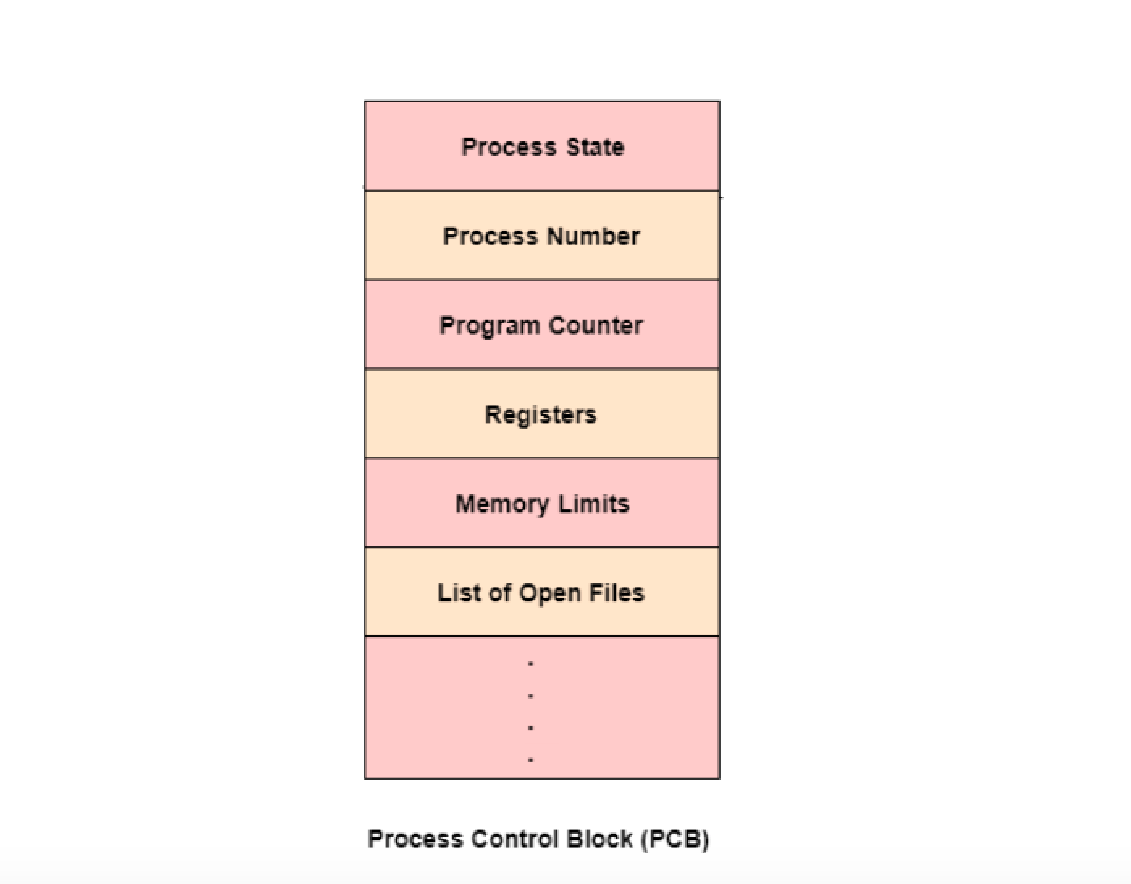
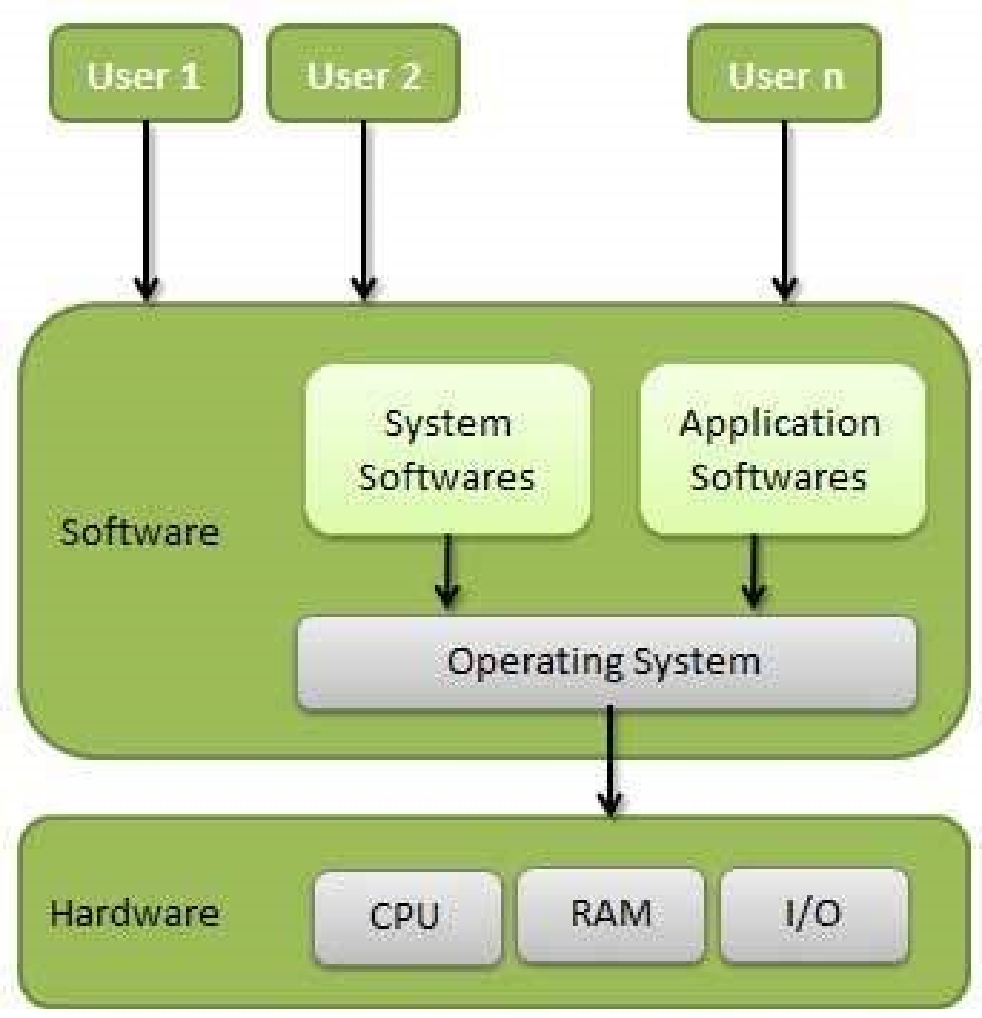
**Definitions:**

* An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.
* Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data.
* Mutual exclusion is a property of process synchronization which states that “no two processes can exist in the critical section at any given point of time. and A race condition occurs when two threads access a shared variable at the same time.
* A deadlock is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process.
* Starvation is a process in which the low priority processes are postponed indefinitely because the resources are never allocated.



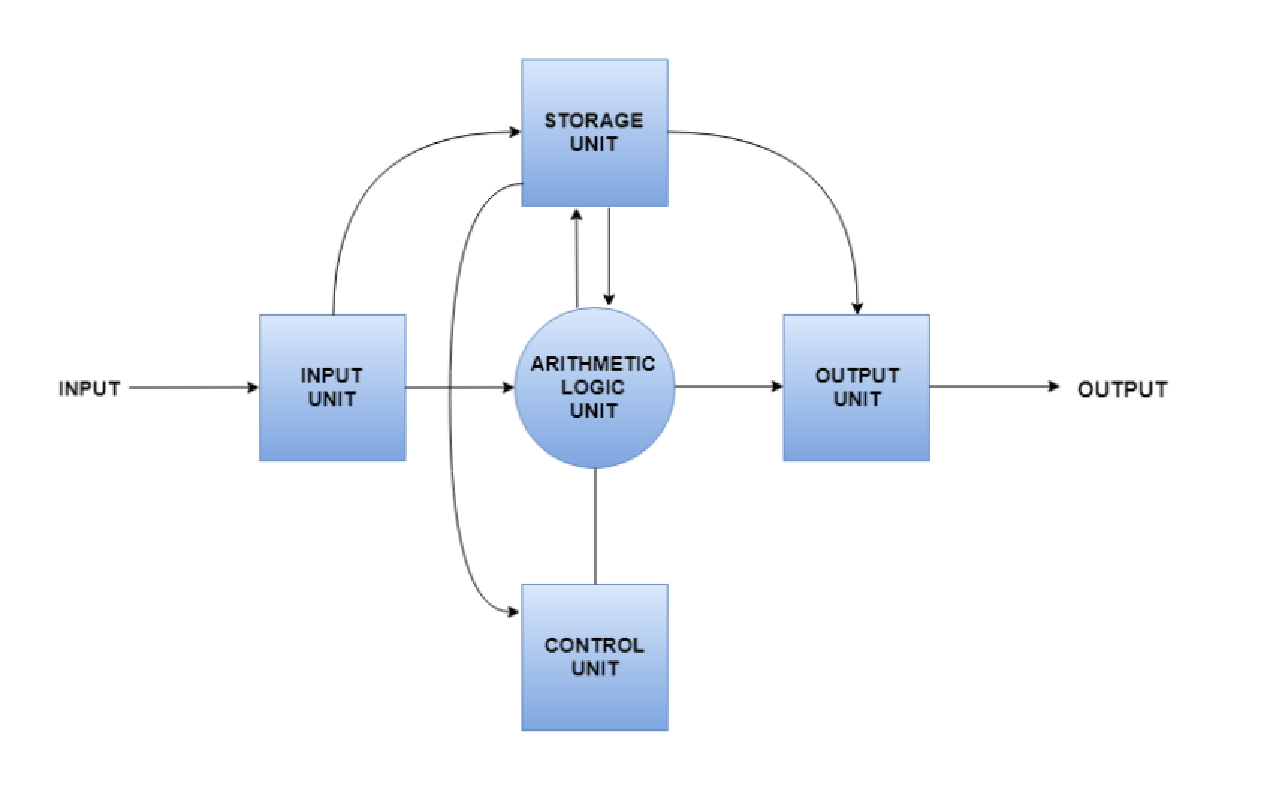
Process control block : Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.

