Operating System (OS)

An **Operating System (OS)** is a collection of software that manages computer hardware resources and provides common services for computer programs.

An operating system acts as an interface between the software and different parts of the computer or the computer hardware.

It controls and monitors the execution of all other programs that reside in the computer, which also includes application programs and other system software of the computer.

Commonly Used OS: Windows, UNIX, LINUX, BOSS, SOLARIS

Functions of OS

1. Memory Management:

- An Operating system manages the allocation and deallocation of memory to various processes and ensures that the other process does not consume the memory allocated to one process.
- It allocates the memory to a process when the process requests and deallocates the memory when the process has terminated or is performing an I/O operation.

2. Processor Management:

- In a multi-programming environment, the OS decides the order in which the processes should access the processor, and how much time each process has, this is also called **Process Scheduling**.

3. Device Management:

- An OS manages device communication via its respective drivers.
- It keeps track of all the connected devices, receives requests from these devices, performs a specific task, and communicates back.

4. User Interface:

- The user interacts with the computer system through the OS. Hence it also acts as an interface between the user and the computer hardware.
- This interface which is used by users to interact with the applications and machine hardware is offered through a set of commands or a GUI (Graphical User Interface).

5. Security:

- The Operating System uses password protection and other similar techniques to protect user data.
- It also prevents unauthorized access to programs and user data.

6. Job Accounting:

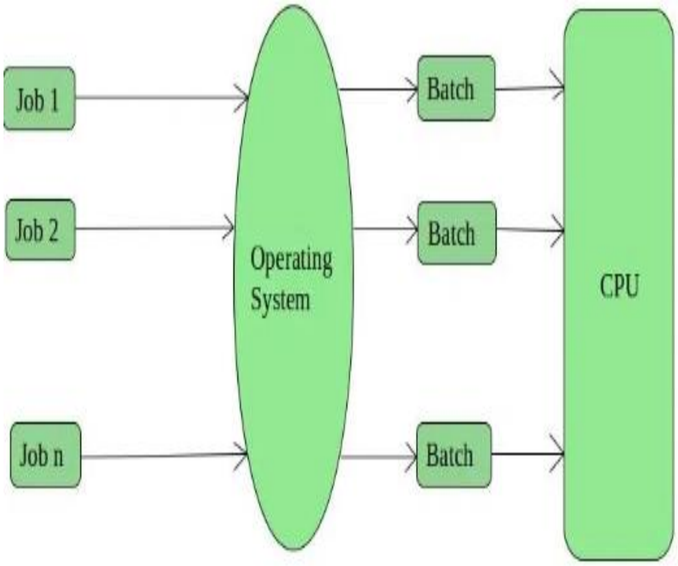
- The operating system Keeps track of time and resources used by various tasks and using this information the OS decides the order of applications running and how much time should be allocated.

Types of OS

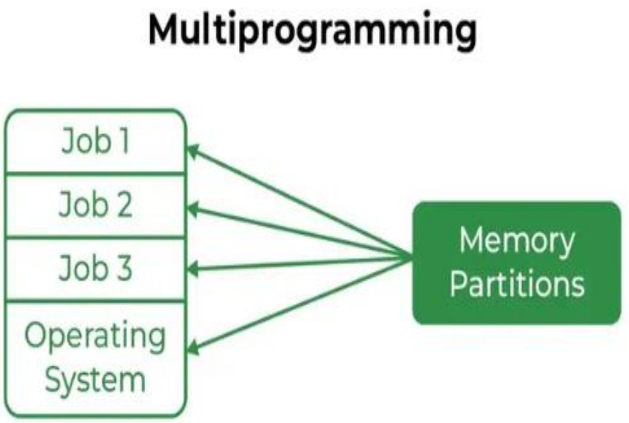
1.Batch Operating System:

This type of operating system does not interact with the computer directly.

There is an operator which takes similar jobs having the same requirement and groups them into batches. the operator to sort jobs with similar needs.



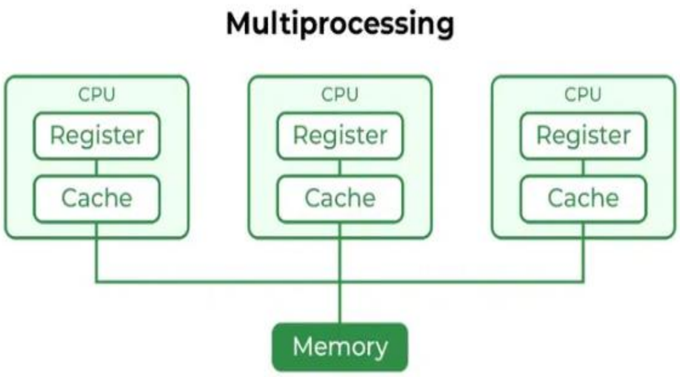
2. Multi-Programming OS



Multiprogramming Operating Systems can be simply illustrated as more than one program is present in the main memory and any one of them can be kept in execution.

This is basically used for better execution of resources.

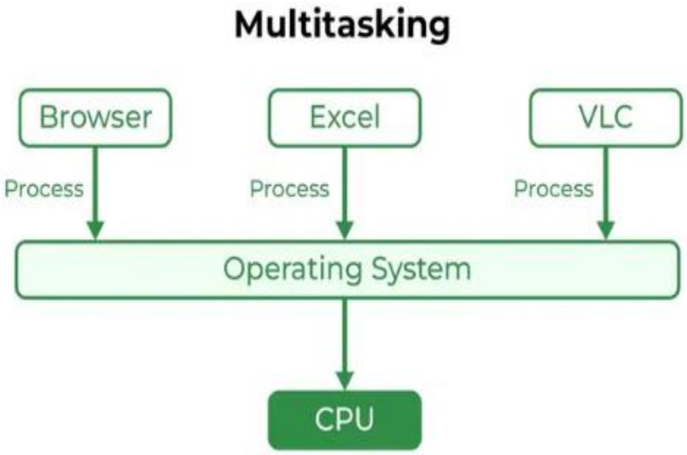
3. Multi-Processing OS



Multi-Processing Operating System is a type of Operating System in which more than one CPU is used for the execution of resources.

