Unit –1: Introduction of Operating System

➢ Explain needs of Operating system:

- An operating system serves various crucial needs, including managing hardware resources, providing a user interface, facilitating communication between software and hardware, ensuring system security, and coordinating task scheduling for efficient operation.
- It acts as a mediator between applications and computer hardware, enabling a seamless user experience while optimizing resource utilization and maintaining system stability.

➢ Fundamental Goals of Operating system:

1. Resource Management: Efficiently handle computer hardware like CPU, memory, and storage.
2. User Interface: Provide an easy way for users to interact with the computer.
3. Security: Keep the system safe from unauthorized access and malicious software.
4. Communication: Enable software to communicate with hardware, ensuring smooth operation.
5. Task Scheduling: Organize and optimize tasks for efficient use of resources and performance.
6. Error Handling: Detect and manage errors to prevent system crashes and ensure reliability.
7. File Management: Organize and control access to files, ensuring data is stored and retrieved accurately.

Types of Operating System:

i. Multi-programming:

- Definition: Multi-programming is an operating system strategy where multiple programs are loaded into the main memory simultaneously.
- Objective: The goal is to keep the CPU busy at all times, increasing overall system efficiency and throughput.
- How it works: While one program is waiting for I/O, the CPU can execute another program, thus maximizing CPU utilization.
- Efficient CPU Utilization: Ensures that the CPU is always busy executing some job, improving overall system performance.
- I/O Overlap: While one program waits for I/O operations, the CPU can execute another program, minimizing idle time.
- Memory Utilization: Multiple programs are kept in the main memory simultaneously, optimizing memory usage.
- Throughput Improvement: Increases the number of programs processed in a given time, enhancing system throughput.
- Context Switching: Involves switching between different programs, known as context switching, to maintain continuous execution.
- Priority Scheduling: Operating system may use priority scheduling to assign higher priority to certain programs for faster execution.

- Batch Processing: Well-suited for batch processing environments where similar tasks are executed in large quantities.
- Job Scheduling: Efficient job scheduling is crucial to manage the execution order of various programs.
- System Resource Allocation: Involves effective allocation of resources such as CPU time, memory, and I/O devices.
- Error Containment: A failure in one program does not necessarily affect others, ensuring a degree of fault tolerance.

ii. Time Sharing:

- Definition: Time Sharing, also known as multitasking, involves dividing the CPU time among multiple users or tasks.
- Objective: The primary aim is to provide each user with the illusion that they have sole control over the CPU, even though it is being shared among many users.
- How it works: The operating system rapidly switches between tasks, giving each user a small time slice to execute their commands.
- Interactive Environment: Designed for multiple users to interact with the system simultaneously.
- Fair CPU Allocation: Each user gets a fair share of the CPU time, preventing one user from monopolizing resources.
- Response Time: Aims for short response times to create an illusion of dedicated computing resources for each user.
- Dynamic Task Switching: Rapidly switches between tasks to give the appearance of concurrent execution.
- Terminal Handling: Manages interactions with multiple terminals or users connected to the system.
- User Authentication: Requires robust user authentication mechanisms to ensure secure access.
- Task Scheduling: Prioritizes tasks based on user priority, preventing resource starvation.
- Resource Quotas: Implements resource quotas to limit the amount of system resources a user or task can consume.
- Concurrency Control: Ensures proper control and synchronization of concurrent access to shared resources.
- Idle Time Utilization: Attempts to use idle CPU time to perform background tasks, enhancing overall efficiency.

iii. Real-Time:

- Definition: Real-time operating systems (RTOS) are designed to process and respond to events within a specific time frame.
- Objective: The primary goal is to ensure that tasks are completed within predefined deadlines, which is crucial in applications like industrial automation, medical systems, and aerospace.
- How it works: Real-time systems often have a predictable and deterministic response time to external stimuli, guaranteeing timely execution of critical tasks.
- Predictable Response Time: Guarantees that tasks will be completed within specified time constraints.

- Deadline Sensitivity: Emphasizes meeting deadlines for critical tasks, crucial in applications like medical equipment and robotics.
- Task Prioritization: Assigns priorities to tasks based on their criticality to ensure high-priority tasks are completed first.
- Deterministic Behavior: Provides a consistent and predictable execution pattern for real-time tasks.
- Fault Tolerance: Incorporates redundancy and error handling mechanisms to ensure system reliability.
- Hard Real-Time vs. Soft Real-Time: Differentiates between systems with strict deadlines (hard real-time) and those with more flexible constraints (soft real-time).
- Embedded Systems: Commonly used in embedded systems where timing and precision are crucial.
- Clock Synchronization: Requires synchronized clocks across the system for accurate time-sensitive operations.
- Interrupt Handling: Efficient handling of interrupts to respond promptly to external events.
- Safety-Critical Systems: Widely used in safety-critical applications like air traffic control and nuclear power plants.

iv. Multithreading:

- Definition: Multithreading is an extension of multitasking where multiple threads within a single process run concurrently.
- Objective: It aims to improve application performance by parallelizing the execution of tasks within a process.
- How it works: Different threads within a process share the same resources (memory space), allowing for more efficient communication and coordination.
- Thread Creation: Allows multiple threads within a single process, each with its own program counter and register set.
- Shared Resources: Threads within a process share the same resources, including memory space and file descriptors.
- Parallelism: Enables parallel execution of tasks, improving overall application performance.
- Efficient Communication: Threads within the same process can communicate more easily than separate processes.
- User-Level Threads vs. Kernel-Level Threads: Differentiates between threading models, with user-level threads managed entirely by the application and kernel-level threads supported by the operating system.
- Resource Sharing: Threads share resources, such as code and data sections, reducing overhead compared to separate processes.
- Context Switching Overhead: Context switching between threads within the same process is typically faster than switching between processes.
- Thread Synchronization: Involves managing access to shared resources to prevent conflicts and ensure data consistency.
- Fault Isolation: A failure in one thread does not necessarily affect the entire process, enhancing fault isolation.
- Scalability: Well-suited for applications that can be divided into smaller, concurrent tasks, leading to better scalability.

v. Distributed:

- Definition: Distributed operating systems involve multiple interconnected computers working together as a single system.
- Objective: The main goal is to provide users with access to resources and services available on the network, treating the distributed environment as a unified computing resource.
- How it works: Tasks can be distributed across various machines, and the operating system manages communication, synchronization, and resource sharing among the connected nodes.
- Resource Sharing: Enables sharing of resources such as files, printers, and computational power across a network.
- Fault Tolerance: Provides redundancy and fault tolerance, reducing the impact of failures in a distributed system.
- Transparency: Strives for transparency in terms of access, location, migration, and relocation of resources in the network.
- Scalability: Allows the system to scale horizontally by adding more machines to the network.
- Interprocess Communication: Involves communication between processes running on different machines in the network.
- Load Balancing: Distributes tasks across multiple machines to ensure balanced resource utilization.
- Security Challenges: Introduces new security challenges, such as ensuring secure communication and access control across a network.
- Distributed File System: Provides a distributed file system that allows users to access files stored on different machines seamlessly.
- Consistency and Replication: Addresses challenges related to maintaining consistency and handling data replication in a distributed environment.
- Centralized vs. Decentralized Control: Varies in terms of having a centralized server for control and coordination or distributing control among nodes in the network.

OS services – User point of view and System point of view:

1. User Point of View:

- User Interface:

Description: The OS provides a user interface for interacting with the computer system.
Details: This could be a graphical user interface (GUI) with icons and windows, a command-line interface (CLI) where users type commands, or a combination of both.

- File Management:

Description: Organizing and managing files and directories on the storage devices.
Details: Users can create, delete, copy, and move files. The OS ensures data integrity and provides a hierarchical structure for organizing information.

- Device Management:

Description: Handling communication between the computer and external devices.
Details: Users can connect and use devices like printers, scanners, and external storage. The OS manages device drivers and ensures proper communication.

- Memory Management:

Description: Efficient allocation and deallocation of memory resources.
Details: Users can run multiple applications simultaneously, and the OS manages the allocation of RAM to ensure each application gets the necessary memory for smooth operation.

- Process Management:

Description: Overseeing the execution of processes or programs.
Details: Users can launch and terminate applications. The OS handles task scheduling, ensuring fair CPU utilization and efficient multitasking.

- Security and Authentication:

Description: Ensuring system and data security.
Details: Users can set passwords, control access to files, and the OS implements authentication mechanisms to protect user data and system resources.

- Networking:

Description: Facilitating communication between computers in a network.
Details: Users can connect to the internet, share files, and access resources on other machines. The OS manages network protocols and configurations.

- Utilities:

Description: Additional tools and applications that enhance user experience.
Details: Users have access to utilities like text editors, calculators, and system maintenance tools provided by the OS.

- Error Handling:

Description: Detecting and managing errors to maintain system stability.
Details: Users receive error messages and prompts, and the OS takes actions to prevent crashes and data loss.

- Updates and Maintenance:

Description: Keeping the OS up-to-date and maintaining system health.
Details: Users can install updates and patches provided by the OS to ensure security and stability.

2. System Point of View:

- Process Scheduling:

Description: Deciding the order in which processes are executed by the CPU.
Details: The OS uses scheduling algorithms to manage the execution of multiple processes, optimizing CPU utilization.

- Memory Allocation:

Description: Assigning memory space to processes and managing memory resources.
Details: The OS allocates and deallocates memory, prevents conflicts between processes, and ensures efficient use of RAM.

- Device Drivers:

Description: Providing interfaces for communication with hardware devices.
Details: The OS includes device drivers that allow applications to interact with hardware components, ensuring compatibility and efficient device usage.

- Interrupt Handling:

Description: Responding to hardware or software-generated interrupts.
Details: The OS manages interrupts to handle events like keyboard input, hardware errors, and external device signals.

- File System Management:

Description: Organizing and controlling access to files and directories.
Details: The OS manages the file system, handling file operations, enforcing permissions, and ensuring data integrity.

- Security and Access Control:

Description: Implementing mechanisms to control user access and protect system resources.
Details: The OS enforces user permissions, authentication, and encryption to safeguard the system and user data.