The kernel is a core component of an operating system and serves as the main interface between the computer's physical hardware and the processes running on it.

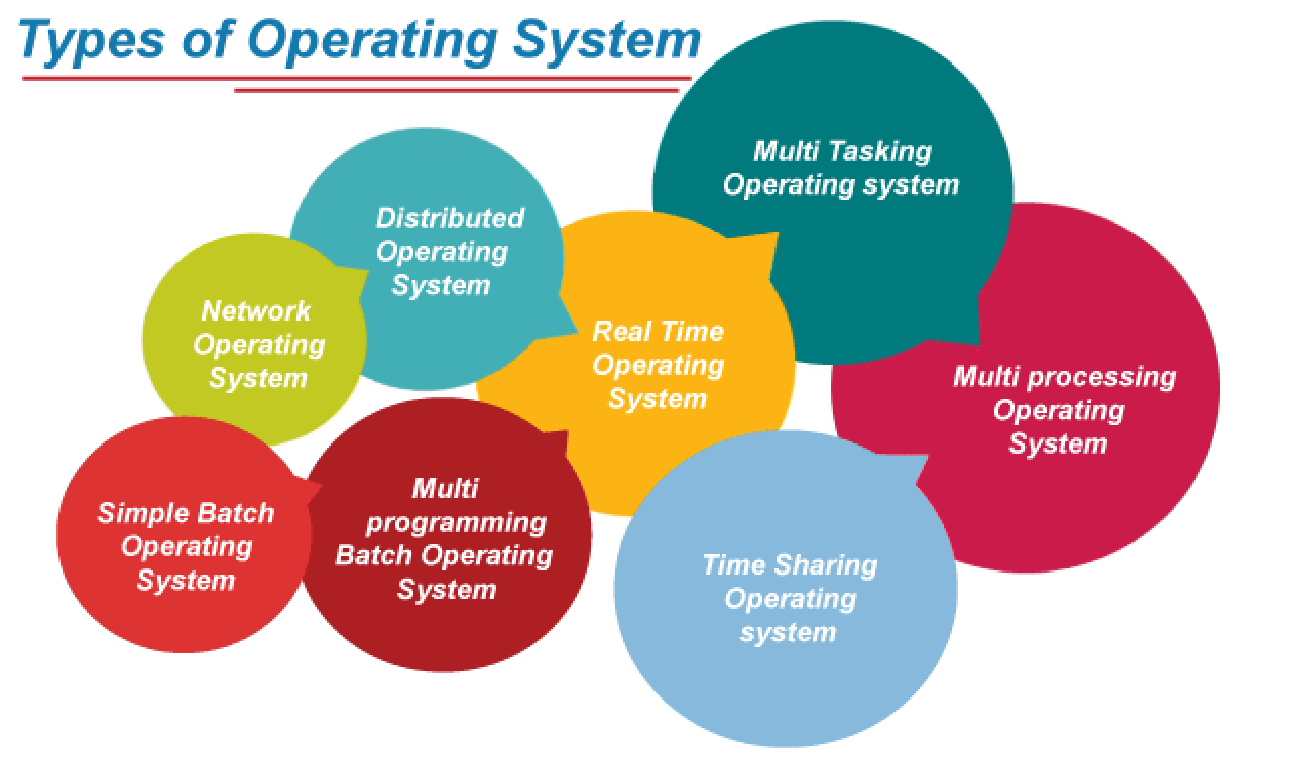
**Functions of os:**

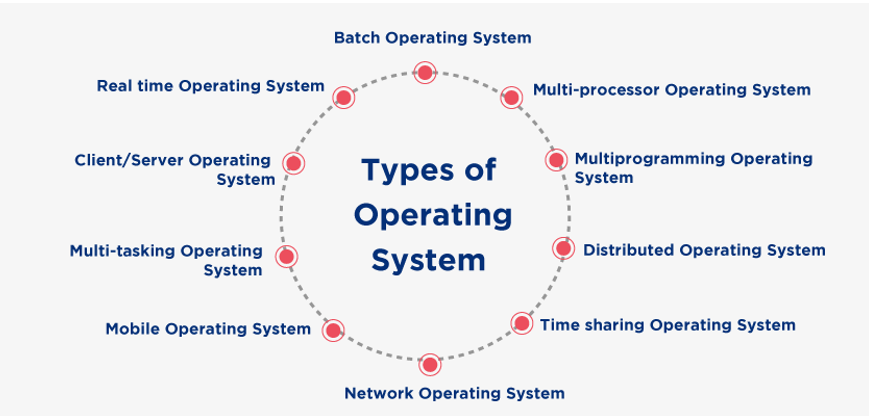
* Memory Management
* Processor Management
* Device Management
* File Management
* Network Management
* Security
* Control over system performance
* Job accounting
* Error detecting aids
* Coordination between other software and users.