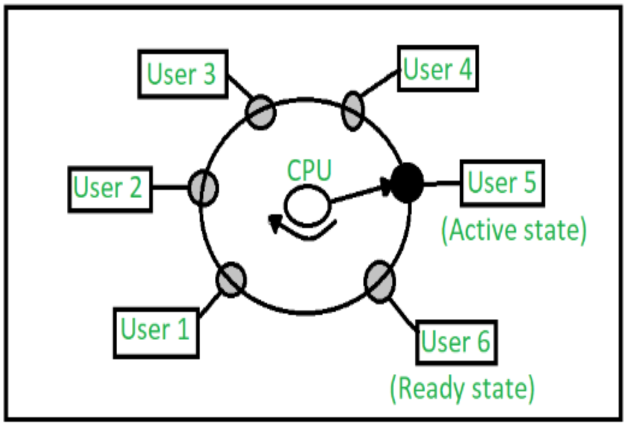
It betters the throughput of the System.

4. Multi-Tasking OS



Multitasking Operating System is simply a multiprogramming Operating System with having facility of a Round-Robin Scheduling Algorithm.

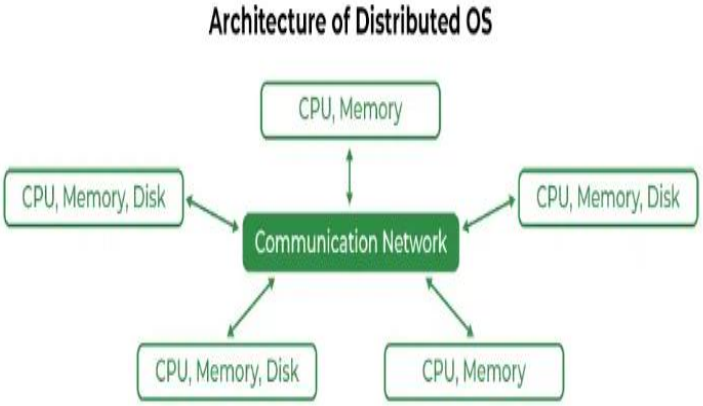
It can run multiple programs simultaneously.



5. Time Sharing OS:

It is the OS in which each task is given some time to execute so that all the tasks work smoothly.

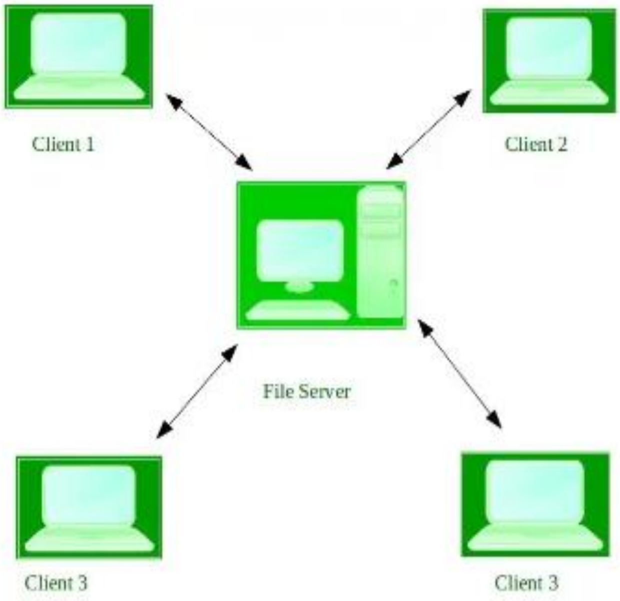
The time that each task gets to be executed is called quantum. After this time interval is over OS switches over to the next task.



6. Distributed OS:

In this type of OS various autonomous interconnected computers communicate with each other using a shared network.

These are referred to as **loosely coupled systems**.

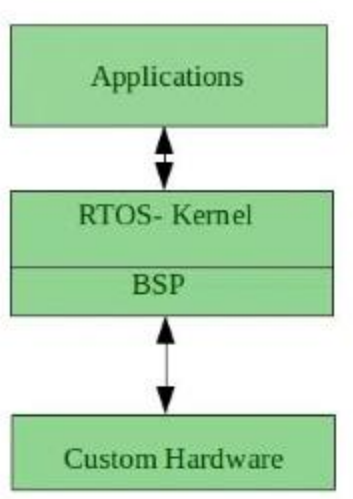


7. Networking OS:

These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions.

These are referred to as **tightly coupled systems**.

8. Real-Time Operating System (RTOS):



Real-Time Operating System

These types of OSs serve real-time systems. The time interval required to process and respond to inputs is very small.

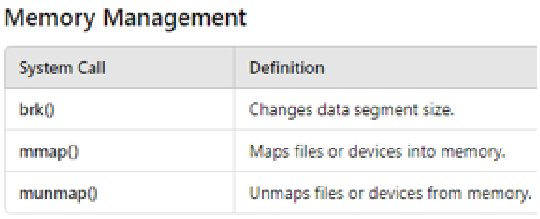
Real-time systems are used when there are time requirements that are very strict like missile systems, air traffic control systems, robots, etc.

Types of RTOS: Hard RTOS, Soft RTOS

System Call

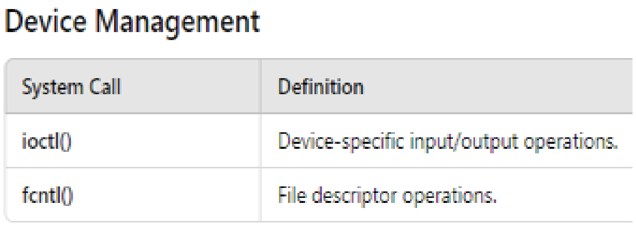
- It is a programmatic way in which a program requests a service from the kernel (central component) of OS.
- System call provides the services of the operating system to the user programs via Application Program Interface (API).
- A system call is initiated by the program executing a specific instruction, which triggers a switch to kernel mode, allowing the program to request a service from the OS. The OS then handles the request, performs the necessary operations, and returns the result back to the program.

