Operating Systems? What is it?

**Q1. What is application and System Software?  
Ans.**

**Application Software** 🡪 software that performs specific functions for a user. The sole purpose of application software is to assist the user in doing specified tasks. Eg. Firefox, Excel etc.

**System Software** 🡪 System software is software that provides a platform for other software. Some examples can be operating systems, antivirus software, disk formatting software, computer language translators, etc.

**Q2. What is OS? What are its goals?**

**Ans.**

1. An operating system acts as an interface between the software and different parts of the computer or the computer hardware.
2. An **operating system** is a piece of software that manages all the resources of a computer system, both  hardware  and  software,
3. It provides  an  environment  in  which  the  user  can  execute  his/her programs in a convenient and efficient manner
4. It hides the underlying complexity of the hardware and acting as a resource manager.

**Goals of OS 🡪**

* Provide the means for proper use of the resources in the operation of

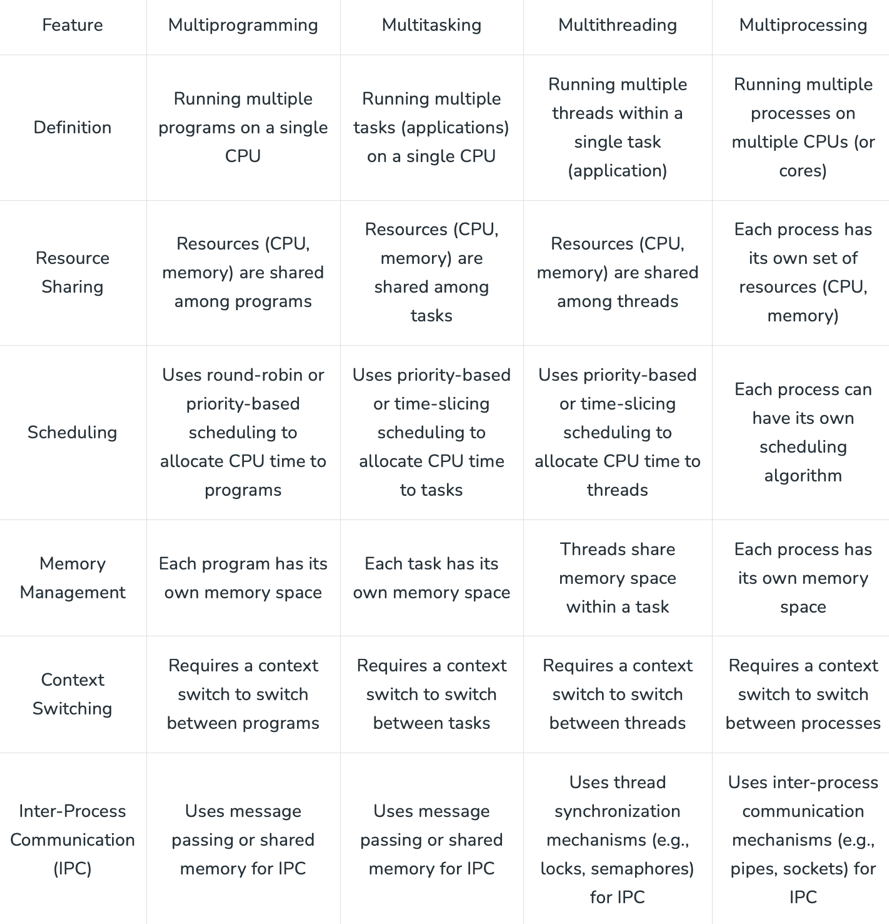
the computer system.