**Computer Architecture:**

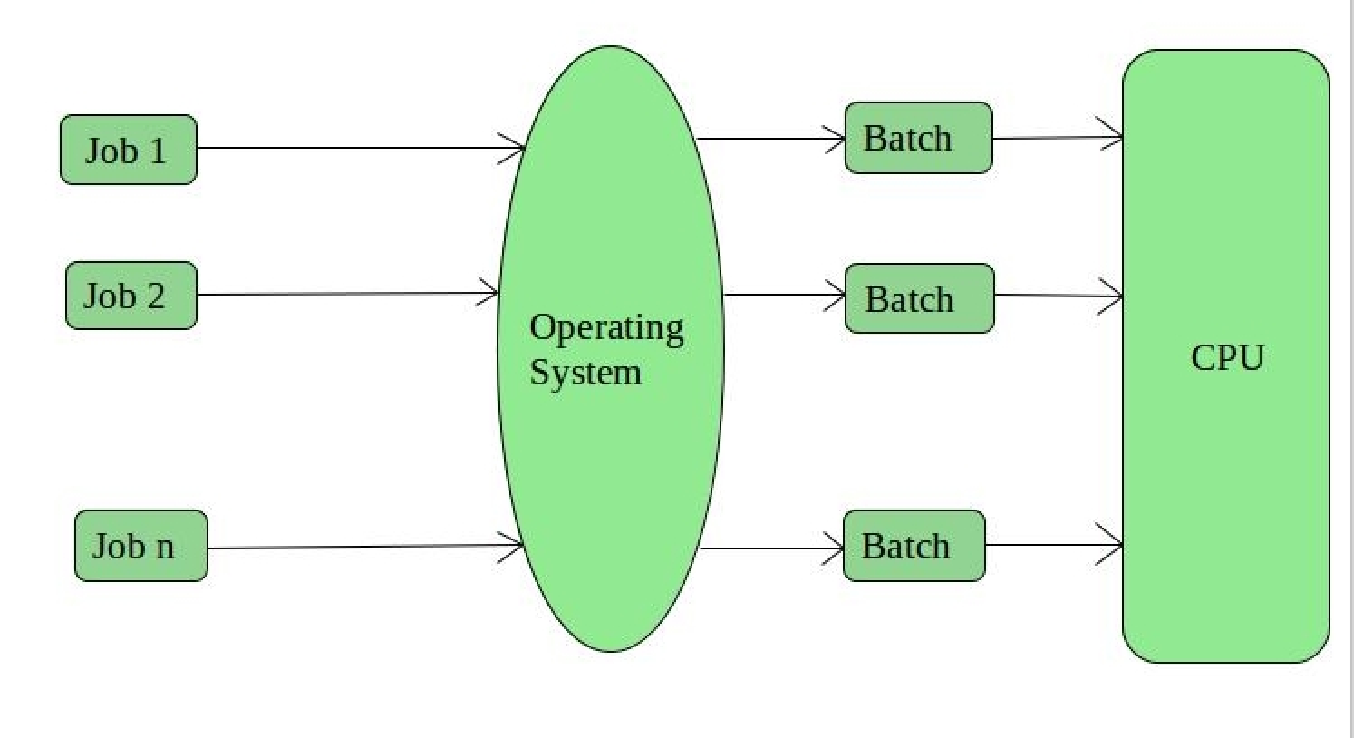
****

**Types of OS:**

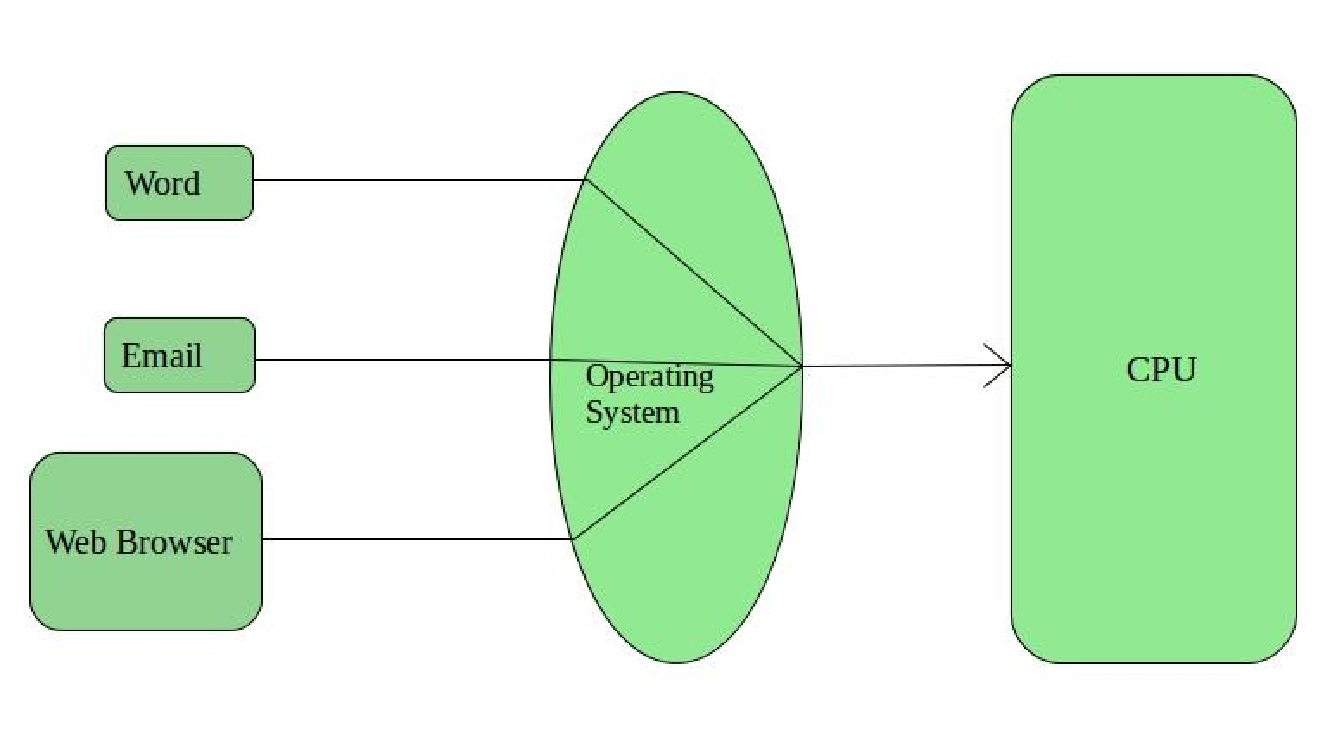
****

****

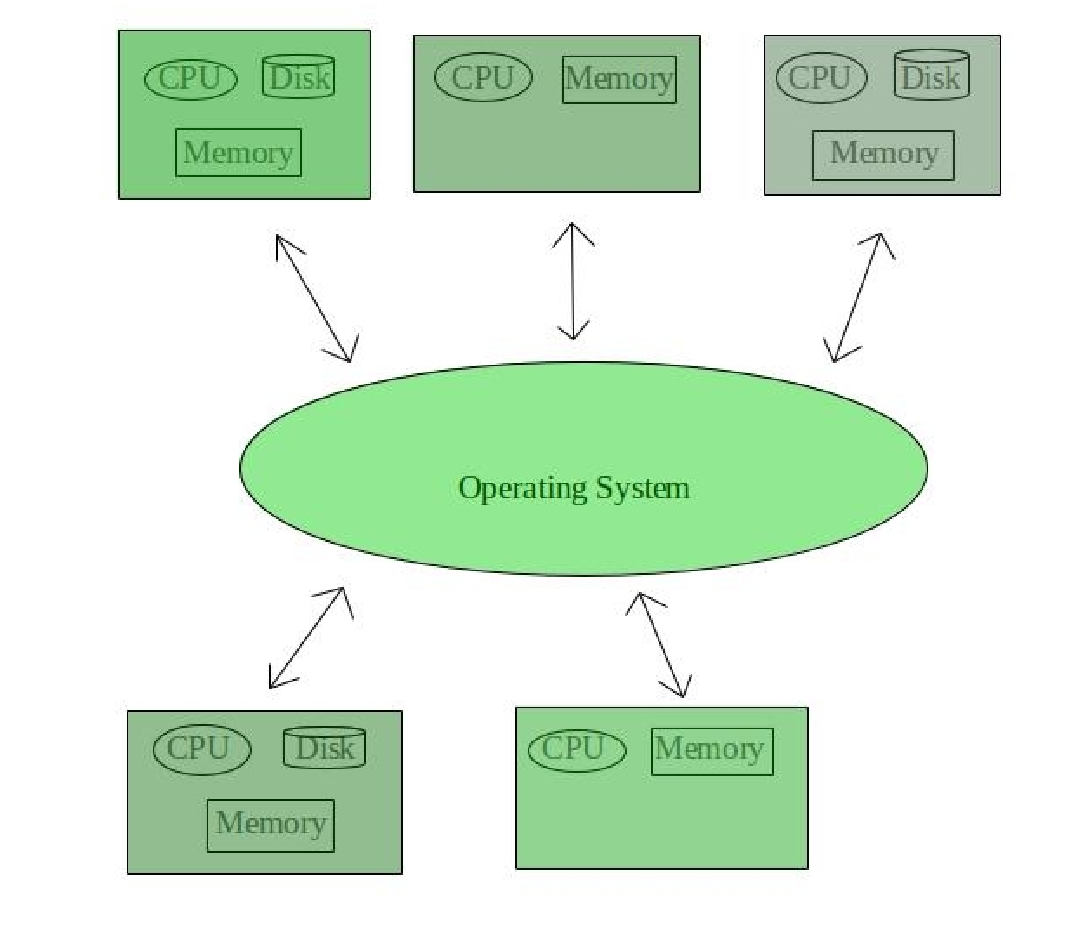
1. Batch os:



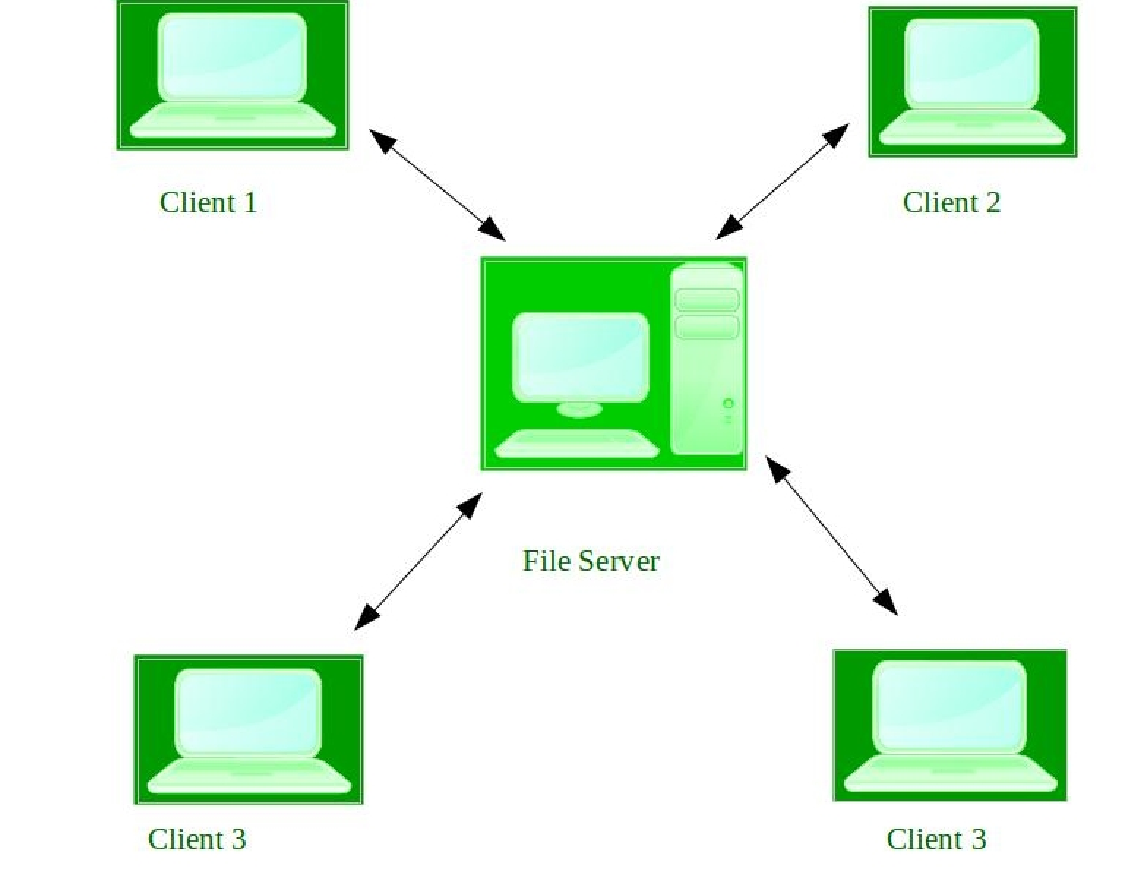
2. Time sharing or Multitasking Systems:



3. Distibuted systems:



4. Network OS:



5. 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. This time interval is called response time.

hard real time(very strict with time like missile launches)

soft real time

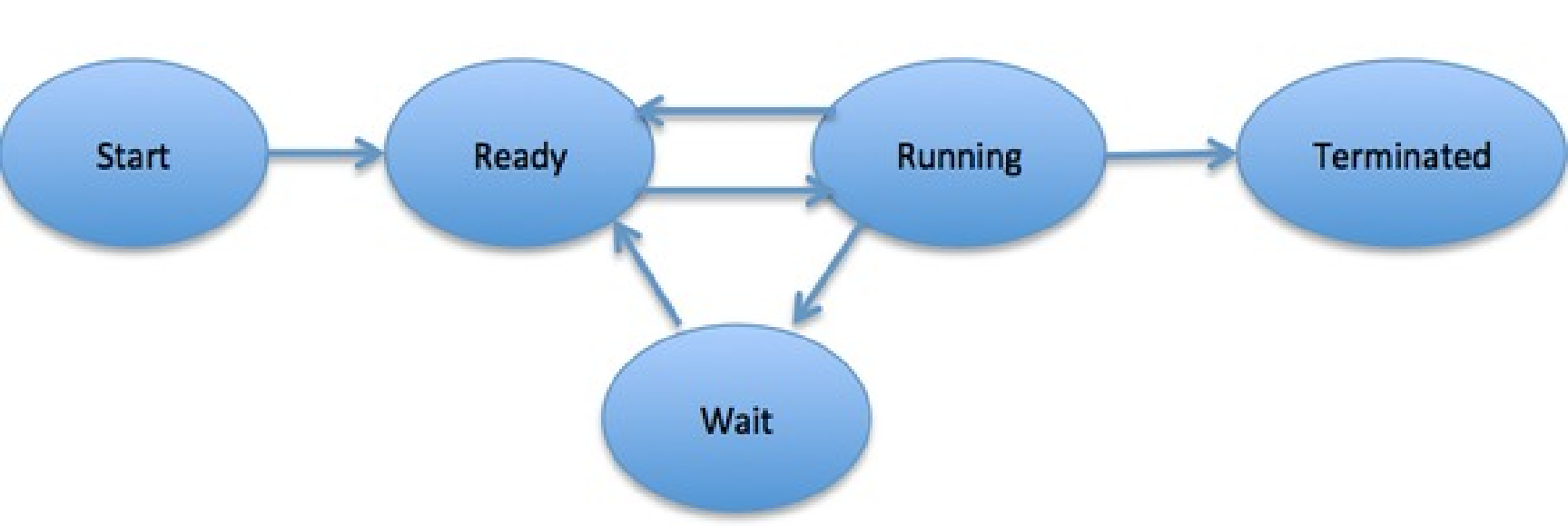
**Processes and Threads:**

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

A thread is a path of execution within a process. A process can contain multiple threads

a context switch is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point, and then restoring a different, previously saved, state.

States:

****

**CPU Scheduling:**

CPU scheduling is the task performed by the CPU that decides the way and order in which processes should be executed.

In Preemptive Scheduling the resources are allocated for limited amount of time and taken back when it switches form running state to waiting state while in non-preemptive once the resources gets allocated it cannot be taken out until process terminates.



Some important terminologies relevant to CPU scheduling are:

* Arrival Time
* Burst Time
* Completion Time
* Turn Around Time
* Waiting Time
* Response Time

CPU scheduling has certain criteria that it uses to schedule processes in the most efficient manner:

* CPU utilization (maximize)
* Throughput (maximize)
* Turn Around Time (minimize)
* Waiting Time (minimize)
* Response Time (minimize)

Algorithms

First Come First Serve

Shortest Job First

Priority Scheduling

Round Robin

**types of scheduler:**

* Long-Term Scheduler or Job Scheduler. The job scheduler is another name for Long-Term scheduler. .. responsible for bringing processes from the JOB queue (or secondary memory) into the READY queue (or main memory)..
* Short-Term Scheduler or CPU Scheduler. CPU scheduler is another name for Short-Term scheduler. ...choose a process from the Ready Queue that is ready to run and assign the processor to it.
* Medium-Term Scheduler. The switched-out processes are handled by the Medium-Term scheduler. This is a decision whether to add a process to those that are at least partially in main memory and therefore available for execution.

**Deadlock:**

Necessary Conditions for Deadlock;

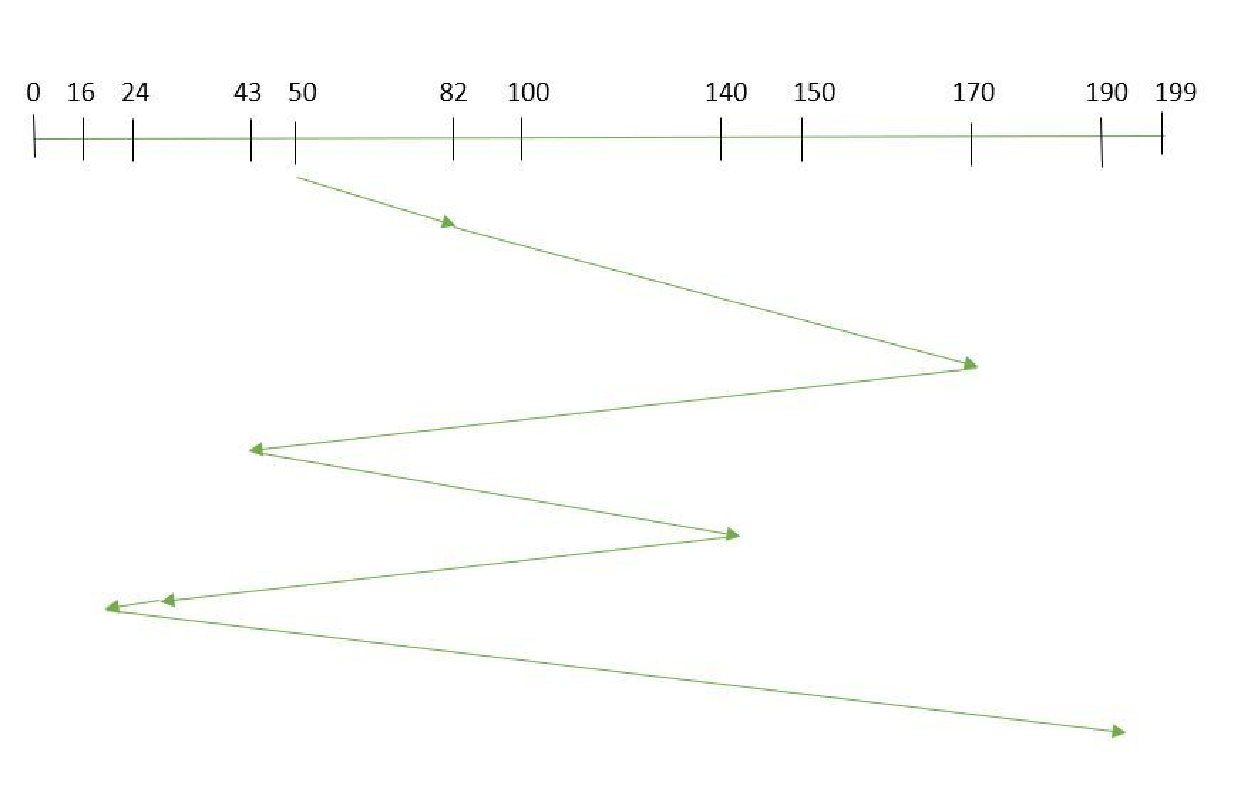
* Mutual Exclusion: Only one process can use a resource at any given time i.e. the resources are non-shareable.
* Hold and wait: A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
* No preemption: The resource can be released by a process voluntarily i.e. after execution of the process.
* Circular Wait: A set of processes are waiting for each other in a circular fashion.

**Disk Scheduling Algorithms:**

1.FCFS:

Suppose the order of request is- (82,170,43,140,24,16,190)