- Error Logging and Reporting:

Description: Logging and reporting errors for system diagnostics.
Details: The OS maintains logs of system events, errors, and performance metrics to aid in troubleshooting and maintenance.

- Network Communication:

Description: Facilitating communication between devices in a network.

Details: The OS manages network protocols, routing, and communication processes, ensuring seamless data exchange.

- System Calls:

Description: Providing an interface for applications to request OS services.
Details: Applications make system calls to access OS functions such as file operations, memory allocation, and process control.

- Resource Allocation:

Description: Distributing and managing resources like CPU time, memory, and I/O devices.
Details: The OS allocates resources based on priority, preventing resource conflicts and ensuring efficient system operation.

Case Study: Linux

Introduction:
Linux, an open-source operating system kernel, has evolved over the years, playing a pivotal role in the world of computing. This case study explores the history, features, and impact of Linux, highlighting its significance in the realm of operating systems.

Background:
In the early 1990s, Linus Torvalds, a computer science student in Finland, initiated the development of Linux as a hobby project. With contributions from the global open-source community, Linux quickly gained momentum, transforming into a robust and feature-rich operating system.

Features and Architecture:
Linux is renowned for its stability, security, and versatility. Its architecture, based on the Unix operating system principles, incorporates a monolithic kernel that manages hardware resources and provides a foundation for various user-space applications. The file system structure, adherence to POSIX standards, and support for multitasking contribute to its efficiency.

Community Collaboration:
One of Linux's defining features is its open-source nature, fostering a collaborative development model. A diverse community of developers, comprising both volunteers and professionals, contributes to the constant improvement of the Linux kernel and associated software. This collaborative spirit has led to rapid advancements, timely bug fixes, and the creation of a vast repository of software packages.

Enterprise Adoption:
Linux's stability and security features have made it an ideal choice for enterprise environments. Many businesses worldwide have adopted Linux for servers, powering critical infrastructure, web servers, and cloud computing platforms. Major corporations, such as Google and Amazon, rely on Linux to manage their extensive server farms.

Economic Impact:

The open-source nature of Linux has economic implications, providing cost-effective solutions for businesses and individuals. Linux is freely available, eliminating the need for expensive licensing fees associated with proprietary operating systems. This cost advantage has contributed to its widespread adoption in various domains.

Role in Embedded Systems:
Linux's adaptability extends beyond traditional computing environments. Its presence in embedded systems, including smartphones, routers, and IoT devices, showcases its versatility. The ability to customize the Linux kernel for specific hardware requirements has made it a preferred choice in the embedded systems industry.

Challenges and Criticisms:
Despite its successes, Linux faces challenges. Fragmentation within the Linux ecosystem, arising from various distributions and divergent software packaging systems, can complicate software compatibility. Additionally, challenges related to driver support for certain hardware configurations have been raised.

Conclusion:
Linux's journey from a personal project to a global phenomenon reflects the power of open-source collaboration. Its impact on enterprise computing, cost-effectiveness, and adaptability in diverse environments underline its significance. As Linux continues to evolve, the case for open-source development and collaborative innovation becomes even more compelling in shaping the future of operating systems.

Case Study: Windows 11 - Transforming User Experience

Introduction:
Windows 11, Microsoft's latest operating system released in 2021, represents a significant leap forward in terms of user interface design, performance, and features. This case study delves into the key aspects of Windows 11, exploring its impact on user experience and the broader computing landscape.

User-Centric Design:
Windows 11 introduces a fresh and modern user interface, centered around simplicity and productivity. The Start menu is now centered, with Live Tiles replaced by static icons for a cleaner look. The redesigned taskbar, updated system icons, and consistent use of rounded corners contribute to a visually appealing and user-friendly environment.

Snap Layouts and Snap Groups:
A notable feature of Windows 11 is the enhanced multitasking experience through Snap Layouts and Snap Groups. Snap Layouts allow users to organize and snap multiple application windows into customizable layouts, facilitating efficient multitasking. Snap Groups ensure that users can easily switch between sets of applications, improving overall workflow management.

Microsoft Store Redesign:
Windows 11 includes a revamped Microsoft Store with a focus on performance, curation, and a wider range of applications. The store now supports a variety of app types, including Win32, Progressive Web Apps (PWAs), and Android apps. This

expansion of app compatibility enhances the user experience by providing a more extensive and diverse application ecosystem.

DirectX 12 Ultimate and Gaming Features:
Microsoft has prioritized gaming with Windows 11, introducing DirectX 12 Ultimate for enhanced graphics and gaming performance. The integration of Auto HDR (High Dynamic Range) and DirectStorage technology results in a more immersive gaming experience. The introduction of the Xbox app and the integration of Xbox Game Pass further reinforce Windows 11 as a gaming-centric platform.

Security Enhancements:
Windows 11 places a strong emphasis on security features, aiming to provide a secure computing environment. The inclusion of Windows Hello for faster and more secure logins, along with the integration of virtualization-based security, enhances system integrity. Additionally, the introduction of Windows Defender Antivirus and SmartScreen technology bolsters protection against malware and phishing attacks.