Examples of System Call:



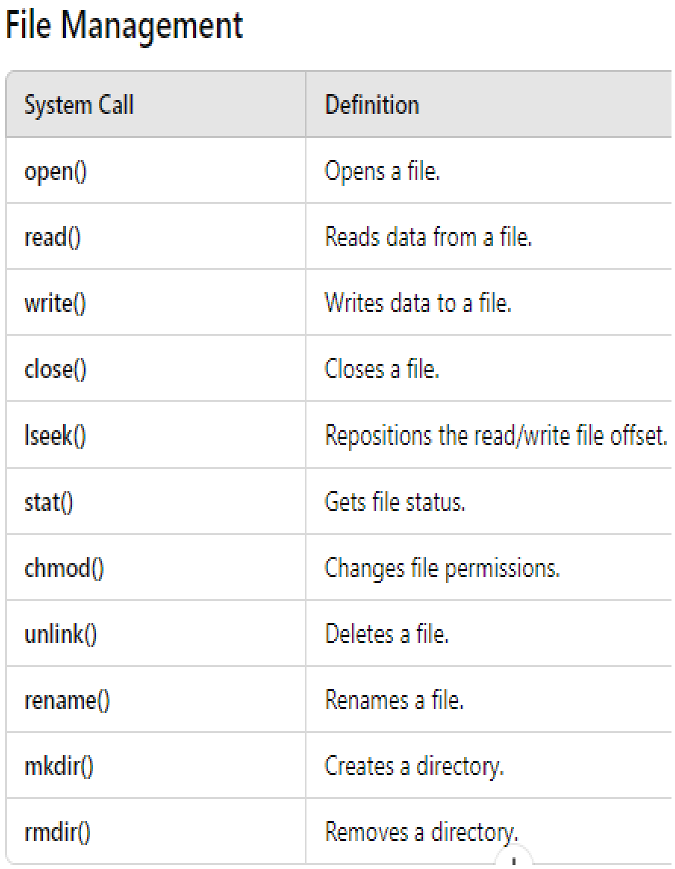
Memory Management

System Call	Definition
brk()	Changes data segment size.
mmap()	Maps files or devices into memory.
munmap()	Unmaps files or devices from memory.



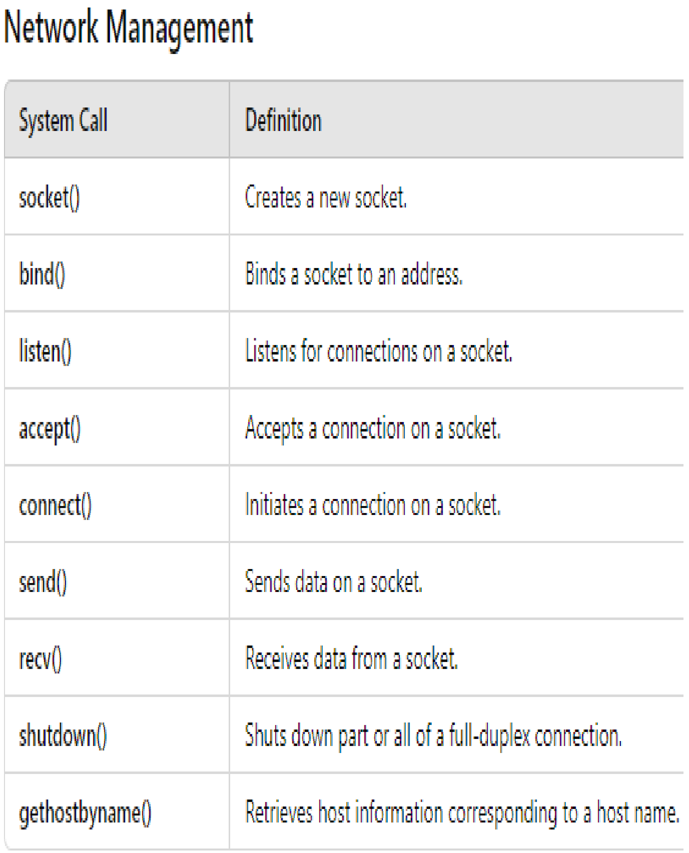
Device Management

System Call	Definition
ioctl()	Device-specific input/output operations.
fcntl()	File descriptor operations.



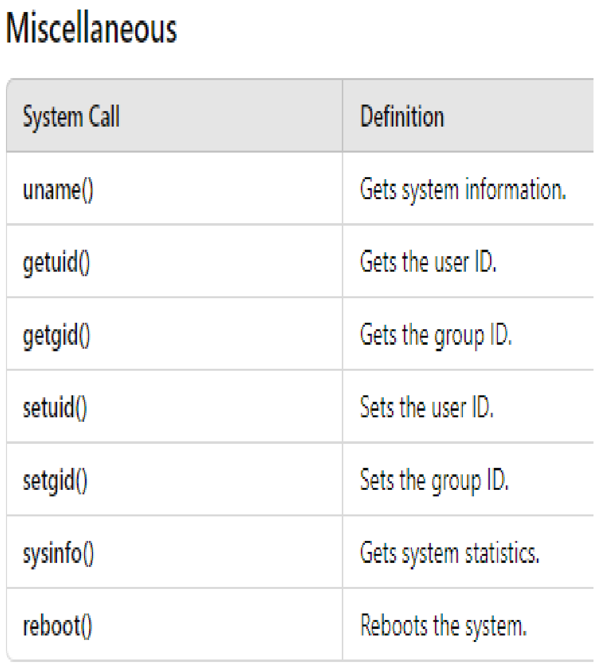
File Management

System Call	Definition
open()	Opens a file.
read()	Reads data from a file.
write()	Writes data to a file.
close()	Closes a file.
lseek()	Repositions the read/write file offset.
stat()	Gets file status.
chmod()	Changes file permissions.
unlink()	Deletes a file.
rename()	Renames a file.
mkdir()	Creates a directory.
rmdir()	Removes a directory.