* Maximize CPU utilization.
* Less process starvation
* Process execution as per priority

**Q3. What will happen if there is no OS?**

**Ans.**

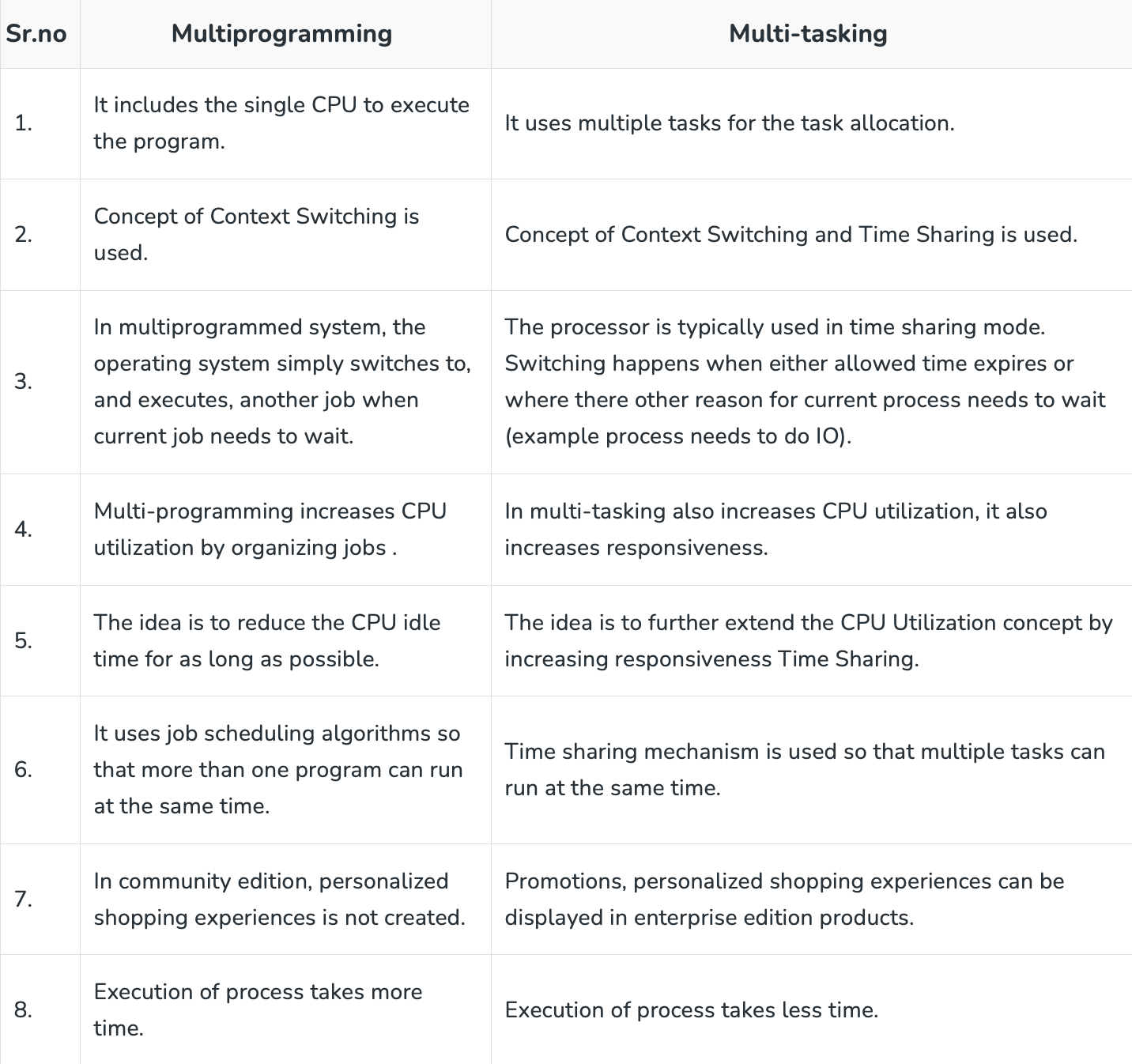
1. All codes for hardware interaction must be in the application being run.
2. There is no memory protection.
3. All resources might get hijacked by a single application because there is no memory allocating system in between.