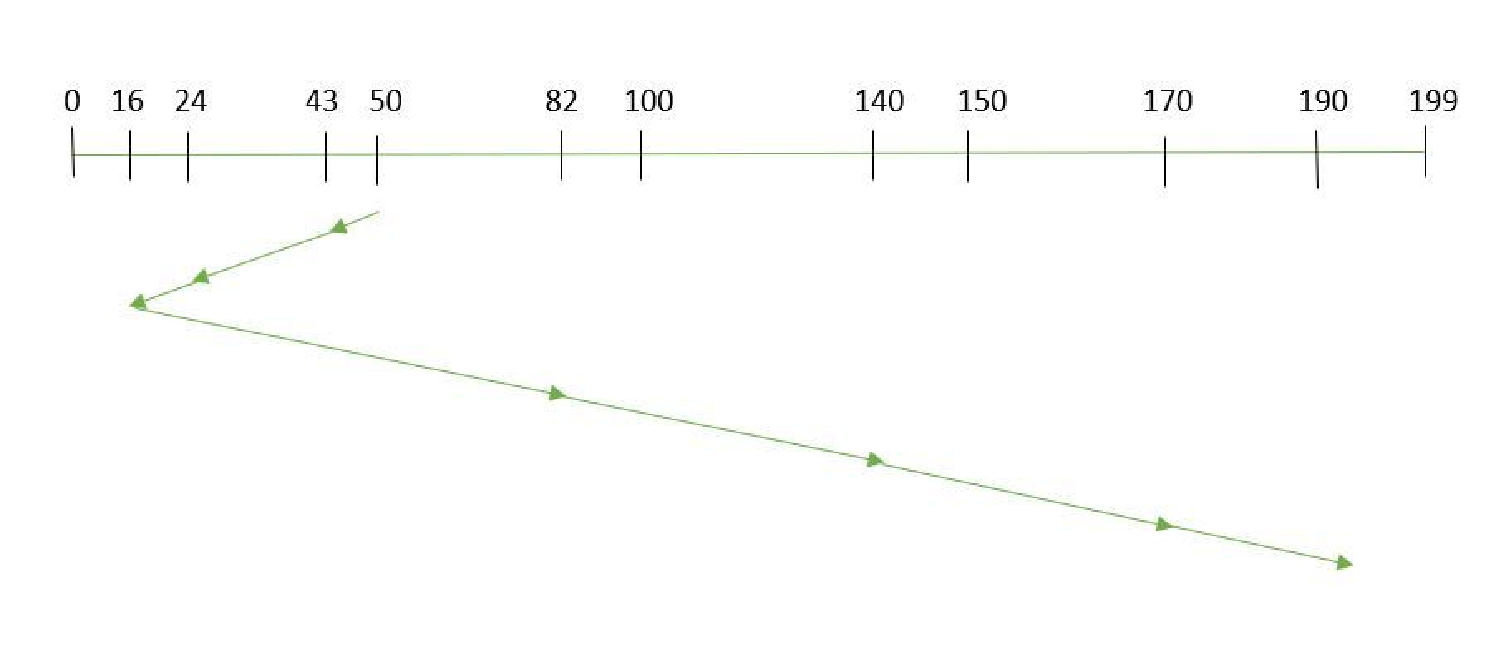
And current position of Read/Write head is : 50



2.SSTF (Shortest Seek Time First)

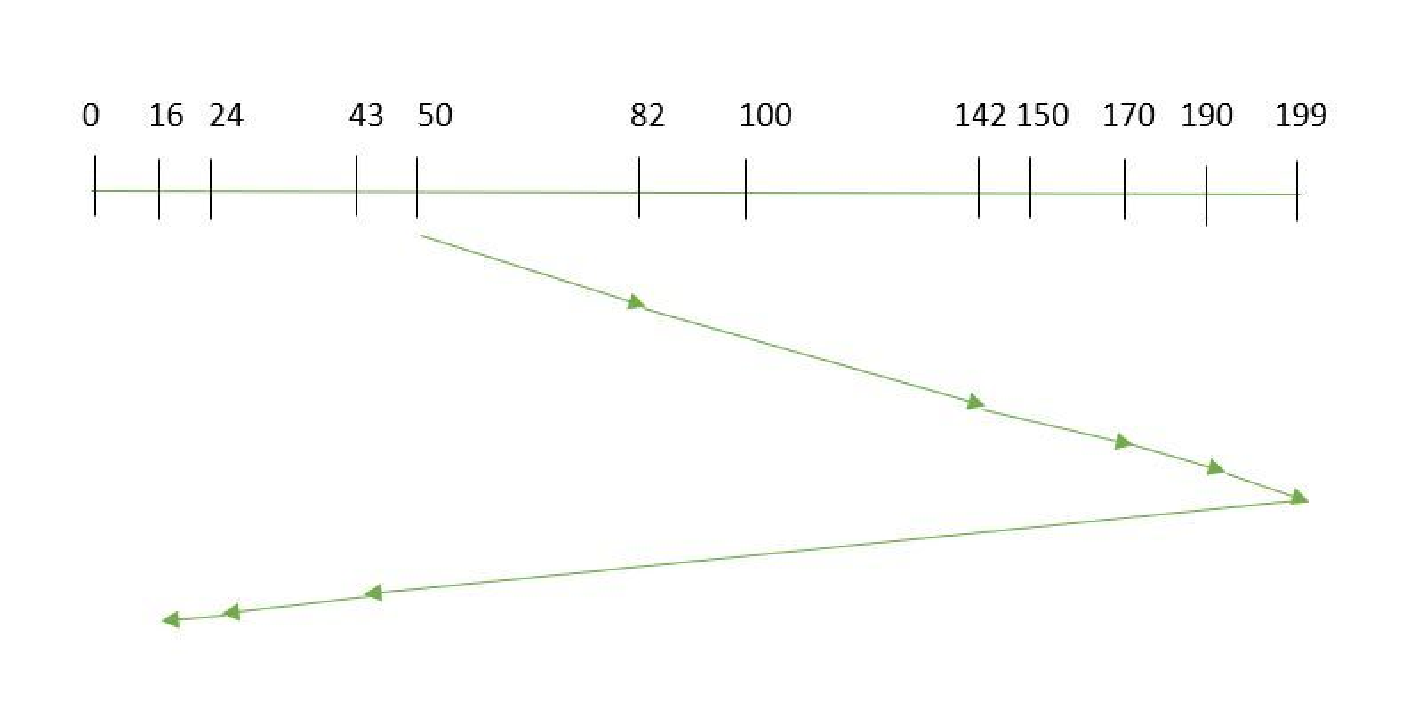
Suppose the order of request is- (82,170,43,140,24,16,190)

And current position of Read/Write head is : 50



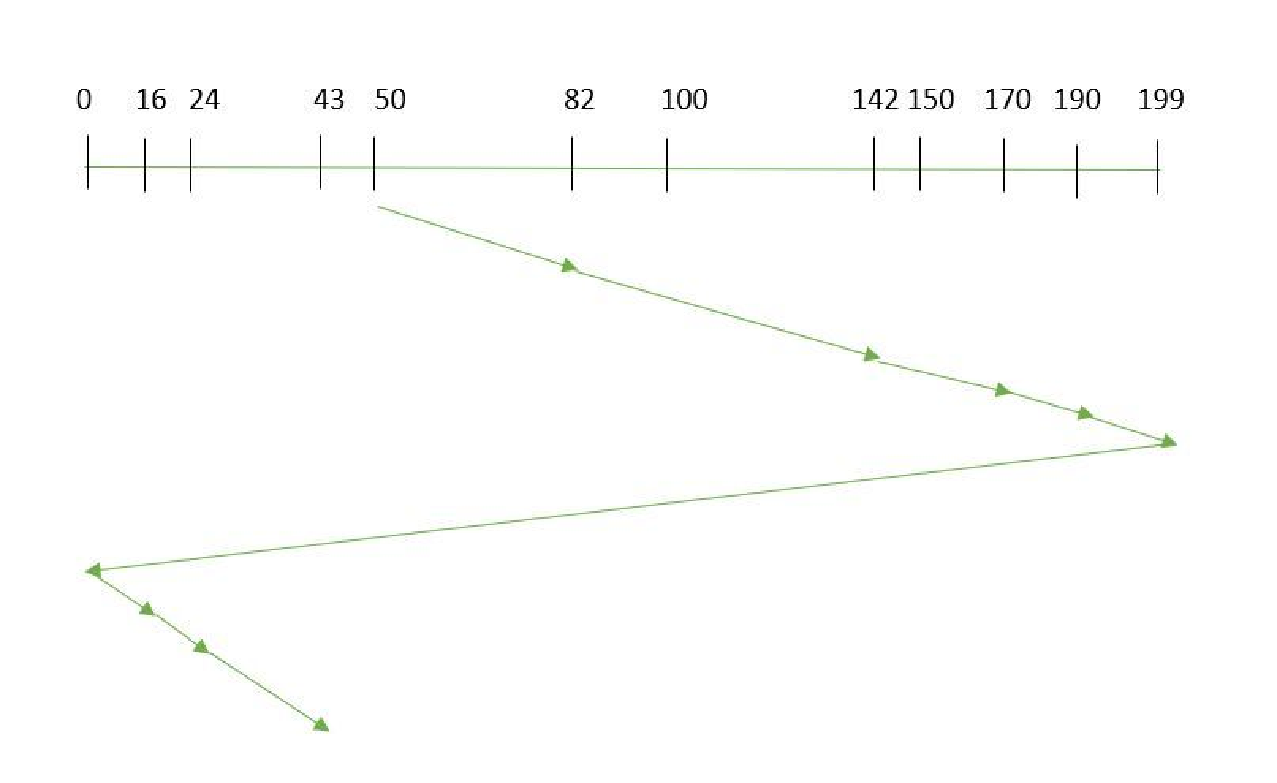
3.SCAN

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”.