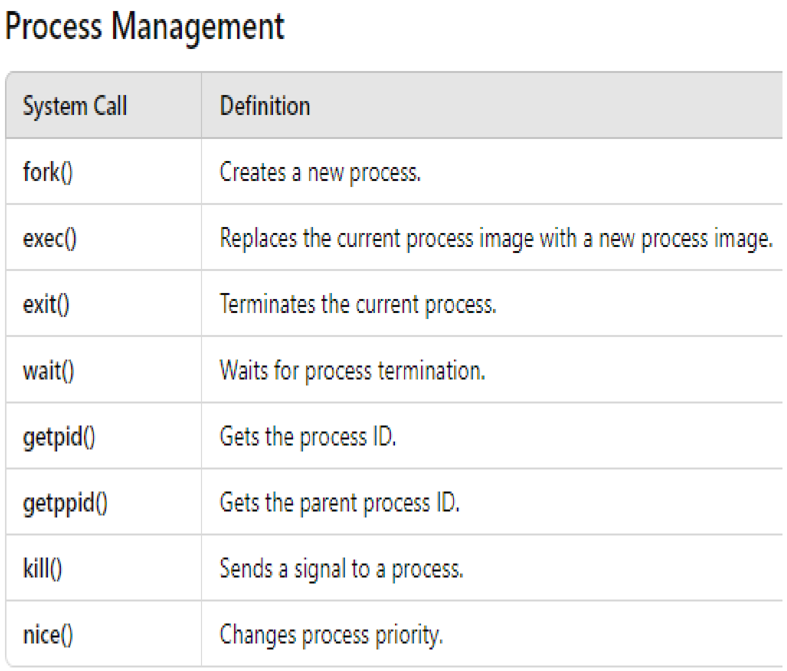
Network Management

System Call	Definition
socket()	Creates a new socket.
bind()	Binds a socket to an address.
listen()	Listens for connections on a socket.
accept()	Accepts a connection on a socket.
connect()	Initiates a connection on a socket.
send()	Sends data on a socket.
recv()	Receives data from a socket.
shutdown()	Shuts down part or all of a full-duplex connection.
gethostbyname()	Retrieves host information corresponding to a host name.



Miscellaneous

System Call	Definition
uname()	Gets system information.
getuid()	Gets the user ID.
getgid()	Gets the group ID.
setuid()	Sets the user ID.
setgid()	Sets the group ID.
sysinfo()	Gets system statistics.
reboot()	Reboots the system.



Process Management

System Call	Definition
fork()	Creates a new process.
exec()	Replaces the current process image with a new process image.
exit()	Terminates the current process.
wait()	Waits for process termination.
getpid()	Gets the process ID.
getppid()	Gets the parent process ID.
kill()	Sends a signal to a process.
nice()	Changes process priority.

Process

- A **process** is a program in execution.
- **For example**, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we run the binary code, it becomes a **process**.
- The OS is responsible for managing the start, stop, and scheduling of processes, which are basically programs running on the system and this management of processes is called **process management**.
- Every process goes through different states throughout the life cycle during the process execution, which is known as **process states**.
- Processes generally has following **7 states**:
    a. **New State**: This is the first state of the process life cycle. When creation of process is taking place; the process is in a **new state**.
    b. **Ready State**: When the process creation is completed, the process comes into a **ready state**. During this state, the process is loaded into the main memory and will be placed in the queue of processes which are waiting for the CPU allocation. When the process is in the creation process is in a new state and when the process gets created process is in the ready state.
    c. **Running State**: Whenever the CPU is allocated to the process from the ready queue, the process state changes to **Running**.
    d. **Block or Wait State**: When the process is executing the instructions, the process might require carrying out a few tasks that might not require CPU. If the process requires performing Input-Output task or the process needs some resources that are already acquired by other processes, during such conditions process is brought back into the main memory, and the state is changed to **Blocking or Waiting for the state**. Process is placed in the queue of processes that are in waiting or block state in the main memory.
    e. **Terminated or Completed**: When the entire set of instructions is executed, and the process is completed. The process is changed to a **terminated or completed state**. During this state the **PCB** of the process is also deleted.

f. **Suspend Ready**: Whenever the main memory is full, the process which is in a ready state is swapped out from main memory to secondary memory. The process is in a ready state when goes through the transition of moving from main memory to secondary memory, the state of that process is changed **to Suspend Ready State**. Once the main memory has enough space for the process, the process will be brought back to the main memory and will be in a ready state.

g. **Suspend Wait or Suspend Blocked**: Whenever the process that is in waiting for state or block state in main memory gets to swap out to secondary memory due to main memory being completely full, the process state is changed to **Suspend wait or Suspend blocked state.**

Inter Processor Communication (IPC):

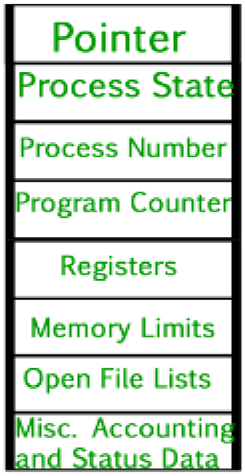
IPC in OS is a way by which multiple processes can communicate with each other. Shared memory in OS, message queues, FIFO and so on are some of the ways to achieve **IPC** in OS.

IPC provides a mechanism to exchange data and information across multiple processes, which might be on single or multiple computers connected by a network.

Process Control Block (PCB):

A **Process Control Block (PCB)** is a data structure used by the operating system to store all the information about a process. It is essential for process management and allows the OS to keep track of the state of processes.

When the process makes a transition from one state to another, the operating system must update information in the process's PCB.



Pointer

Process State

Process Number

Program Counter

Registers

Memory Limits

Open File Lists

Misc. Accounting and Status Data

Process Control Block

Process Scheduling:

Process scheduling is the method by which an operating system decides which processes run at any given time. It aims to allocate CPU time efficiently to ensure smooth and efficient execution of processes.

Process scheduling in an operating system is crucial for managing CPU resources and ensuring efficient execution of processes.

Process scheduling involves using various algorithms to decide the order and timing of process execution, aiming to optimize performance metrics like CPU utilization, throughput, turnaround time, waiting time, and response time

Process scheduling can be categorized into two types based on whether a process can be interrupted while it's running: **preemptive and non-preemptive scheduling.**

Preemptive Scheduling:

In preemptive scheduling, the operating system can interrupt a running process and switch the CPU to another process. This ensures that the system can respond quickly to important tasks and maintain a balanced workload. Preemptive scheduling is commonly used in time-sharing and real-time systems.

Key Features:

Interruptions: The running process can be interrupted and moved to the ready queue.