**Q4. What are the types of OS?**

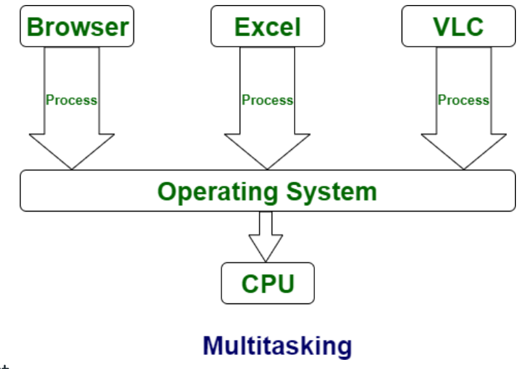
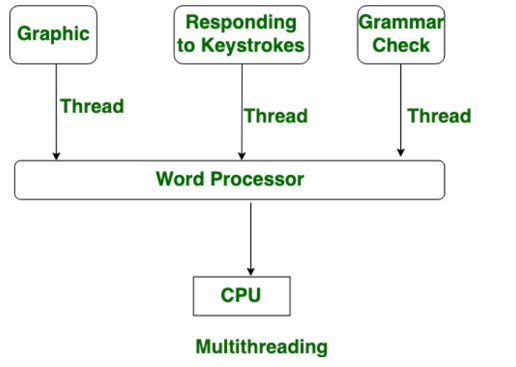
**Ans.**

1. **Single Process OS 🡪**
   1. MS DOS
2. **Batch Process OS 🡪**
3. It is the responsibility of the operator to sort jobs with similar needs.
4. It can manage large work repeatedly in batch systems.
5. ATLAS
6. **Multiprogramming OS 🡪** 
   1. more than one program is present in the main memory and any one of them can be kept in execution. Single CPU
7. **Multiprocessing OS 🡪**
   1. more than one CPU is used for the execution of resources.
   2. As it has several processors, so, if one processor fails, we can proceed with another processor. Multiple CPU
8. **Multitasking OS 🡪**
   1. It is simply a multiprogramming Operating System with having facility of a Round-Robin Scheduling Algorithm. It can run multiple programs/tasks simultaneously.
9. **Distributed System 🡪**
   1. Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as loosely coupled systems or distributed systems.
   2. Failure of one will not affect the other network communication, as all systems are independent of each other.
10. **Real Time OS 🡪**
    1. 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.
    2. Used in missile systems, air traffic control systems, robots, etc.
    3. It can be further divided into Hard Real Time and Soft Real Time Systems. In case of Hard Real Time constraints are very strict and even shortest delay is not permitted.

**Q5. Difference between Multiprogramming and Multitasking.**

**Ans.**

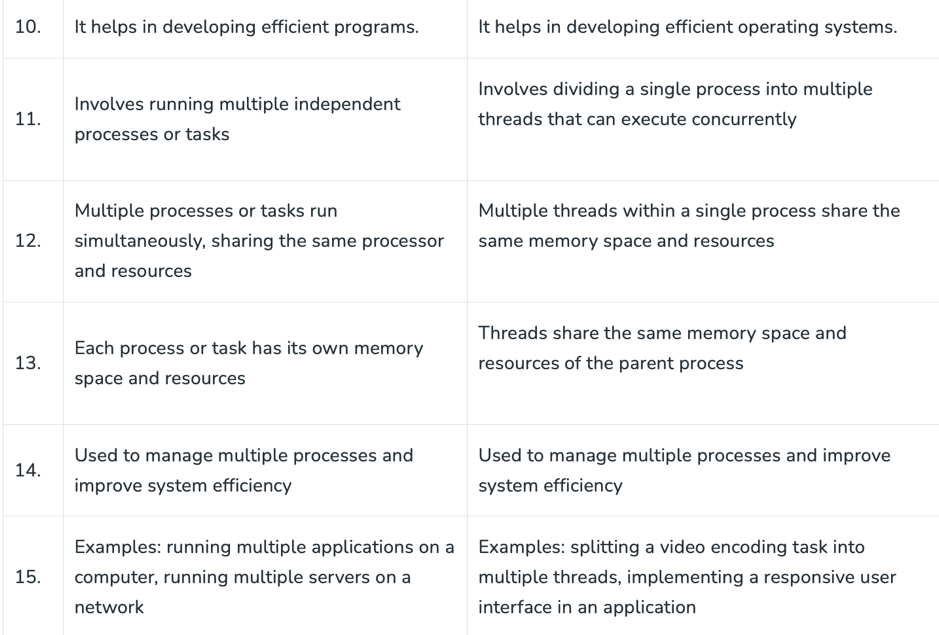
* Multitasking is the ability of an operating system to switch rapidly between multiple tasks or processes on a single CPU, giving the illusion of parallel execution.
* Multiprogramming refers to the technique where multiple programs are loaded into memory simultaneously, with the CPU switching between them as needed.