Compatibility and System Requirements:
While Windows 11 introduces a range of new features, it also brings changes to system requirements. The need for TPM 2.0 (Trusted Platform Module) and specific hardware configurations has sparked discussions about compatibility, with some users facing challenges upgrading their existing systems. Microsoft has addressed these concerns through updates and clarifications.

Continued Support for Productivity:
Windows 11 builds upon the productivity features of its predecessor, Windows 10, with enhancements such as the integration of Microsoft Teams directly into the taskbar. The focus on productivity extends to features like Widgets, which provide personalized information at a glance, and improvements in virtual desktop management.

Conclusion:
Windows 11 represents a significant evolution in the Windows operating system, emphasizing a modern and user-centric design, enhanced gaming capabilities, and robust security features. While introducing new elements, Microsoft continues to build on the core aspects that users value for productivity and efficiency. The case of Windows 11 highlights the dynamic nature of operating systems, adapting to user needs and technological advancements.

Generations of Operating System:

Certainly! The generations of operating systems represent different stages in the development and evolution of these crucial software components. Let's break down the generations in detailed way:

1. First Generation (1940s - 1950s):
Characteristics:
Hardware Focus: Mainly designed for specific hardware systems, often with limited functionality.

No User Interaction: Users interacted with computers using low-level programming languages and had minimal direct control over the system.
Batch Processing: Jobs were submitted in batches, and the entire job had to be completed before the next one could start.
No OS as We Know It: Operating systems in this generation were rudimentary and focused on managing the hardware resources efficiently.

2. Second Generation (1950s - mid 1960s):
Characteristics:
Introduction of Assembly Language: Assembly languages allowed for more straightforward programming compared to machine language.
Batch Processing Continued: Operating systems started to support more sophisticated batch processing, enabling the execution of multiple jobs without manual intervention.
Introduction of High-Level Languages: Fortran and COBOL emerged, making programming more accessible.
Job Scheduling: Operating systems began to incorporate job scheduling algorithms for improved resource utilization.

3. Third Generation (1960s - 1970s):
Characteristics:
Time-Sharing Systems: Introduced the concept of time-sharing, enabling multiple users to interact with the computer simultaneously.
Multiprogramming: Multiple programs could reside in the computer's memory simultaneously, improving overall system efficiency.
High-Level Languages Flourished: Languages like BASIC, PL/I, and C gained popularity, making programming more accessible.
Interactive User Interfaces: Users could interact with the system in a more user-friendly manner using terminals and monitors.

4. Fourth Generation (1970s - 1980s):
Characteristics:
Microprocessors and Personal Computers: The advent of microprocessors led to the development of personal computers.
Graphical User Interfaces (GUI): GUIs became prevalent, allowing users to interact with the system using visual elements like icons and windows.
Networking: Introduction of networking capabilities, facilitating communication between computers.
Distributed Operating Systems: Distributed systems emerged, enabling the coordination of tasks across multiple computers.

5. Fifth Generation (1980s - Present):
Characteristics:
Advancements in GUIs: GUIs continued to evolve, becoming more intuitive and user-friendly.
Multitasking and Multithreading: Operating systems supported running multiple tasks and threads simultaneously.
Networking Became Pervasive: The internet became widely accessible, leading to a focus on networking and communication.
Client-Server Architectures: Client-server models became prominent, with centralized servers providing services to connected clients.

Mobile and Cloud Computing: The rise of mobile devices and cloud computing further transformed the computing landscape.

6. Sixth Generation (Present - Future):

Characteristics:

AI Integration: Increasing integration of artificial intelligence and machine learning capabilities into operating systems.

Edge Computing: Focus on distributed computing at the edge of the network, reducing latency and improving performance.

Security and Privacy: Enhanced emphasis on security and privacy features to address the evolving threat landscape.

IoT Integration: Operating systems are adapting to support the Internet of Things (IoT) ecosystem.

Unit– 2: Process Management

Define process model:

A process model in operating systems refers to an abstract representation that describes how processes behave and interact with the system during their lifecycle.

Overview of the Process & threads:

1. Process Overview:-

Definition: A process is an independent, self-contained unit of execution in an operating system. It consists of the program code, data, and system resources required for its execution. Processes are managed by the operating system, and each process runs in its own address space, isolated from other processes.

Process States:-
New:The process is being created.

Ready:The process is ready to execute and waiting for CPU time.

Running:The process is currently being executed by the CPU.

Blocked (Waiting):The process is waiting for some event, such as I/O completion.

Terminated:The process has finished its execution.

Process Lifecycle:-
Creation:The process is created by the operating system or another process.

Ready:The process moves to the ready state, waiting for CPU time.

Running:The process is selected for execution and moves to the running state.

Blocked:The process moves to the blocked state if it needs to wait for an event (e.g., I/O operation).

Termination:The process finishes its execution and moves to the terminated state.

2. Threads Overview:-

Definition:A thread is a lightweight, smaller unit of a process. Multiple threads within the same process share the same resources, including the program code and data, but each thread has its own registers and stack. Threads allow for parallel execution within a process.