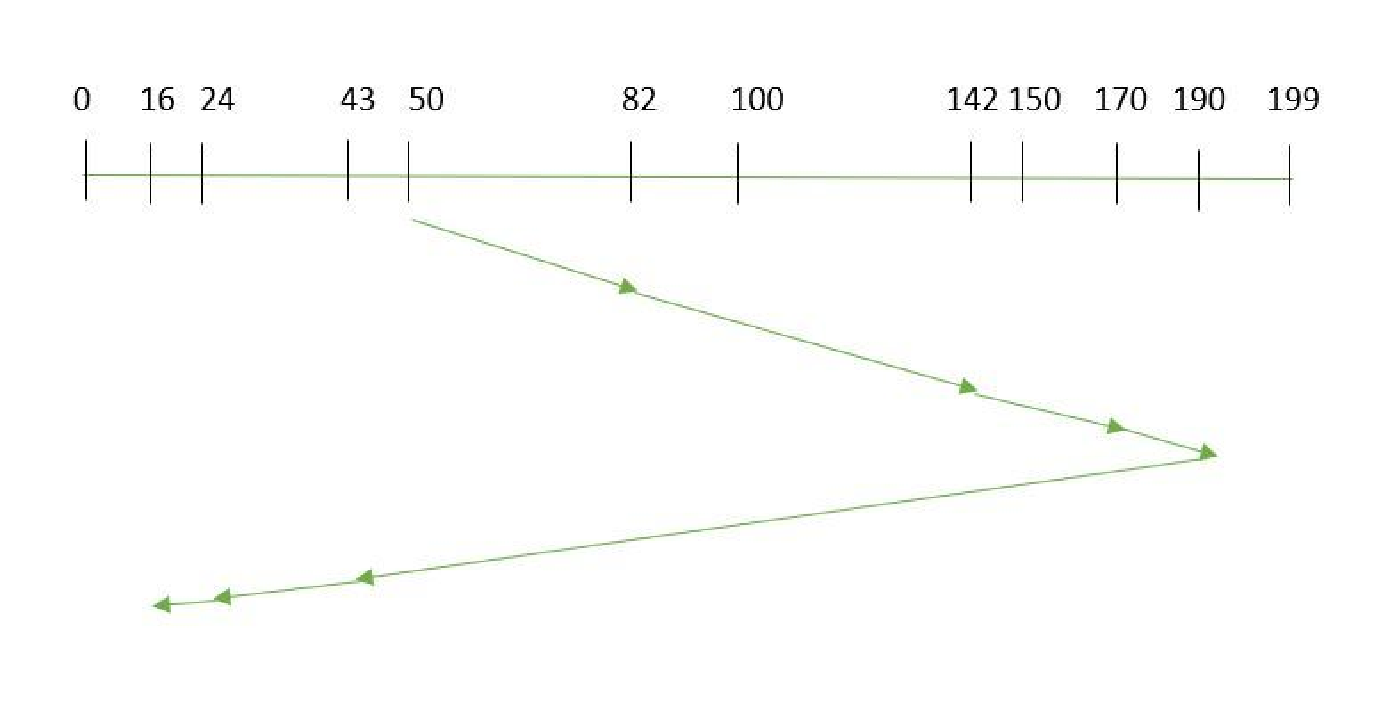
4.CSCAN

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”.



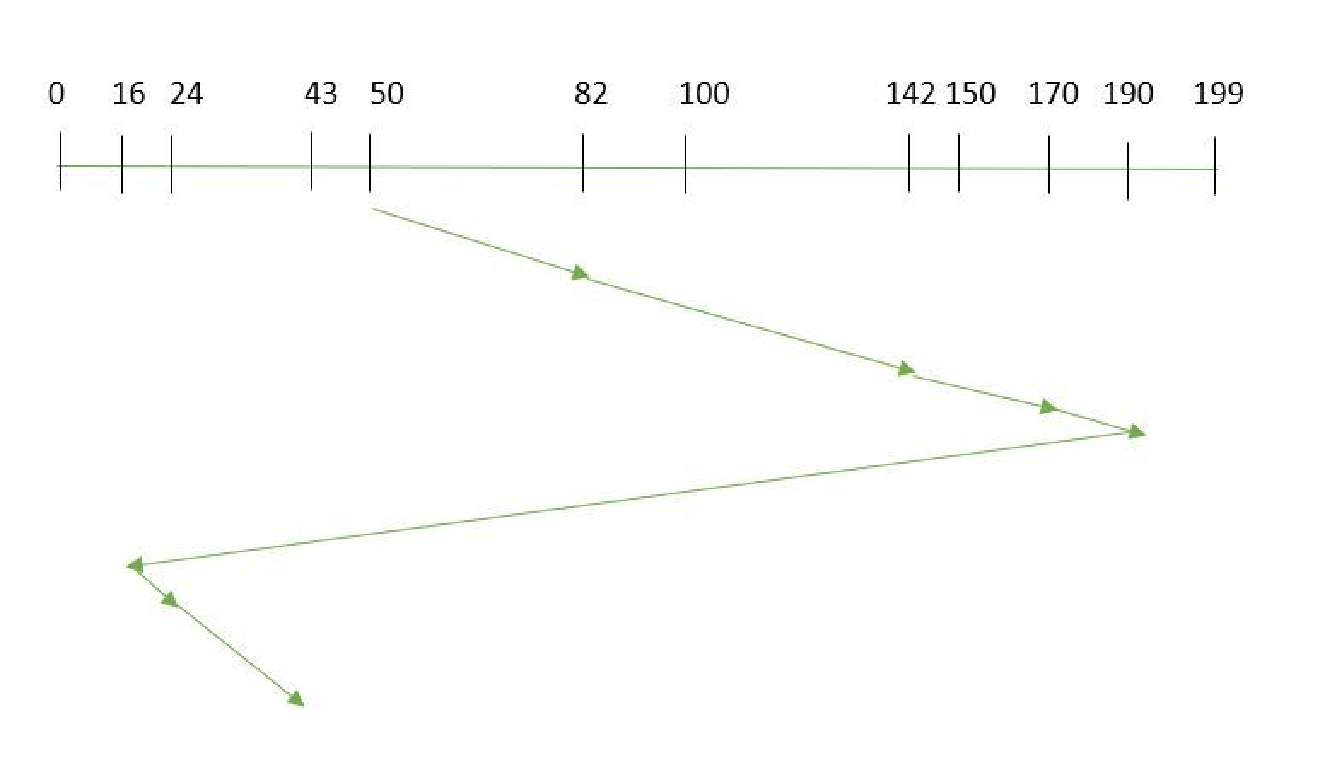
5.LOOK:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”.



6.CLOOK:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”



**Memory Management:**

Address Binding

Compile time

Load time

Execution time

Logical versus Physical Address Space

Swapping

Contiguous Allocation

Single Partition

Multiple Partition

First Fit

Best Fit

Worst Fit

Internal and External Fragmentation

Paging and Virtual Memory

Basics

Demand Paging

Page Replacement

Page Replacement Algorithms

FIFO

Belady’s anomaly

Optimal

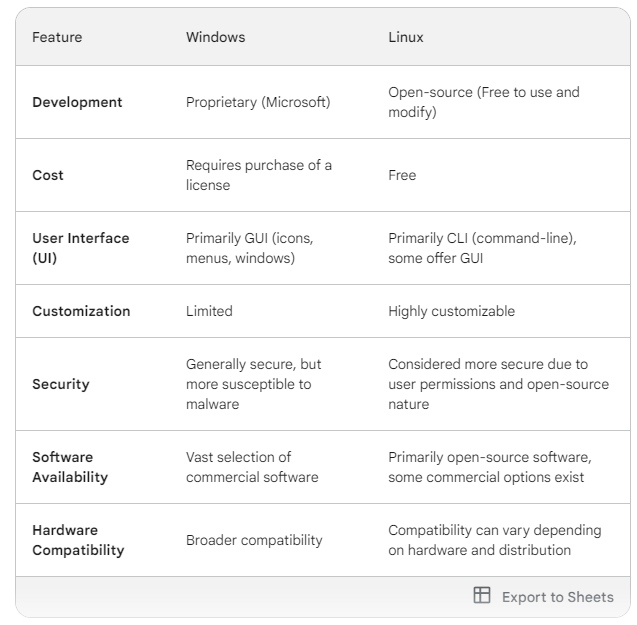
LRU

MFU

Thrashing

**Master’s Notes for OS:**

1. Explain the difference between Windows and Linux, in detail. Which one do you use? Why?



Note1- Windows provides a user-friendly interface, vast software availability, and wider hardware compatibility, making it ideal for beginners and those seeking a seamless experience.

Note2 - Linux, on the other hand, excels in cost-effectiveness, customization, security, and stability, catering to power users, budget-conscious individuals, and security-focused individuals.

Why Choose Windows?

* User-friendliness: Windows boasts a familiar and intuitive graphical user interface (GUI) with icons, menus, and windows, making it easy to navigate for beginners and non-technical users.
* Software Availability: Windows offers a vast selection of readily available software, including popular games, productivity applications, and creative tools.
* Hardware Compatibility: Due to its widespread market dominance, Windows generally has broader compatibility with various hardware components.
* Technical Support: Microsoft provides extensive official and community support for Windows, making it easier to find solutions to any technical issues you might encounter.

Why Choose Linux?

* Cost-effectiveness: As an open-source operating system, Linux is completely free to use and modify.
* Customization: Linux offers unparalleled levels of customization. You can tweak settings, configurations, and even the entire user interface to precisely match your preferences and workflow.
* Security: Due to its open-source nature and user permission structure, Linux is generally considered more secure than Windows.
* Stability and Performance: Linux is renowned for its stability and efficient resource management.

2. What are the five major activities of an operating system regarding process management?

An operating system undertakes several crucial activities to manage processes effectively. Here are the five major ones:

1. \*\*Creating and Deleting Processes:\*\* This involves allocating resources like memory and CPU time for new processes when requested and reclaiming those resources when processes are finished or terminated.

2. \*\*Suspending and Resuming Processes:\*\* Sometimes, a process might need to be temporarily paused, freeing up resources for other processes. The operating system allows for suspending and resuming processes as needed, ensuring efficient utilization of resources.