Fairness: Ensures all processes get a fair share of the CPU.

Responsiveness: Better for interactive systems as it can quickly respond to user inputs.

Common Preemptive Scheduling Algorithms:

Round Robin (RR):

- Each process gets a fixed time slice (quantum).
- Processes are rotated in a circular queue.

Preemptive Priority Scheduling:

- Processes are assigned priorities.
- Higher priority processes preempt lower priority ones.

Shortest Remaining Time First (SRTF):

- The process with the shortest remaining execution time is selected next.

Multilevel Feedback Queue:

- Processes can move between different priority queues based on their behavior and age.

Non-Preemptive Scheduling

In non-preemptive scheduling, once a process starts executing, it runs to completion or until it voluntarily releases the CPU (e.g., it goes into the waiting state). Non-preemptive scheduling is simpler to implement but can lead to inefficiencies and poor responsiveness.

Key Features:

No Interruptions: The process runs until it finishes or voluntarily yields the CPU.

Simplicity: Easier to implement and manage.

Risk of Starvation: Long processes can cause delays for others.

Common Non-Preemptive Scheduling Algorithms:

First-Come, First-Served (FCFS):

- Processes are scheduled in the order they arrive.
- Simple but can lead to long wait times (convoy effect).

Non-Preemptive Priority Scheduling:

- Processes are assigned priorities.
- Higher priority processes are selected first, but running processes are not preempted.

Shortest Job First (SJF):

- The process with the shortest burst time is selected next.
- The process with the highest priority runs next.

Context Switching

- **Context switching** is the process by which an operating system saves the state of a currently running process or thread and restores the state of a different process or thread. This allows the CPU to switch from executing one process to executing another, enabling multitasking.
- Context switching enables all processes to share a single CPU to finish their execution and store the status of the system's tasks.
- The **context** of a process consists of its stack space, address space, virtual address space, register set image (e.g. Program Counter (PC), Instruction Register (IR), Program Status Word (PSW) and other general processor registers), Stack Pointer (SP).
- A **Context switch time** is a time which is spent between two processes (i.e., getting a waiting process for execution and sending an executing process to a waiting state).

Threads

- **Threads** can be defined as **lightweight subprocesses** as they possess some of the properties of processes. Threads run in parallel improving the application performance.
- Each thread belongs to exactly one process and a single process can have multiple threads. Each such thread has its own **stack space, register set, program counter,** but they share the address space of the process and the environment.
- Threads can share common data, so they do not need to use **inter-process communication**. Like the processes, threads also have **states** like **new, runnable, blocked, terminated**.
- Each thread has its own **Thread Control Block (TCB)**. Like the process, a context switch occurs for the thread, and register contents are saved in (TCB).

Types of Threads:

1. **User Level Threads:**
   - These threads are implemented and managed by users; the kernel has no work in the management of user level threads.
   - User Level Threads are more efficient than kernel level threads as implementation and context switching is easy.
2. **Kernel Level Threads:**
   - Kernel level threads are implemented and managed by the operating system kernel. The kernel is aware of each thread and is responsible for scheduling and managing them.
   - Kernel level threads can be scheduled on different processors in a multiprocessor system, leading to true parallelism.

Multithreading

Multithreading is a programming and execution model that allows multiple threads to be created within a single process. Each thread runs independently but shares the same memory space. Multithreading enables parallelism, improves performance, and enhances the responsiveness of applications.

Synchronization

- **Synchronization** ensures that multiple processes or threads can operate concurrently without interfering with each other.
- To achieve **Synchronization** various mechanisms such as **Locks/mutexes, semaphores, Mointors** are used.
- Based on synchronization, processes are categorized into two categories:
  1. **Independent Process**: The execution of one process does not affect the execution of other processes.
  2. **Cooperative Process**: A process that can affect or be affected by other processes executed in the system. (Synchronization required)

Mechanisms for Process Synchronization:

1. Locks (Mutexes):

A lock or mutex (short for mutual exclusion) is a mechanism to ensure that only one thread or process can access a resource at a time.

How They Work:

Acquiring the Lock: Before a thread enters the **critical section** (the part of the code that accesses shared resources), it must acquire the lock.

Releasing the Lock: After the thread finishes using the shared resource, it releases the lock.

Example: If Thread A has the lock, Thread B must wait until Thread A releases it before it can proceed.

2. Semaphores:

Semaphores are counters used to control access to shared resources.

Types: Binary Semaphores, Counting Semaphores.

Binary Semaphores: These act like simple locks and can have only two values: 0 (locked) and 1 (unlocked).

Counting Semaphores: These can have values greater than one and are used to manage a finite number of resources.

How They Work:

Wait (P operation): Decreases the semaphore value. If the value is already zero, the process must wait.

Signal (V operation): Increases the semaphore value, potentially waking up a waiting process.

Example: For a counting semaphore initialized to 3 (indicating 3 resources), each wait operation decreases the count, and each signal operation increases it. If the count reaches zero, any further wait operations will block until a signal operation occurs.

3. Monitors:

Monitors are high-level synchronization constructs that provide a convenient way to manage shared resources.

How They Work:

Encapsulation: Monitors encapsulate both the shared resources and the procedures to access them.

Mutual Exclusion: Only one thread can execute a monitor procedure at a time.

Condition Variables: Monitors often use condition variables to handle situations where a thread must wait for some condition to be true.

Example: In a Java program, you can use the synchronized keyword to define a monitor. A method marked as synchronized ensures that only one thread can execute it at a time.

Problems in Synchronization:

- ❖ **Deadlock**
- ❖ **Starvation**
- ❖ **Race Condition**

Deadlock

Deadlock is a situation in a multi-threaded or multi-process system where two or more processes are unable to proceed because each is waiting for one of the others to release a resource.

Deadlocks can cause programs to freeze and can be challenging to detect and resolve.

Conditions for Deadlock

A deadlock situation can occur if the **following four conditions** hold simultaneously:

1. **Mutual Exclusion**: At least one resource must be held in a non-shareable mode; only one process can use the resource at any given time.
2. **Hold and Wait**: A process holding at least one resource is waiting to acquire additional resources that are currently being held by other processes.
3. **No Preemption**: Resources cannot be forcibly taken from a process holding them; they must be released voluntarily by the process.
4. **Circular Wait**: A set of processes are waiting for each other in a circular chain, where each process holds at least one resource needed by the next process in the chain.