**Q6. Multitasking and Multithreading.**

**Ans.**

**Multitasking** is when a CPU is provided to execute multiple tasks at a time. Multitasking involves often CPU switching between the tasks. Only one CPU.

**Multithreading** is a system in which many threads are created from a process through which the computer power is increased. In multithreading, CPU is provided in order to execute many threads from a process at a time. More than one CPU.



Context switching is like texting while studying, looks like doing multiple task at same time but not both at same time.

\*\*component of OS 🡪 Kernel Space + User Space

**Q7. What is a kernel?**

**Ans.**

1. It is central component of an operating system that manages operations of computer and hardware.
2. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.
3. Interacts directly with hardware, works as per the commands from user space.

**Q8. What are the objectives of Kernels?**

**Ans.** Basically it looks around 4 objectives 🡪

1. **Process Management 🡪**
2. Process creation and termination. process and thread scheduling.
3. sync process and communication between processes as the process are independent.
4. **Memory Management 🡪** Allocate and deallocate memory.
5. **File Management 🡪** create or delete file, directory management.
6. **IO Management 🡪** Management and controlling the IO devices.

\*\*A shell, also known as a command interpreter, is that part of the operating system that receives

commands from the users and gets them executed.

**Q9. What are the types of kernels?**

**Ans.**

1. **Monolithic Kernel 🡪**

* All 4 functions are in kernel itself.
* Eg. Unix, Linux, Open VMS, XTS-400 etc.
* Faster than other types because they don’t have to switch between user and kernel mode for every system call. They are simpler to implement.
* It is less reliable because if any bug or vulnerability can affect the whole system.

1. **Micro Kernel 🡪**

* Only major function is in kernel i.e. Memory management and Process management.
* Slower but more reliable and stable.
* Slow because it has to switch between user and kernel mode.
* Eg. Linux, Symbian OS etc.

1. **Hybrid Kernel 🡪**

* Combined approach of both the above systems.
* File management is put in user space and rest 3 are stored in the kernel space.
* Eg. Windows and MAC OS.

1. **Nano Kernel**
2. **Exo Kernel**

**\*\*Transitions from US to KS done by software interrupts.**

**Q10. How will communication happen between user mode and kernel mode?**

**Ans.**

Inter process communication (IPC).

1. Two processes executing independently, having independent memory space (Memory protection), But some may need to communicate to work.
2. Done by shared memory and message passing.

**Q11. What is the difference between micro kernel and macro kernel?**

**Ans.**

**Micro kernel:** micro kernel is the kernel which runs minimal performance affecting services for operating system. In micro kernel operating system all other operations are performed by processor.

**Macro Kernel:** Macro Kernel is a combination of micro and monolithic kernel.

**Q12. What is User Space? 🡪**

**Ans.**

No hardware access. Convenient environment for user apps interacted using GUI / CLI , interacts with kernel.

**Q13. What are System Calls and why are they used?**

**Ans.**

It is the only way through which a process can go into kernel mode from user mode.

**System call** is a programmatic way in which a computer program requests a service from the kernel of the operating system. User programs typically do not have permission to perform operations like accessing I/O devices and communicating other programs.

**Q14. Types of System Calls.**

**Ans.**

1. Process Control
2. File Management
3. Device Management
4. Information Maintenance
5. Communication Management

**Q15. What happens when the computer turns ON?**

**Ans.**

1. After the power supply is provided to the computer CPU loads BIOS(Basic Input Output System) or UEIF(Unified Extensible Firmware Interface). BIOS is a ROM storage that provides the computer with data at very basic level.
2. CPU is initialized and BIOS chip is loaded which runs tests and initialize hardware and loads settings from a specific memory( Powered by CMOS battery).
3. Runs power on self- tests.
4. BIOS handovers the control to the boot device(Boot Loader) stored in HDD/SDD.
5. The execution of Bootloader starts the actual booting of the OS( kernel and then user space).