Types of Threads:-

- User-level Threads (ULTs):Managed entirely by the application and the thread library without kernel support.

Operating system is unaware of the existence of user-level threads.

- Kernel-level Threads (KLTs):Managed by the operating system kernel.

Provides more efficient parallelism as the kernel can schedule threads independently.

Thread States:-
Running: The thread is actively executing instructions.

Ready: The thread is ready to run but waiting for CPU time.

Blocked (Waiting): The thread is waiting for an event or resource.

Thread Lifecycle:-
Creation: A thread is created within a process.

Ready: The thread is ready to execute.

Running: The thread is actively executing.

Blocked: The thread is waiting for an event or resource.

Termination: The thread finishes its execution.

Process Life Cycle/ Process States:

The process life cycle, also known as process states, describes the different stages that a process goes through during its execution in an operating system. A process can exist in several states, and its state may change based on its interaction with the operating system. The typical process states include:

1. New:
Description: The process is in the initial stage, having been newly created by the operating system or another process.
Actions: Resources are allocated, and the process is initialized. It moves to the "Ready" state once it is ready for execution.
2. Ready:
Description: The process is prepared to run and is waiting for the CPU to be allocated to it.
Actions: The process is loaded into the main memory, and it is now eligible to be selected for execution.
3. Running:
Description: The process is currently being executed by the CPU.
Actions: The CPU executes the instructions of the process. The process remains in this state until it voluntarily relinquishes the CPU or is preempted by the operating system.
4. Blocked (Waiting):

Description: The process is unable to proceed because it is waiting for some event or resource, such as I/O completion or the availability of a semaphore.
Actions: The process is moved to this state when it is waiting for a particular event. It remains in this state until the event occurs.
5. Terminated:
Description: The process has completed its execution or has been terminated due to an error.
Actions: Resources associated with the process, including memory and open files, are released. The process is removed from the system.

Process State Transitions:
Creation (New to Ready): The process transitions from the "New" state to the "Ready" state when it has been initialized and is ready for execution.

Scheduling (Ready to Running): The operating system scheduler selects a process from the "Ready" queue to execute and moves it to the "Running" state.

I/O or Event Wait (Running to Blocked): If a process needs to wait for an event or resource, such as an I/O operation, it moves to the "Blocked" state.

Event Completion (Blocked to Ready): When the event or resource becomes available, the process transitions back to the "Ready" state.

Completion (Running to Terminated): A process moves from the "Running" state to the "Terminated" state when it completes its execution or encounters an error.

Process Synchronization:

In computer science, process synchronization refers to the coordination of activities among multiple processes to ensure their correct and orderly execution. The goal is to prevent conflicts and race conditions that may arise when multiple processes access shared resources concurrently. Synchronization mechanisms are essential for maintaining data consistency, avoiding deadlocks, and ensuring the correctness of concurrent programs.

One of the fundamental aspects of process synchronization is achieving mutual exclusion, which ensures that only one process can access a critical section of code at a time.
1. Critical Section:

A critical section is a part of the code in a program where shared resources or variables are accessed or modified. It is crucial to ensure that only one process can execute the critical section at any given time to avoid data corruption or inconsistent results.

Requirements for a Solution:

Mutual Exclusion:Only one process can be in the critical section at a time.

Progress:If no process is in the critical section and some processes want to enter it, then only those processes not in the remainder section can participate in deciding which will enter.

Bounded Waiting:There exists a bound on the number of times other processes are allowed to enter the critical section after a process has made a request to enter the critical section and before that request is granted.

2. Mutual Exclusion:

Mutual exclusion is a synchronization mechanism that ensures only one process or thread can access the critical section at any given time. The objective is to prevent concurrent access by multiple entities to shared resources that may lead to conflicts and inconsistencies.

Approaches to Achieve Mutual Exclusion:

Locks:Processes acquire a lock before entering a critical section. If the lock is already held by another process, the requesting process is blocked until the lock is released.

Semaphore:A semaphore is an integer variable used to control access to shared resources. It is initialized to the number of available resources. Processes increment the semaphore when entering the critical section and decrement it when leaving.

Mutex (Mutual Exclusion):A mutex is a synchronization primitive that ensures only one process can access a critical section at a time. It provides locking and unlocking mechanisms.

Atomic Operations:Certain hardware or software instructions guarantee atomicity, meaning they are executed as a single, uninterruptible unit. These operations are often used to implement mutual exclusion without additional synchronization mechanisms.

Deadlock:

A deadlock in computer science occurs when two or more processes are unable to proceed because each is waiting for the other to release a resource. In a deadlock, processes are stuck in a circular waiting state, and no progress is possible. Deadlocks can significantly impact the performance and reliability of a system.

Conditions for Deadlock:

Deadlocks arise from the simultaneous satisfaction of four necessary conditions, often known as the Coffman conditions:

Mutual Exclusion:
At least one resource must be held in a non-sharable mode, meaning only one process at a time can use the resource.

Hold and Wait:
A process must be holding at least one resource and waiting to acquire additional resources that are currently held by other processes.