3. \*\*Synchronization:\*\* When multiple processes access shared resources (e.g., a file), conflicts can arise. The operating system provides mechanisms like semaphores and mutexes to ensure synchronized access, preventing data corruption and maintaining consistency.

4. \*\*Communication:\*\* Processes often need to exchange data and information. The operating system provides facilities like shared memory, pipes, and message queues to enable communication between processes, fostering collaboration and efficient data flow.

5. \*\*Deadlock Handling:\*\* The operating system employs strategies like deadlock detection and avoidance to prevent deadlocks or break them if they occur.

3. What are the three major activities of an operating system regarding memory management?

* Memory Allocation and Deallocation: As processes are created and terminated, the operating system needs to dynamically allocate memory space for them and deallocate the memory space once a process is finished. This involves keeping track of free and used memory blocks and assigning them to requesting processes efficiently.
* Tracking Memory Usage: The operating system constantly monitors which memory locations are being used by each process and for what purpose. This detailed tracking of memory usage enables efficient resource allocation, preventing conflicts and ensuring processes have the memory they need to run smoothly.
* Protection and Sharing: The operating system implements mechanisms to protect the memory space of each process from unauthorized access by other processes. This prevents data corruption and ensures the integrity of each process's execution. Additionally, the operating system can facilitate controlled sharing of memory between processes when necessary, enabling collaboration and data exchange.

3. What are the three major activities of an operating system regarding secondary-storage management?

1. \*\*Free Space Management:\*\* The operating system keeps track of \*\*available free space\*\* on the secondary storage device. When a new file needs to be written or existing files need to be modified, the operating system locates suitable free space to accommodate the data. This involves maintaining an efficient data structure (e.g., file allocation table) to record the allocation status of each storage block.

2. \*\*Storage Allocation:\*\* When a process requests to create a new file or expand an existing one, the operating system is responsible for \*\*allocating storage space\*\* for the data. This involves selecting appropriate free blocks from the available space and recording the allocation information in the file system. The choice of allocation strategy (e.g., contiguous or non-contiguous) can impact efficiency and performance.

3. \*\*Disk Scheduling:\*\* When multiple processes simultaneously request access to the secondary storage device (e.g., reading or writing files), the operating system needs to \*\*schedule\*\* these requests efficiently. This involves minimizing the total head movement of the disk drive, as excessive movement can significantly impact performance. Different disk scheduling algorithms (e.g., FCFS, SCAN) exist, each with its own advantages and drawbacks in terms of minimizing seek time and optimizing storage access.

5. What is an interrupt and name a few common functions of Interrupts?

In computer systems, an \*\*interrupt\*\* is a signal generated by either \*\*hardware\*\* or \*\*software\*\* that temporarily \*\*halts\*\* the currently executing program and \*\*switches the processor's attention\*\* to the source of the interrupt. This allows the processor to \*\*respond to urgent events\*\* in a timely manner without neglecting the ongoing task entirely.

Here are a few common \*\*functions\*\* of interrupts:

\* \*\*Handling I/O (Input/Output) operations:\*\* When a hardware device (e.g., keyboard, network card, disk drive) finishes an operation or requires attention, it can send an interrupt to the processor. This allows the processor to handle the device's request efficiently and return to the original task once the I/O operation is complete.

\* \*\*Responding to system calls:\*\* Programs can also initiate interrupts, called \*\*system calls\*\*, to request services from the operating system. This enables programs to perform tasks they cannot do on their own, such as accessing files, managing memory, or interacting with devices.

\* \*\*Handling errors and exceptions:\*\* Unexpected events like program errors, division by zero, or invalid memory access can trigger interrupts. This allows the operating system to take appropriate action, such as terminating the faulty program, displaying an error message, or attempting to recover from the error.

\* \*\*Implementing real-time computing:\*\* In real-time systems where timely response is crucial, interrupts are used to guarantee that critical events are handled promptly. For instance, in a real-time video editing system, an interrupt might be used to signal the need to update the displayed video frame at specific intervals.

Q.6 Write the Storage hierarchy according to its cost and speed.

- \*\*Registers:\*\* Tiny, blazing-fast storage within the CPU (think: lightning-quick notepad for calculations).

- \*\*Cache:\*\* High-speed memory bridging CPU and RAM, like a handy reference desk for frequently used data.

- \*\*RAM:\*\* Holds active programs and data, offering faster access than secondary storage, but still not as quick as cache (think: your active workspace).

- \*\*Secondary Storage (HDD/SSD):\*\* Non-volatile storage devices like hard drives and solid-state drives, slower than RAM but much cheaper and offer massive capacity (think: your filing cabinets for documents).

- \*\*Tertiary Storage (Tapes/Optical Disks):\*\* Slowest and cheapest option for long-term data archiving, like dusty boxes in the attic for rarely accessed information

Q.7 Describe the actions taken by a kernel to context-switch between processes.

Saves: Current process's state (registers, memory info, state) to its Process Control Block (PCB).

Chooses: The next process to run based on a scheduling algorithm.

Loads: The chosen process's state from its PCB back into registers and memory.

Resumes: Execution of the new process from where it left off.

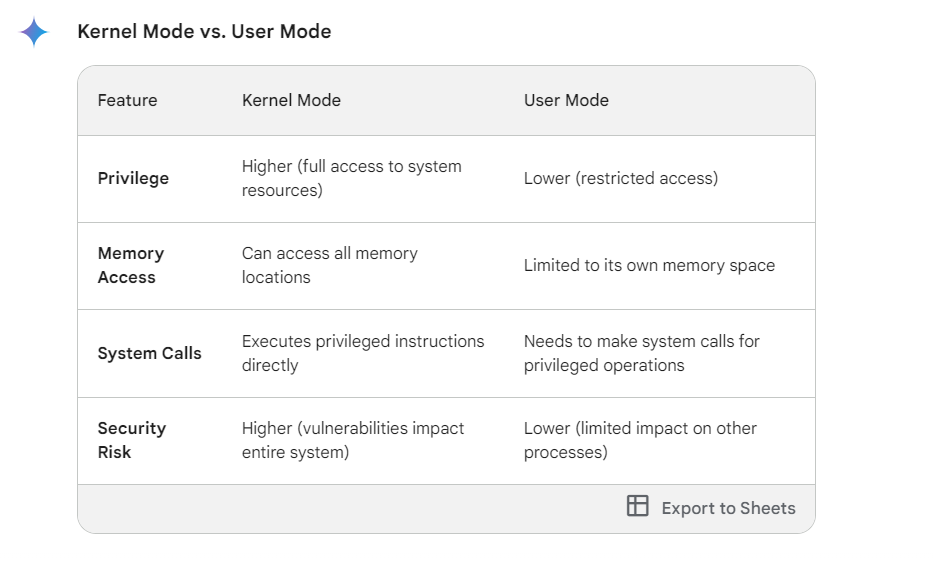
A kernel performs several actions to context-switch between processes, ensuring smooth execution and efficient resource allocation.

Firstly, the context switch is triggered by events like clock interrupts, system calls, or external events. The kernel then saves the state of the current process, including its program counter, registers, stack pointer, and other relevant information. This information is typically stored in a Process Control Block (PCB) associated with the process.

Next, the kernel uses a scheduling algorithm to determine the next process to run, considering factors like priority and resource needs. Once chosen, the kernel retrieves the saved state of the new process from its PCB and loads it onto the CPU, essentially restoring its execution environment. Finally, the kernel switches the CPU back to user mode, allowing the newly chosen process to resume execution seamlessly.

It's important to note that context switching involves significant data movement and takes some time compared to normal program execution. However, it's crucial for multitasking and efficient resource management by the operating system.

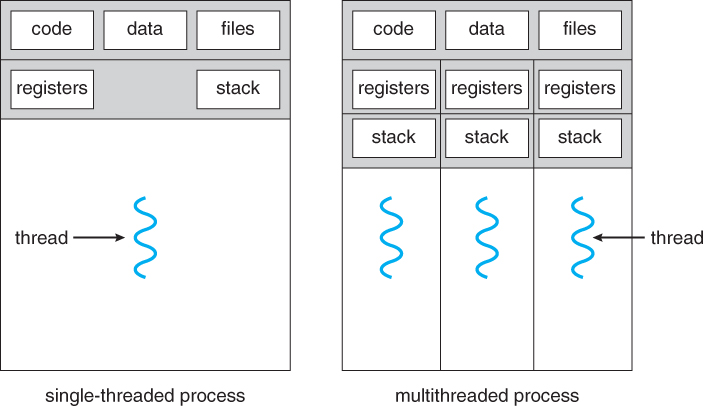
Q 8 Briefly, point out a few differences between Kernel mode and User mode.



Q.9 What resources are used when a thread is created? How do they differ from those used when a process is created?