**Q16. What does 32-bit and 64-bit OS mean?**

**Ans.**

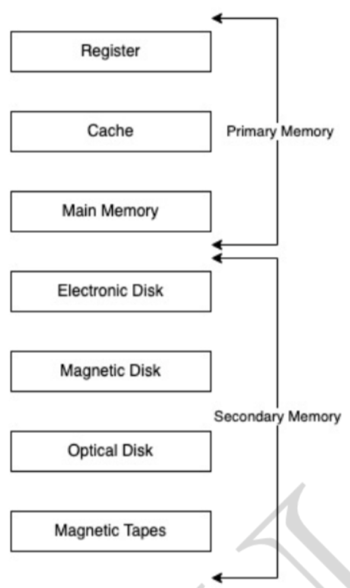
**A 32-bit OS** has 32-bit registers, and it can access 2^32 unique memory addresses. i.e., 4GB of physical memory. 32-bit CPU architecture can process 32 bits of data & information.

**A 64-bit OS** has 64-bit registers, and it can access 2^64 unique memory addresses. i.e., 17,179,869,184 GB

of physical memory. 64-bit CPU architecture can process 64 bits of data & information.

\*\*64-bit CPU can run both 32- and 64-bit OS.

**Q17. Types of memory in computer system.**

**Ans.**

1. Register: Smallest unit of storage. It is a part of CPU itself. A register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters). Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.
2. Cache: Additional memory system that temporarily stores frequently used instructions and data for quicker processing by the CPU.
3. Main Memory: RAM.
4. Secondary Memory: Storage media, on which computer can store data & programs.

**Comparison**

1. **Cost:**
2. Primary storages are costly.
3. Registers are most expensive due to expensive semiconductors & labour.
4. Secondary storages are cheaper than primary.

**2. Access Speed:**

1. Primary has higher access speed than secondary memory.
2. Registers has highest access speed, then comes cache, then main memory.

**3. Storage size:**

1. Secondary has more space.

**4. Volatility:**

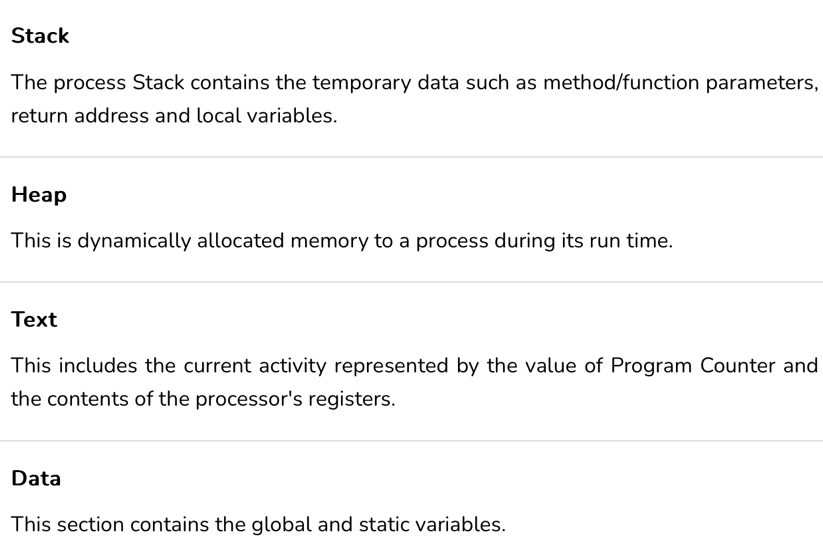
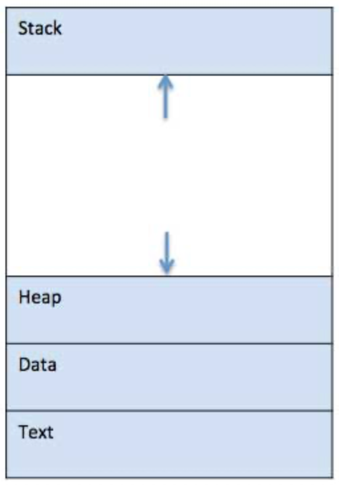
1. Primary memory is volatile.
2. Secondary is non-volatile.

