memory conflicts.
- ***Process Management:*** An operating system manages the execution of different programs and processes on a computer system. It creates and terminates processes and manages their state, including waiting, running, and blocked.
- ***File Management***: An operating system manages the files and directories of a computer system. It creates, reads, writes, and deletes files and directories and manages their permissions and access control.
- ***Device Management***: An operating system manages the input and output devices of a computer system, such as the keyboard, mouse, printer, and monitor. It provides a common interface for different devices and manages their drivers and communication protocols.
- ***User Interface:*** An operating system provides a graphical user interface (GUI) or a command-line interface (CLI) to interact with the computer system. The GUI allows users to interact with the computer using windows, menus, icons, and buttons, while the CLI allows users to interact with the computer using text commands.
- ***Security Management***: An operating system provides security features to protect the computer system from unauthorized access and malicious software. It manages user authentication, access control, and data encryption to ensure the system's security and integrity.
- ***Error Handling:*** An operating system detects and handles errors and exceptions that may occur in the computer system. It provides error messages and logs to help users and system administrators diagnose and fix system errors.

Concept of Systems Calls:

A system call is a way for programs to request services from the operating system. It is an interface between the application software and the operating system kernel. When a program needs to access system resources, it makes a system call, which transfers control to the operating system kernel. The operating system kernel performs the required operation and returns control to the application.

The system call mechanism provides a safe and controlled way for user programs to interact with the kernel. The kernel exposes a set of system calls, each of which has a unique system call number. The user program can invoke these system calls using the system call number, and the kernel will execute the corresponding operation.

System calls can be broadly categorized into the following types:

- ***Process Control System Calls***: These system calls are used to manage processes, such as creating or terminating processes, waiting for a process to complete, and changing the process priority.
- ***File System Calls:*** These system calls are used to perform file-related operations such as creating, reading, writing, and deleting files. Examples of file system calls include open(), close(), read(), write(), and unlink().
- ***Device System Calls***: These system calls are used to manage input and output devices, such as printers and keyboards. Examples of device system calls include ioctl(), read(), and write().
- ***Information System Calls:*** These system calls are used to obtain information about system resources, such as process IDs, file sizes, and system configurations.

Process Management:

Process management is an essential part of operating systems that deals with the execution of programs or tasks, known as processes. A process is a program in execution that requires system resources, including CPU time, memory, and input/output devices. The operating system manages and controls these processes, ensuring that they are executed efficiently and safely. The following are detailed notes on the process concept in operating systems:

Process Concept:

- A process is a program in execution that requires system resources to complete the task.
- Each process has its memory space, execution state, and system resources.
- The operating system uses a process control block (PCB) to store all the necessary information about a process, including the process state, priority, and memory allocation.
- The process control block is a data structure that stores all the relevant information about a process, including the process state, program counter, registers, memory allocation, and other information.

- The operating system uses the process control block to manage the execution of a process, including scheduling, synchronization, and communication between processes.
- The process state refers to the current state of the process, which can be either running, waiting, or blocked.
- The process priority is used to determine the order in which the processes are executed.
- The memory allocation refers to the memory space allocated to a process, including the code, data, and stack.
- The operating system uses a scheduler to manage the execution of processes. The scheduler decides which process to run next based on their priority, the process state, and the available system resources.
- The operating system uses a context switch to switch between processes, which involves saving the current process's state and restoring the state of the next process to be executed.
- The operating system provides various system calls and API functions for process management, including process creation, termination, synchronization, and communication.

Process Scheduling:

- Process scheduling is an essential part of process management that determines which process to run next.
- The scheduler decides which process to run based on their priority, the process state, and the available system resources.
- The scheduler uses various scheduling algorithms, such as First Come First Serve (FCFS), Round-Robin (RR), and Priority Scheduling, to determine the order of process execution.
- FCFS scheduling algorithm executes the processes in the order they arrive in the system.
- RR scheduling algorithm executes the processes in a circular order with a fixed time slice for each process.
- Priority Scheduling executes the process with the highest priority first.
- The operating system can use a combination of these scheduling algorithms to optimize system performance.