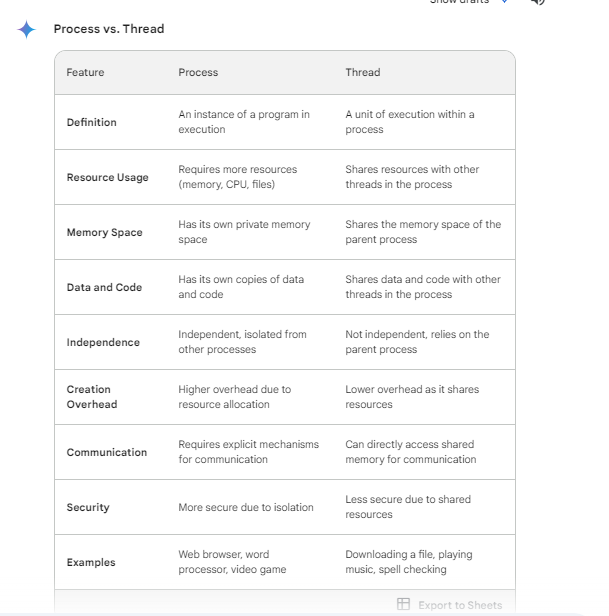
Threads are lightweight processes that share resources, making them more efficient to create and manage.

Processes are independent entities with their own set of resources, leading to higher resource requirements and overhead during creation



Q.10 Write the differences between a process and a thread.

Note- Processes provide a higher level of isolation and security, while threads offer better performance and efficiency due to their shared resources.



Q11 List the four conditions that must hold simultaneously to characterize a deadlock.

The four necessary conditions that must hold simultaneously for a deadlock situation to occur are:

1. \*\*Mutual exclusion:\*\* At least one resource must be held in a non-sharable mode, meaning only one process can access it at a time. This resource could be a physical device like a printer, a logical resource like a file lock, or even a specific memory segment.

2. \*\*Hold and wait:\*\* A process must be holding at least one resource while waiting to acquire another resource that is currently being held by another process. This creates a chain of waiting processes, each holding a resource and waiting for another.

3. \*\*No preemption:\*\* Acquired resources cannot be forcibly taken away from a process. They must be released voluntarily by the holding process. This prevents the operating system from intervening and freeing up resources to break the deadlock.

4. \*\*Circular wait:\*\* There must exist a circular chain of two or more processes where each process is waiting for a resource held by the next process in the chain. This creates a loop where no process can proceed, as each is waiting for a resource held by another process in the circular dependency.

If all four of these conditions are met simultaneously, a true deadlock situation exists, and the processes involved will be unable to proceed without external intervention from the operating system or other mechanisms.

Q 11: List three examples of deadlocks that are not related to a computer system environment.

1. \*\*Traffic Intersection Deadlock:\*\* Two cars approach a four-way intersection from different directions, wanting to turn left. They both need to enter the same lane to turn left, but neither can proceed because the other car is blocking their path. This creates a deadlock as both cars wait for the other to move, but neither can move due to the other's presence.

2. \*\*Dinner Party Deadlock:\*\* Three friends, A, B, and C, are planning a dinner party. A needs to borrow B's car to pick up groceries, but B needs A's help to set up the tables and chairs. C has the key to B's apartment, but they are waiting for A to pick them up so they can all go to B's place. This creates a circular waiting situation: A needs B's car, B needs A's help, and C needs access to the apartment but is waiting for A.

3. \*\*Handshake Deadlock:\*\* Two people try to greet each other by shaking hands, but they both extend their right hands. Neither can shake the other person's hand without changing their own hand position, leading to a stalemate. This represents a simple example of a deadlock arising from a lack of proper coordination or communication.

Q 12. Classify the Schedule algorithm in RTOS

The primary classification of scheduling algorithms in an RTOS (Real-Time Operating System) is based on \*\*preemption:\*\*

\* \*\*Preemptive scheduling:\*\* This allows the RTOS to \*\*interrupt\*\* a currently running task and switch to a higher-priority task. This ensures that more critical tasks are not delayed waiting for lower-priority tasks to finish.

\* \*\*Non-preemptive scheduling:\*\* This does not allow the RTOS to interrupt a running task. Once a task starts execution, it runs to completion without interruption. This can be simpler to implement but may not be suitable for real-time systems where strict deadlines need to be met.

Here are some popular examples of scheduling algorithms within each category:

\*\*Preemptive:\*\*

\* \*\*Fixed-priority scheduling:\*\* Tasks are assigned priorities, and the scheduler chooses the highest-priority ready task for execution.

\* \*\*Shortest Job First (SJF):\*\* Tasks are scheduled based on their execution time, with the shortest job being executed first.

\* \*\*Rate Monotonic Scheduling (RMS):\*\* Tasks with stricter deadlines have higher priorities.

\*\*Non-preemptive:\*\*

\* \*\*First Come, First Served (FCFS):\*\* Tasks are scheduled in the order they arrive.

\* \*\*Round-Robin (RR):\*\* Tasks are allocated fixed time slices and executed in a round-robin fashion.

The choice of scheduling algorithm depends on the specific needs of the real-time system, such as:

\* \*\*Deadline requirements:\*\* If tasks have strict deadlines, a preemptive algorithm with guaranteed execution time is often preferred.

\* \*\*Complexity:\*\* Non-preemptive algorithms are generally simpler to implement, but preemptive algorithms may offer better performance for complex systems.

\* \*\*Fairness:\*\* RR scheduling ensures a level of fairness among tasks, while FCFS can lead to starvation for lower-priority tasks.

Q 13. Define RTOS, Preemption, Reentrancy, Concurrency and Deadlock.

\*\*RTOS (Real-Time Operating System):\*\* An RTOS is a specialized operating system designed for \*\*real-time applications\*\*. These applications have strict timing requirements, meaning they need to respond to events within a specific time frame. RTOSes are typically smaller and simpler than general-purpose operating systems, as they focus on providing essential features for real-time processing, such as task scheduling, interrupt handling, and device drivers.

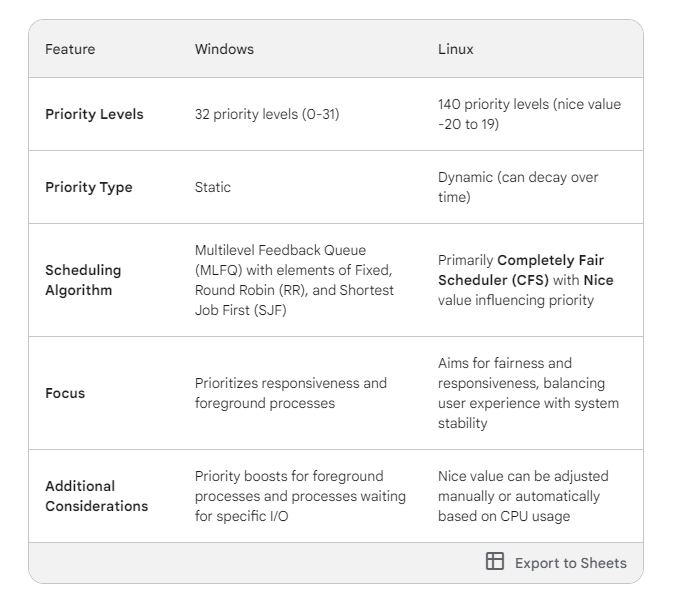
\*\*Preemption:\*\* In the context of multitasking, \*\*preemption\*\* refers to the ability of the operating system to take control of the CPU from a currently running task and give it to another higher-priority task. This allows the system to respond to more critical tasks in a timely manner, even if a lower-priority task is currently using the CPU.

\*\*Reentrancy:\*\* A function or code segment is considered \*\*reentrant\*\* if it can be safely called from multiple tasks or contexts without introducing errors. This means that the function does not rely on global variables or other state information that could be modified by other tasks in unpredictable ways. Reentrancy is essential for developing thread-safe code in multithreaded environments.

\*\*Concurrency:\*\* \*\*Concurrency\*\* refers to the ability of a system to handle multiple tasks or activities at the same time. This does not necessarily mean that the tasks are executing simultaneously on the same processor core; instead, it implies that the system can rapidly switch between tasks, giving the illusion of simultaneous execution. Concurrency is often achieved through multitasking or multithreading techniques.

\*\*Deadlock:\*\* A \*\*deadlock\*\* is a situation where two or more tasks are blocked indefinitely, waiting for resources that are held by each other. This can occur when tasks acquire resources in a specific order and then attempt to acquire resources already held by other waiting tasks, creating a circular dependency. Deadlocks can be difficult to prevent and can significantly impact the performance of a system.

Q.14 linux vs windows scheduling



Q15 What is a Race condition and list two real-life examples that are under Race condition?

\*\*What is a Race Condition?\*\*

\* In computing, a race condition occurs when multiple processes or threads access and modify a shared resource simultaneously. The final outcome of the operations depends on the unpredictable timing or order in which these threads execute.

\* This can lead to unexpected results, data corruption, or even system crashes if not handled carefully.

\*\*Real-Life Examples of Race Conditions\*\*

1. \*\*Bank Account Withdrawals:\*\*

- Imagine two people trying to withdraw money from the same bank account at the same time.

- If the system doesn't check the balance correctly or handle the withdrawal processes in a controlled way, both withdrawals might succeed even if there are insufficient funds, leading to an overdraft.

2. \*\*Ticket Sales:\*\*

- Consider an online ticketing system where multiple people are trying to purchase the last remaining ticket for a popular concert.

- If the system doesn't properly synchronize the ticket availability and purchase processes, it's possible that the system might sell the same ticket to multiple customers.

\*\*Important Considerations\*\*

\* Race conditions can be difficult to identify and reproduce, as they often rely on very specific timing conditions.

\* To prevent race conditions, techniques like synchronization mechanisms are used (mutexes, semaphores, etc.) which ensure that only one thread or process can modify a shared resource at a time.