\*\* all the process are created using fork(). Because almost all the process are child. Init is the first process and its PID is 1.

**Q18.What is a process and thread?**

**Ans.**

A program under execution is called process.



**Steps of creating a process 🡪**

1. Load the program & static data into memory. 🡪 for allocate static data
2. Allocate runtime stack.
3. Heap memory allocation 🡪 for allocating dynamic data.
4. IO tasks 🡪 input / output / error handles
5. OS handoffs control to main ().

Concurrency is the feature of the OS by which multiple instructions can be executed at once. It happens in the operating system when there are several process threads running in parallel.

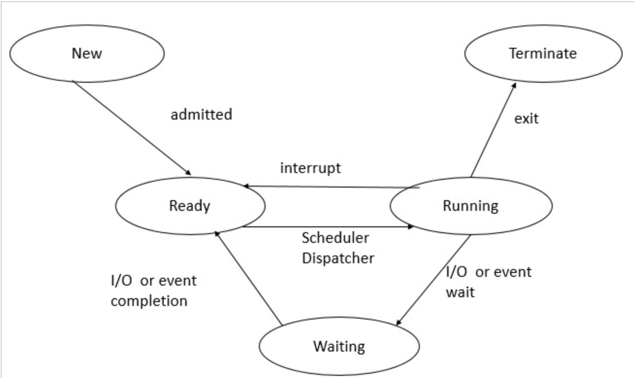
**Thread:**

* Single sequence stream within a process.
* An independent path of execution in a process.
* Light-weight process.
* Used to achieve parallelism by dividing a process’s tasks which are independent path of execution.
* E.g., Multiple tabs in a browser, text editor (When you are typing in an editor, spell checker, formatting etc.)
* Useful only when we have multiple cores in the CPU so that multiple threads can be allocated to different cores.

\*\*Chrome uses multiprocessing and Firefox uses multithreading. So, each tab in chrome is a process whereas in Firefox each tab is a thread. Since multithreading is faster, Firefox is faster(due to shared memory, context switching is faster in threads).

**Q19. What are the Process States?**

**Ans.** As process executes it changes state. Each process may be in one of the following states 🡪

1. **New 🡪** OS is about to pick the program and convert it into process or the process is being created.
2. **Ready 🡪** The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.
3. **Run 🡪** Instructions are being executed and CPU is allocated.
4. **Waiting 🡪** The process is in memory waiting to be assigned to a processor.
5. **Terminated 🡪** The process has finished execution. PCB entry is removed from the process table.

\*\*Job scheduler puts the process from new to ready state and CPU scheduler puts the process from ready to running state.

\*\* thread have only three states ready, running and blocked.

**Q20. What is ready queue?  
Ans.** Ready queue 🡪

1. Process in ready state.
2. Present in main memory
3. CPU scheduler picks the process from ready queue and dispatch it to CPU.

**Q21. What is PCB?**

**Ans.** Data Structure used for each process that stores information of a process such as process id, program counter, process state, priority etc.

**Q22. What is context switching?**

**Ans.**

1. It is a technique/method used by the OS to switch processes from a given state to another one for the execution of its function using the CPUs present in the system.
2. Used in case a running process is engaged in IO
3. The old running process’s status is stored as registers, and the CPU is assigned to a new process for the execution of its tasks.
4. While new processes are running in a system, the previous ones must wait in the ready queue.

\*\*If we print the parent’s PID of a orphan process its 1(Init).

**Q23. What is Orphan process?**

**Ans.** Let us have a process P1 and it forks another process P2 but due to some faults the P1 has to terminate. In this case there is no parent of P2 this situation is handled by OS by assigning Init(1st process of OS) as its parent this is called orphan process.

\*\* when a child process is called the parent calls wait() to extract the exit status of it so that if the child runs successfully the parent can remove the child process from the process table.

\*\*When we run ps -al we see Z (zombie process) , I (idle), R(running or runnable), S(interruptible sleep)

**Q24. What is Zombie process?**

**Ans.**

Process is not working but exists. Let us have a process P1 and it forks another process P2. Let’s say that the parent calls wait() at 10 sec interval but the child completes its execution at 2 secs. In this case the process still exists in the process table because it parent is yet to call wait() but the child has completed its task.