No Preemption:
Resources cannot be forcibly taken away from a process; they must be released voluntarily by the process holding them.

Circular Wait:
There must exist a circular chain of two or more processes, each waiting for a resource held by the next process in the chain.

Unit– 3: Memory Management

Describe memory management:

Memory management is a crucial aspect of operating systems, responsible for handling the allocation and deallocation of memory resources to processes or programs. It ensures efficient utilization of the available memory, preventing conflicts and providing a structured approach for storing and accessing data. The primary goals of memory management include maximizing system performance, minimizing fragmentation, and ensuring the secure and organized use of memory resources.

Logical and Physical Address Mapping:

Logical Address:

Definition:A logical address, also known as a virtual address, is generated by the CPU during the execution of a program. It represents a location in the logical address space of a process.

Usage:Logical addresses are used by programs and are relative to the base address of the process's logical address space.

Responsibility:The Memory Management Unit (MMU) in the CPU is responsible for translating logical addresses to physical addresses during program execution.

Role in Virtual Memory:Logical addresses play a crucial role in virtual memory systems, allowing each process to have its own dedicated memory space.

Physical Address:

Definition:A physical address represents an actual location in the computer's physical memory (RAM).

Usage:Physical addresses are used by the computer's memory management hardware to access data in the RAM.

Responsibility:The operating system, in collaboration with the MMU, is responsible for mapping logical addresses to physical addresses.

Role in Virtual Memory:Physical addresses represent actual locations in the RAM, and the operating system manages the mapping between logical and physical addresses.

Logical-to-Physical Address Mapping:

Page Tables: In virtual memory systems, logical addresses are often mapped to physical addresses using page tables. The page table maintains a mapping between logical page numbers and corresponding physical page frames.

Segmentation:In systems that use segmentation, logical addresses consist of a segment number and an offset within that segment. The operating system maintains a mapping between logical segment numbers and corresponding physical base addresses.

MMU Translation:The MMU translates logical addresses to physical addresses during program execution. It uses page tables or other mapping structures for this translation.

Swapping:

Definition: Swapping is a memory management technique used to move a process or parts of a process from the main memory (RAM) to secondary storage (usually the hard disk) and vice versa.

Purpose: Swapping allows the operating system to free up space in RAM by temporarily moving less frequently used processes or pages to disk. It helps in efficient utilization of memory resources.
Steps in Swapping:

Selection: The operating system selects a process or a page to be swapped out of the main memory.

Transfer: The selected process or page is transferred from the main memory to a designated space on the secondary storage (swap space).

Bring-In: If needed later, the process or page can be swapped back into the main memory. This is also known as bringing in or paging in.

Swapping Policies: Different swapping policies determine when and which processes or pages should be swapped. Common policies include Least Recently Used (LRU), First-In-First-Out (FIFO), and Clock algorithms.

Advantages of Swapping:

Increased Utilization: Swapping enables the system to run more processes than can fit into the physical memory by temporarily storing less active processes on disk.

Multi-programming Support: Allows for efficient multi-programming, where multiple processes can be kept in the main memory simultaneously.

Fair Resource Allocation: Ensures fair allocation of resources among processes by swapping out less active processes.

Disadvantages of Swapping:

Performance Overhead: Swapping introduces overhead due to the time required to transfer data between the main memory and secondary storage.

Disk I/O Bottleneck: Frequent swapping can lead to increased disk I/O, creating a bottleneck for system performance.

Increased Latency: Swapping can introduce latency when bringing processes back into the main memory, affecting overall system responsiveness.

Difference between Contiguous & Non- contiguous memory allocation:

S.NO.	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method.	Only External fragmentation occurs in Non-Contiguous memory allocation method.
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.
8.	In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space.	In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.
9.	It is of two types: 1. Fixed(or static) partitioning 2. Dynamic partitioning	It is of five types: 1. Paging 2. Multilevel Paging 3. Inverted Paging 4. Segmentation 5. Segmented Paging

S.NO.	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
10.	It could be visualized and implemented using Arrays.	It could be implemented using Linked Lists.
11.	Degree of multiprogramming is fixed as fixed partitions	Degree of multiprogramming is not fixed

Differentiate primary and secondary memory:

Primary memory	Secondary memory
Primary memory is temporary.	Secondary memory is permanent.
Primary memory is directly accessible by Processor/CPU.	Secondary memory is not directly accessible by the CPU.
Nature of Parts of Primary memory varies, RAM- volatile in nature. ROM- Non-volatile.	It's always Non-volatile in nature.
Primary memory devices are more expensive than secondary storage devices.	Secondary memory devices are less expensive when compared to primary memory devices.
The memory devices used for primary memory are semiconductor memories.	The secondary memory devices are magnetic and optical memories.
Primary memory is also known as Main memory or Internal memory.	Secondary memory is also known as External memory or Auxiliary memory.
Examples: RAM, ROM, Cache memory, PROM, EPROM, Registers, etc.	Examples: Hard Disk, Floppy Disk, Magnetic Tapes, etc.