Process Synchronization:

- Process synchronization is an essential part of process management that deals with the coordination of processes.
- The operating system provides various synchronization mechanisms, such as semaphores, mutexes, and monitors, to ensure that processes do not interfere with each other's execution.
- Semaphores are used to provide mutual exclusion between processes, ensuring that only one process can access a shared resource at a time.
- Mutexes are used to protect shared resources from concurrent access by multiple processes.
- Monitors are used to provide synchronization between processes, allowing only one process to access a shared resource at a time.
- The operating system provides various synchronization primitives to ensure the safe and efficient execution of processes.

Process state

A process state is the condition in which a process exists at a particular point in time. There are several states that a process can be in, including running, ready, blocked, and terminated. The running state indicates that the process is currently executing instructions, while the ready state means that the process is waiting to be executed. The blocked state occurs when a process is waiting for a resource or event, while the terminated state indicates that the process has completed its execution.

The process state is essential to the efficient management of system resources, as it enables the operating system to control and allocate resources such as CPU time and memory effectively.

Process Control Block (PCB):

The Process Control Block (PCB) is a data structure that the operating system uses to manage information about each process in the system. The PCB contains information about the process, including its process state, program counter, CPU registers, memory allocation, and the status of the process's resources.

When a process is created, the operating system creates a corresponding PCB to store this information. The PCB is then used to manage the process throughout its lifetime, and the information in the PCB is updated as the process moves between different states.

The PCB is a critical component of modern operating systems, as it enables the operating system to manage and control system resources effectively.

Context switching

Context switching is a process by which an operating system saves and restores the state of a process or a thread so that the process or thread can be resumed from the same point when it is next executed.

When a system executes multiple processes simultaneously, it needs to switch between them, and context switching is the process of switching between these processes.

The operating system saves the state of a process or a thread, which includes the values of its registers, program counter, and stack pointer, and restores the state of the next process or thread to be executed. The process or thread that was running is paused, and its context is saved to memory, and then the next process or thread is loaded into the CPU, and its context is restored.

There are two main types of context switching: process context switching and thread context switching. Process context switching occurs when the operating system switches between two or more processes, whereas thread context switching occurs when the operating system switches between threads within the same process. The process context switch involves saving and restoring the complete state of the process, while the thread context switch involves saving and restoring only the state of the thread.

CPU scheduling:

CPU scheduling is an essential component of operating systems that manages the allocation of processing time for different processes running on the system. The scheduling algorithms determine the order in which processes are executed on the central processing unit (CPU). There are different types of schedulers that are used to manage the CPU scheduling in operating systems.

Long-Term Scheduler: The long-term scheduler is responsible for selecting new processes to be admitted into the system. This scheduler determines which processes should be moved from the job queue to the ready queue. The main objective of the long-term scheduler is to maintain an efficient system by ensuring that the number of processes in the system is optimal.

Short-Term Scheduler: The short-term scheduler, also known as the CPU scheduler, determines which process from the ready queue should be executed next. It selects processes from the ready queue and assigns the CPU to the selected process. The objective of the short-term scheduler is to provide optimal CPU utilization by minimizing the waiting time of processes.

Medium-Term Scheduler: The medium-term scheduler is responsible for managing the processes in the memory. It decides which processes should be swapped out of memory and which ones should be swapped into memory. The main objective of the medium-term scheduler is to manage the memory effectively to improve the overall performance of the system.

Scheduling Criteria:

The scheduling criteria are the set of parameters that the scheduler uses to decide which process to allocate resources to. Here are some common scheduling criteria:

- ***CPU Utilization***: The CPU utilization criterion measures how efficiently the CPU is being used. The scheduler tries to keep the CPU utilization high by allocating CPU resources to processes that are ready to run.