Deadlock Detection

Deadlock Detection is the process of finding out whether any process is stuck in a loop or not.

Deadlock detection involves monitoring the system to identify when a deadlock has occurred. This typically involves the use of algorithms that check for cycles in resource allocation graphs or wait-for graphs.

Algorithms for Deadlock Detection

1. **Resource Allocation Graph** 2. **Banker's Algorithm**

Resource Allocation Graph: This method involves periodically checking the resource allocation graph for cycles, which indicates the presence of deadlock.

Banker's Algorithm: This method checks for safe states by simulating resource allocation for processes and determining if the system can allocate resources to each process without leading to deadlock.

Race condition

Race condition occurs when multiple threads read and write the same variable i.e. they have access to some shared data and they try to change it at the same time. In such scenario threads are “racing” each other to access/change the data. Simple solution for race condition is to use Mutual Exclusion with locks.

Critical section

The critical section refers to a segment of code that is executed by multiple concurrent threads or processes, and which accesses shared resources. These resources may include shared memory, files, or other system resources that can only be accessed by one thread or process at a time to avoid data inconsistency or race conditions.

Note

- A **critical section** is a defined part of the code that accesses shared resources and needs synchronization to prevent concurrent execution by multiple threads.
- A **race condition** is an error condition that occurs when multiple threads access shared resources concurrently without proper synchronization, leading to unpredictable and erroneous results.
- **Synchronization mechanisms** are used to protect critical sections and prevent race conditions, ensuring the correct and consistent behavior of multi-threaded programs.

Classical Synchronization Problems

The classic synchronization problems arise due to the inherent challenges of concurrent access to shared resources. Proper synchronization is crucial to ensure data consistency, prevent race conditions, avoid deadlock, and mitigate starvation. Using synchronization mechanisms like mutexes, semaphores, and condition variables helps manage access and solve these problems effectively.

Producer-Consumer Problem

Problem:

- Involves two types of processes: producers and consumers.
- Producers generate data and place it into a buffer.
- Consumers take data from the buffer and process it.
- The challenge is to ensure that the producer does not add data to a full buffer and the consumer does not remove data from an empty buffer.

Solution:

Use a bounded buffer, mutexes, and semaphores to synchronize access.

Dining Philosophers Problem

Problem:

- Five philosophers sit at a table, each alternating between thinking and eating.
- There are five forks, one between each pair of philosophers.
- A philosopher needs both adjacent forks to eat, creating potential for deadlock.

Solution:

Introduce mechanisms to avoid deadlock, such as only allowing four philosophers to sit at the table at the same time, using a semaphore for each fork, or implementing a resource hierarchy.

Readers-Writers Problem

Problem:

- Involves processes (readers and writers) accessing a shared resource (such as a database).
- Multiple readers can read the resource simultaneously.
- Writers need exclusive access to modify the resource.

Solution:

Implement synchronization to ensure that no writer is writing while readers are reading, and that no readers are reading while a writer is writing.

Memory Management in Operating Systems

Memory management is a crucial function of an operating system (OS) that involves managing the computer's primary memory. It ensures the efficient allocation, deallocation, and use of memory, allowing multiple processes to run concurrently without interfering with each other.

Key Functions of Memory Management

Memory Allocation:

1. ***Static Allocation***: Memory is allocated at compile time.
2. ***Dynamic Allocation***: Memory is allocated at runtime.

Memory Deallocation:

- Reclaiming memory that is no longer in use by the processes.

Address Binding:

1. ***Compile-Time Binding***: Addresses are determined at compile time.
2. ***Load-Time Binding***: Addresses are determined at load time.
3. ***Execution-Time Binding***: Addresses can be changed during execution using dynamic relocation.

Memory Protection:

- Ensuring that a process cannot access memory allocated to another process.

Memory Sharing:

- Allowing processes to share memory to reduce redundancy and increase efficiency.

Types of Memory Management

Single Contiguous Allocation:

The simplest form of memory management where the entire memory is available to a single process.

Partitioned Allocation:

1. ***Fixed Partitioning:*** Memory is divided into fixed-sized partitions.
2. ***Dynamic Partitioning:*** Memory is divided into partitions of variable sizes.

Paged Memory Management:

- ***Paging:*** Memory is divided into fixed-size pages, and physical memory is divided into frames. Pages are mapped to frames.
- ***Page Table:*** Keeps track of where each page is stored in physical memory.

Segmented Memory Management:

- ***Segmentation:*** Memory is divided into segments based on logical divisions of a program, such as functions, arrays, etc.
- ***Segment Table:*** Keeps track of the base address and length of each segment.
- ***Segmented Paging:*** Combines segmentation and paging. Each segment is divided into pages, and pages are mapped to frames.

Memory Allocation Strategies

1. **First-Fit:**
    - Allocates the first block of memory that is large enough.
2. **Best-Fit:**
    - Allocates the smallest block of memory that is large enough, minimizing wasted space.
3. **Worst-Fit:**
    - Allocates the largest available block, which may leave large leftover spaces.
4. **Fragmentation**
    - ***External Fragmentation:*** Free memory is scattered throughout, making it difficult to find a contiguous block of memory.
    - ***Internal Fragmentation:*** Allocated memory may have small unused portions within it, leading to wasted space.