Memory Allocation:

1. Contiguous memory allocation:-

Contiguous memory allocation strategies involve dividing memory into partitions and allocating entire blocks to processes. The choice of strategy depends on factors like process size, memory utilization efficiency, and the need for flexibility. Techniques like compaction and proper allocation policies help mitigate fragmentation issues. Memory relocation and protection mechanisms ensure the secure and efficient execution of processes. Allocation techniques like First Fit, Best Fit, and Worst Fit aim to optimize memory usage based on different criteria.

i. Fixed and Variable Partition:

Fixed Partition:

Definition: Memory is divided into fixed-size partitions, and each partition is assigned to a process.

Allocation: Processes are allocated entire partitions based on their size.

Advantages:
- Simple and straightforward.
- Reduces external fragmentation.

Disadvantages:
-Inflexible, leading to inefficient memory usage.
-May result in underutilization of memory.

Variable Partition:

Definition: Memory is divided into variable-sized partitions based on the size of processes.

Allocation: Processes are allocated partitions according to their actual size.

Advantages:
-Efficient utilization of memory.
-More flexible than fixed partitioning.

Disadvantages:
-Can lead to external fragmentation.
-Requires dynamic memory management.

ii. Internal and External Fragmentation and Compaction:

Internal Fragmentation:

Definition: Wasted memory within a partition due to the allocation of larger memory blocks than required by a process.

Addressing:Addressed by proper allocation strategies to minimize unused space.

Mitigation:Can be reduced by using variable partitioning and intelligent allocation policies.

External Fragmentation:

Definition: Unallocated memory scattered in small, non-contiguous blocks, making it challenging to allocate contiguous memory to larger processes.

Addressing: Compaction or paging strategies can help address external fragmentation.

Mitigation: Compaction involves shifting processes to consolidate free memory blocks, reducing external fragmentation.

iii. Memory Relocation and Protection Mechanism:

Memory Relocation:

Definition:Involves moving a program to a different location in memory during its execution.

Purpose:Allows the program to execute as if it starts at a specific fixed address, even if loaded at a different address.

Mechanism:The operating system adjusts the program's addresses during loading and execution.

Protection Mechanism:

Definition:Ensures that processes cannot access or modify each other's memory space.

Implementation:Achieved through hardware mechanisms like Memory Management Unit (MMU) and operating system controls.

Purpose:Prevents unauthorized access, ensuring data integrity and system stability.

iv. Allocation Techniques – First Fit, Best Fit, and Worst Fit:
First Fit:

Algorithm:
Allocates the first available memory block that is large enough to accommodate the process.

Advantages:
- Simple and quick.
- Efficient for average-sized processes.
- Disadvantages:
- Can lead to external fragmentation.
- Best Fit:

Algorithm: Allocates the smallest available memory block that fits the process.

Advantages:
- Reduces external fragmentation.
- Efficient for small-sized processes.

Disadvantages:
- May result in underutilization of memory.

Worst Fit:

Algorithm:
Allocates the largest available memory block for the process.

Advantages:
-Reduces the number of small holes in memory.

Disadvantages:
-Can lead to more significant external fragmentation.
-Inefficient for small to medium-sized processes.

2. Non Contiguous Memory allocation:-

Non-contiguous memory allocation, through paging and segmentation, allows for more efficient utilization of memory by allocating processes in smaller, non-sequential units. Address translation is handled through page tables or segment tables. Page replacement algorithms like FIFO and LRU are employed to manage the contents of memory efficiently. These techniques are commonly used in modern operating systems, providing flexibility and improved memory utilization compared to contiguous memory allocation.

i. Overview of Paging:

Definition:Paging is a non-contiguous memory allocation technique that divides both physical and logical memory into fixed-size blocks called pages.

Page Size:Memory is divided into fixed-sized pages, and processes are divided into corresponding pages.

Advantages:
- Efficient use of memory, as processes can be allocated non-contiguously.
- Reduces external fragmentation compared to contiguous allocation.

Addressing: Logical addresses are divided into a page number and an offset within the page.

Mapping: Page tables are used for mapping logical page numbers to physical page frames.

ii. Address Translation using Basic Method of Paging:

Address Structure:
Logical Address = (Page Number, Offset within Page)
Physical Address = (Frame Number, Offset within Frame)

Page Table: Page tables store the mapping between logical pages and corresponding physical frames.

Address Translation: The logical address's page number is used as an index in the page table to retrieve the corresponding frame number.
The offset within the page remains unchanged.

Example:

If the logical address is (5, 300), the page table is consulted to find that logical page 5 maps to physical frame 8. The resulting physical address is (8, 300).

iii. Overview of Segmentation:

Definition: Segmentation is another non-contiguous memory allocation technique that divides a process into segments, where each segment represents a logical unit of the program.

Segments: Segments include code, data, stack, etc., and each segment has its own size and attributes.

Advantages:

- Supports dynamic memory allocation, allowing for flexibility in program structure.
- Easier to implement protection mechanisms for each segment.

Addressing: Logical addresses consist of a segment number and an offset within the segment.