- ***Throughput:*** The throughput criterion measures the number of processes that are completed in a given time. The scheduler tries to maximize the throughput by allocating resources to processes that can be completed quickly.
- ***Turnaround Time:*** The turnaround time criterion measures the time it takes for a process to complete, from the time it is submitted to the time it finishes. The scheduler tries to minimize the turnaround time by allocating resources to processes that have been waiting for a long time.
- ***Waiting Time:*** The waiting time criterion measures the amount of time a process spends waiting in the ready queue before it is allocated resources. The scheduler tries to minimize the waiting time by allocating resources to processes that have been waiting for the least amount of time.
- ***Response Time:*** The response time criterion measures the time it takes for a process to start responding to a user's input. The scheduler tries to minimize the response time by allocating resources to processes that are most likely to respond quickly.

Scheduling Algorithms:

Scheduling algorithms are the methods used by the scheduler to allocate resources to processes. Here are some common scheduling algorithms:

First-Come-First-Served (FCFS):

In FCFS, the processes are allocated resources in the order in which they arrive. The first process that arrives is allocated resources first, followed by the next process, and so on.

Shortest Job First (SJF):

In SJF, the processes are allocated resources based on their expected execution time. The process that requires the least amount of time is allocated resources first, followed by the next process, and so on.

Priority Scheduling:

In priority scheduling, each process is assigned a priority value, and the process with the highest priority is allocated resources first. Processes with the same priority are allocated resources based on the FCFS algorithm.

Round Robin Scheduling:

In round-robin scheduling, the processes are allocated resources in a circular queue. Each process is allocated a fixed time slice, and the scheduler switches between processes in a circular order.

Multilevel Feedback Queue Scheduling:

In multilevel feedback queue scheduling, the processes are allocated resources based on their priority and feedback from previous execution. The scheduler maintains multiple queues, and processes are moved between queues based on their priority and feedback.

Critical Section Problem:

The critical section problem is a classic problem in computer science that arises in multi-threaded or multi-process systems where multiple processes or threads share a common resource. A critical section is a section of code that accesses a shared resource and must be executed by one process at a time to avoid race conditions and ensure consistency. The critical section problem can be solved using various synchronization techniques such as semaphores and monitors.

Semaphores:

A semaphore is a synchronization primitive that is used to manage access to a shared resource in a multi-process or multi-threaded environment. A semaphore is essentially a counter that can be incremented or decremented atomically, and it can be used to manage access to critical sections of code. The two most common types of semaphores are binary semaphores and counting semaphores.

- ***Binary Semaphores***: A binary semaphore is a semaphore that can only take on two values, 0 and 1. It is used to manage access to a critical section where only one process can enter at a time. When a process enters the critical section, it sets the value of the semaphore to 0, indicating that the critical section is busy. When the process leaves the critical section, it sets the value of the semaphore to 1, indicating that the critical section is free.
- ***Counting Semaphores***: A counting semaphore is a semaphore that can take on any non-negative integer value. It is used to manage access to a critical section where multiple processes can enter at a time, subject to a limit. When a process enters the critical section, it decrements the value of the semaphore, and when it leaves, it increments the value of the semaphore.

Inter-process Communication (IPC):

Inter-process communication (IPC) is a mechanism for exchanging data between processes in a multi-process environment. It allows processes to communicate and synchronize their activities, and it can be implemented using various techniques such as pipes, shared memory, and message passing.

- ***Pipes***: A pipe is a mechanism for transferring data between two processes, where one process writes to the pipe and the other process reads from it. Pipes can be implemented as unidirectional or bidirectional, and they can be used for inter-process communication or input/output redirection.
- ***Shared Memory:*** Shared memory is a technique for sharing data between processes, where a region of memory is mapped to the address space of multiple processes. Processes can read

from and write to the shared memory region, and changes made by one process are visible to other processes.

- ***Message Passing:*** Message passing is a technique for exchanging data between processes, where a process sends a message to another process, and the receiving process reads the message. The message can be sent asynchronously or synchronously, and it can contain data or control information.