Paging

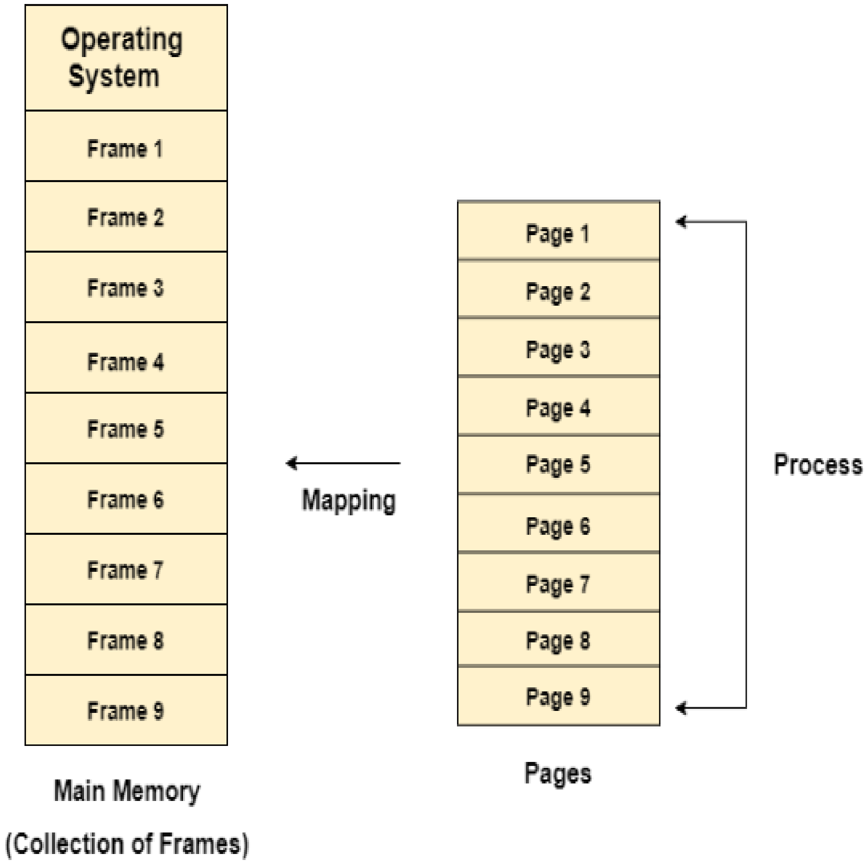
In Operating Systems, **Paging** is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the **paging** is to divide each process into the form of **pages**. The main memory will also be divided into the form of **frames**.

One page of the process is to be stored in one of the **frames** of the memory. The pages can be stored at different locations of the memory, but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise, they reside in the secondary storage.

Memory management unit (MMU) of OS needs to convert the page number to the frame number.



The purpose of **MMU** is to convert the logical address into the physical address. The **logical address** is the address generated by the CPU for every page while the **physical address** is the actual address of the frame where each page will be stored.

The logical address has two parts:

1. Page Number
2. Offset

Physical address space in a system can be defined as the size of the main memory.

Logical address space can be defined as the size of the process.

Segmentation

Segmentation is a memory management technique in which the memory is divided into variable size parts. Each part is known as a **segment** which can be allocated to a process.

The details about each segment are stored in a table called a **segment table**.

Segment table contains mainly two information about segment:

1. ***Base***: It is the base address of the segment
2. ***Limit***: It is the length of the segment.

Need for Segmentation

Till now, we were using Paging as our main memory management technique. Paging is closer to the Operating system rather than the User. It divides all the processes into the form of pages although a process can have some relative parts of functions which need to be loaded in the same page.

The operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.

Types of Segmentation in Operating System

1. ***Virtual Memory Segmentation:*** Each process is divided into several segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.
2. ***Simple Segmentation:*** Each process is divided into several segments, all of which are loaded into memory at run time, though not necessarily contiguously.

Note

Paging is a **non-contiguous memory allocation** technique which divides process into **fixed size pages**, and it suffers from **internal fragmentation**. In paging, **Logical address** is divided into **page numbers** and **page offset** and it uses **page table** to maintain page information.

Segmentation is a **non-contiguous memory allocation** technique which divides process into **variable size segments**, and it suffers from **external fragmentation**. Here **Logical address** is divided into **segment numbers** and **segment offset** and it uses **segment table** to maintain segment information.

Fragmentation

Fragmentation refers to the inefficient use of memory that leads to wastage and reduced performance. It occurs when memory is allocated and deallocated in a way that leaves unusable gaps.

Types of Fragmentation:

1. ***Internal Fragmentation***
   It occurs when the allocated memory to a process is more than the amount of memory requested/required by the process.
2. ***External Fragmentation***
   It occurs when free memory is fragmented into small, non-contiguous blocks over time, making it difficult to allocate large contiguous blocks of memory to processes, even though there is enough total free memory available.

Examples:

Internal Fragmentation

Scenario: A process requests 5 KB of memory, but the memory management system allocates a fixed-size block of 8 KB.

Result: 3 KB of the allocated block is wasted and cannot be used by the process.

External Fragmentation

Scenario: After several allocations and deallocations, the memory is scattered into several small free blocks (e.g., 5 KB, 12 KB, 3 KB).

Result: A new process requests 15 KB of contiguous memory, but it cannot be allocated even though the total free memory (20 KB) is sufficient because there is no single contiguous block of 15 KB available.