Mapping: Segment tables map logical segment numbers to corresponding base addresses in physical memory.

iv. Page Replacement Algorithm – FIFO, LRU:

FIFO (First-In-First-Out):

Algorithm:Pages are replaced in the order they entered the memory.

Implementation:Uses a queue to keep track of the order of page entries.

Advantages:
- Simple and easy to implement.

Disadvantages:
-May not always be the most efficient, especially if older pages are still actively used.

LRU (Least Recently Used):

Algorithm: Replaces the page that has not been used for the longest time.

Implementation:Requires maintaining a list or counter to track the usage history of each page.

Advantages:
- Tends to retain actively used pages, improving overall performance.

Disadvantages:
-More complex to implement than FIFO.

Unit–4: File Management System

File System:

A file system is a method used by computers and operating systems to organize and store data on storage devices, such as hard drives, solid-state drives, or external storage. It provides a structured way to manage, access, and manipulate data stored on these devices.

1. File Attributes:
File attributes are properties associated with files that provide information about their characteristics. Common file attributes include:

Name: Unique identifier for the file within a directory.

Type: Specifies the file's format or type (e.g., text, image, executable).

Size: Indicates the size of the file in bytes or another unit.

Location: Physical or logical location where the file is stored on the storage device.

Permissions: Define who can read, write, or execute the file, controlling access rights.

Timestamps: Record when the file was created, last modified, or last accessed.

Owner: Identifies the user or group that owns the file.

Attributes: Additional properties, such as whether the file is hidden or read-only.

3. File Operations:
File operations refer to the actions that can be performed on files within a file system. Common file operations include:

Create: Generates a new file with a specified name and attributes.

Read: Retrieves data from a file.

Write: Stores or updates data in a file.

Delete: Removes a file from the file system.

Open: Initiates access to a file for reading, writing, or both.

Close: Terminates access to a file.

Rename: Changes the name of a file.

Copy: Duplicates a file, creating a new file with the same or different name.

Move: Relocates a file within the same file system or to another location.

3. File Types:
Files in a file system can be categorized into various types based on their content and purpose. Common file types include:

Text Files:Contain human-readable characters and are often used for storing plain text.

Binary Files:Contain non-textual data, such as images, executables, or multimedia files.

Executable Files:Contain machine code or scripts that can be executed by the computer.

Compressed Files:Store data in a compressed format to reduce file size.

Directories/Folders:Contain lists of file names and their associated attributes, providing a hierarchical structure.

Special Files:Represent devices or system-related entities (e.g., character devices, block devices).

Link Files:Create references to other files, allowing them to be accessed using multiple names.

Directory System:

A directory system is a crucial component of a file system that organizes and manages files in a hierarchical structure. It provides a way to group related files together, facilitating efficient organization and retrieval of data. The directory system is responsible for maintaining the structure of directories, managing file metadata, and ensuring access control.

Directory Structures:

Hierarchical Directory Structure:

Definition: Organizes directories and files in a tree-like structure, with a single root directory and subdirectories branching from it.

Advantages:
- Provides a logical and intuitive way to organize files.
- Facilitates easy navigation and location of files.

Flat Directory Structure:

Definition:All files are stored in a single directory without any subdirectories.

Advantages:
- Simplicity in organization.
- Suitable for small-scale systems.

Two-Level Directory Structure:

Definition: Introduces a separate user directory and a master directory.

Advantages:
- Allows users to have their own directories for organization.
- Simplifies access control.

Tree-Structured Directory:

Definition: Each subdirectory can have multiple subdirectories, forming a tree structure.

Advantages:
- Supports a hierarchical organization with greater depth.

Acyclic-Graph Directory Structure:

Definition: Permits directories to have multiple parent directories, creating a directed acyclic graph.

Advantages:
-Allows for more flexible organization.

Protection:

Access Control:

Definition: Determines who can perform specific operations on files or directories.

Mechanism: Access control lists (ACLs) or permissions associated with user roles.

File Permissions:

Read: Allows reading the content of a file.

Write: Permits modification or deletion of a file.

Execute: Enables the execution of a file (for executable files or scripts).

Directory Permissions:

Search/Execute: Allows accessing the contents of a directory.

Write: Permits the creation, deletion, or renaming of files within the directory.

Allocation Methods – Contiguous:

Contiguous Allocation:

Definition: Allocates a contiguous block of disk space to a file.

Advantages:
- Simple and efficient for sequential access.
- Reduces disk fragmentation.

Disadvantages:
- Fragmentation may occur with variable-sized files.
- Difficult to allocate space dynamically.

Allocation Methods - Non-Contiguous:

Linked Allocation:

Definition: Allocates each file block individually and maintains pointers to the next block.

Advantages:
- Efficient for variable-sized files.

Disadvantages:
-Sequential access can be slower due to scattered blocks.

Indexed Allocation:

Definition: Uses an index block to store pointers to all blocks of a file.

Advantages:
- Efficient for both sequential and direct access.

Disadvantages:
- Additional overhead of maintaining an index.

Multi-level Indexed Allocation:

Definition: Extends the indexed allocation method by introducing multiple levels of index blocks.

Advantages:
- Efficient for large files.

Disadvantages:
- Increased complexity.