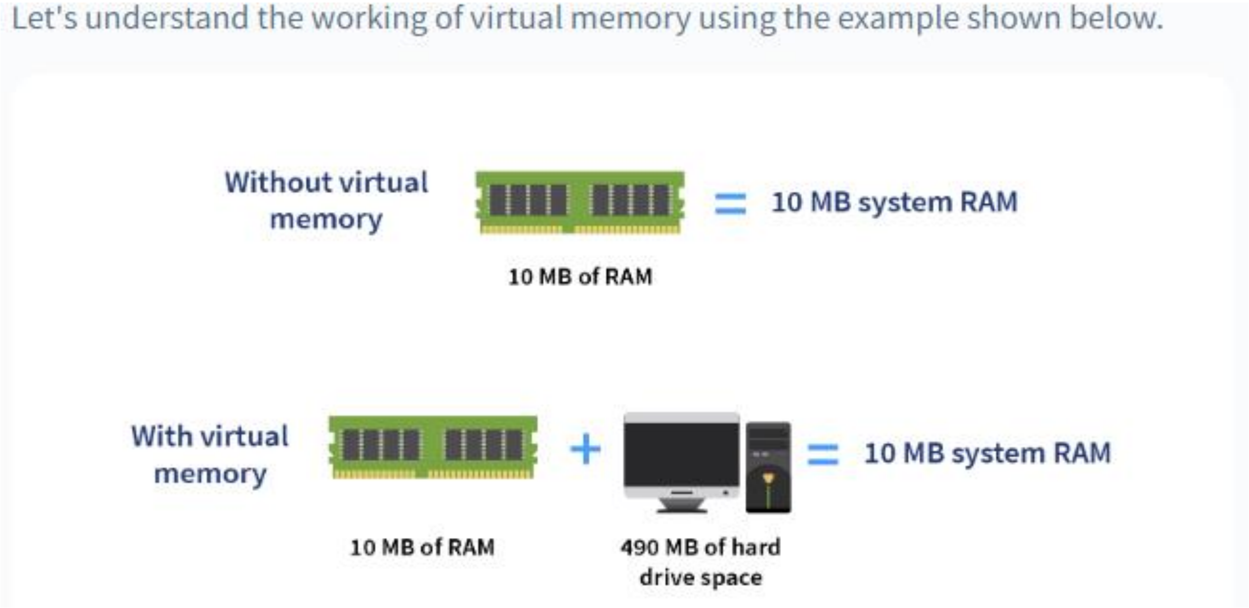
Virtual Memory

Virtual Memory is a part of secondary storage that gives the user the **illusion** that it is a part of the main memory. **(Virtual memory = combination of main memory & secondary memory).**

Virtual memory allows a system to execute heavier applications or multiple applications simultaneously without exhausting the main memory (RAM).

When running multiple heavy applications at once, the system's **RAM may get overloaded.** To mitigate this issue, some data stored in RAM that isn't being actively used can be **temporarily relocated to virtual memory**.

How Virtual Memory Works?



Let's understand the working of virtual memory using the example shown below.

Assume that an operating system uses **500 MB of RAM** to hold all the running processes. However, there is now only **10 MB** of actual capacity accessible on the RAM. The operating system will then allocate **490 MB of virtual memory** and manage it with an application called the ***Virtual Memory Manager (VMM).*** As a result, the **VMM** will generate a **490 MB** file on the **hard disc** to contain the extra RAM that is necessary. The OS will now proceed to address memory, even if only 10 MB of space is available because it considers 500 MB of actual memory saved in RAM. It is the **VMM's** responsibility to handle 500 MB of memory, even if only 10 MB is available.

Demand Paging

Demand Paging is a process that **swaps pages** from main memory to virtual memory (illuded form of secondary memory) and from virtual memory to main memory using **Page Replacement Algorithms.**

Demand paging keeps **frequently used pages** in main memory and **infrequently used pages** in secondary memory.

When the primary memory is full or insufficient a method called **swap out** is used to **swap processes/pages out** from main memory to virtual memory.

Similarly, when the main memory becomes available, a method called **swap in** is used to **swap processes/pages in** from virtual memory to main memory.

Page Replacement Algorithms

Page replacement algorithms are essential in **managing virtual memory** in an operating system. They decide which memory pages to swap out, write to disk when new pages are brought into memory, and ensure efficient memory usage. Here's an overview of **some common page replacement algorithms:**

1. **FIFO:** Replaces the oldest page.

   Example: If the pages in memory are [2, 3, 1] and a new page 4 arrives, page 2 (the oldest) is replaced, resulting in [3, 1, 4].

2. **LRU:** Replaces the least recently used page.

   Example: If the pages in memory are [2, 3, 1] and a new page 4 arrives, and the last access order was 2, 3, then 1, page 2 (least recently used) is replaced, resulting in [4, 3, 1].

3. **Optimal:** Replaces the page that will not be used for the longest time in the future.

   Example: If the pages in memory are [2, 3, 1] and the future page access order is 3, 1, 2, page 2 (used farthest in the future) is replaced, resulting in [4, 3, 1].

Thrashing

Thrashing occurs when a system spends more time handling **page faults** than executing processes, leading to a significant decline in performance.

This usually happens when there is not enough physical memory to support the active processes, causing constant swapping of pages in and out of memory.

Page Faults occur when a process tries to access a page that is not currently in physical memory (RAM).

File System: A file is a collection of related information that is recorded on secondary storage or file is a collection of logically related entities.

File Directories: Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system.

SINGLE-LEVEL DIRECTORY: In this a single directory is maintained for all the users

TWO-LEVEL DIRECTORY: Due to two levels, there is a path name for every file to locate that file.

TREE-STRUCTURED DIRECTORY: Directory is maintained in the form of a tree. Searching is efficient and there is grouping capability.

Continuous Allocation: A single continuous set of blocks is allocated to a file at the time of file creation.

Linked Allocation (Non-contiguous allocation): Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain.

Indexed Allocation: It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file.

Disk Scheduling

Disk scheduling is a technique in operating systems which is used to manage the order in which disk I/O (input/output) requests are processed. Disk scheduling is also known as **I/O Scheduling**. The main goals of disk scheduling are to optimize the performance of disk operations, reduce the time it takes to access data and improve overall system efficiency.

Key Terminology in Disk Scheduling

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or written.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Seek Time + Rotational Latency + Transfer Time
- **Disk Response Time:** Response Time is the average time spent by a request waiting to perform its I/O operation. Average Response time is the response time of all requests.

Disk Scheduling Algorithms

Disk scheduling algorithms are crucial in managing how data is read from and written to a computer's hard disk. These algorithms help determine the order in which disk read and write requests are processed, significantly impacting the speed and efficiency of data access.

Common disk scheduling methods include **First-Come, First-Served (FCFS), Shortest Seek Time First (SSTF), SCAN, C-SCAN, LOOK, and C-LOOK.**

FCFS: FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

SSTF: In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in a queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first.

SCAN: In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also known as elevator algorithm.

CSCAN: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

LOOK: It is like the SCAN disk scheduling algorithm except for the difference that the disk arm despite going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

CLOOK: As LOOK is like SCAN algorithm, in a similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm despite going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Disclaimer

- The following notes are general summaries and overviews of the topics discussed.
- These notes are not exhaustive and do not cover all aspects of the subject matter.
- The information provided herein is intended for educational purposes only and should not be used as a substitute for professional advice, detailed study, or official course materials.

References

For more detailed information, please refer to the following resources:

Reference 1: Complete OS Tutorial

Reference 2: Complete OS Tutorial

Reference 3: OS Cheat Sheet

Reference 4: OS Most Asked Questions

Reference 5: Top 50 OS Interview Questions

Prepared By: ***Reddy Venkat Kalyan***