\*\* removal of zombie process from process table is called reaping.

**Q25. What is process scheduling ?**

**Ans.**

It is the basis of Multi-programming OS. Where we use algorithms to choose the process form the ready queue to be executed by the CPU.

Many processes are kept in memory at a time, when a process must wait or time quantum expires, the OS takes the CPU away from that process & gives the CPU to another process & this pattern continues.

Non-Pre-emptive scheduling 🡪

1. Once CPU has been allocated to a process, the process keeps the CPU until it releases CPU either by terminating or by switching to wait-state.
2. Starvation, as a process with long burst time may starve less burst time process.
3. Low CPU utilization.

Pre-emptive scheduling 🡪

1. CPU is taken away from a process after time quantum expires along with terminating or switching to wait-state.
2. Less Starvation
3. High CPU utilization.

Goals of CPU scheduling 🡪

1. Maximum CPU utilization
2. Minimum Turnaround time (TAT).
3. Min. Wait-time
4. Min. response time.
5. Max. throughput of system.

Related terminology 🡪

1. Throughput: No. of processes completed per unit time.
2. Arrival time (AT): Time when process is arrived at the ready queue.
3. Burst time (BT): The time required by the process for its execution.
4. Turnaround time (TAT): Time taken from first time process enters ready state till it terminates. (CT - AT)
5. Wait time (WT): Time process spends waiting for CPU. (WT = TAT – BT)
6. Response time: Time duration between process getting into ready queue and process getting CPU for the first time.
7. Completion Time (CT): Time taken till process gets terminated.

**Q26. Explain FCFS(First come-first serve).**

**Ans.**

Whichever process comes first in the ready queue will be given CPU first. Poor in performance as average wait time is high. It is a non-preemptive algorithm.

**Q27. Explain Convoy Effect.**

**Ans.**

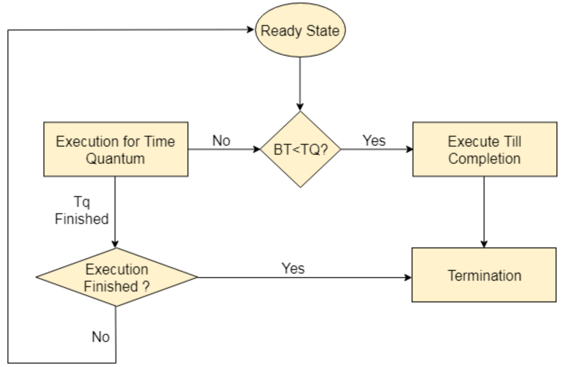
In FCFS, if one process has longer BT. It will have major effect on average WT of different processes, called Convoy effect. Convoy Effect is a situation where many processes, who need to use a resource for a short time, are blocked by one process holding that resource for a long time.

**Q28. Explain SJF( Shortest Job First).**

**Ans.**

1. The process with least burst time is executed first by the CPU.
2. This is a non-pre-emptive, pre-emptive scheduling algorithm. Best approach to minimize waiting time.
3. Easy to implement in Batch systems where required CPU time is known in advance.
4. Impossible to implement in interactive systems where required CPU time is not known.
5. The processer should know in advance how much time process will take.
6. Non-pre-emptive SJF 🡪Process with least Brust Time is executed first. Drawback is that we don’t know the time that a particular process without executing first will, to estimate the BT. Faces Convoy Effect.
7. Pre-emptive SJF 🡪 No Convoy Effect and Less starvation.

**Q29. What is Round Robin Algorithm?**

**Ans.** Round Robin is the pre-emptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a quantum.

1. Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period.
2. Doesn’t depend on Burst time.
3. Context switching is used to save states of pre-empted processes.
4. Cannot be non-pre-empted.
5. It has overheads as we may have to context switch multiple times.

**Q30. What are critical section?**

**Ans.**

Critical Section is any piece of code that is shared between different processes. If more than one process tries to operate in critical section we can reach to undesired output or else if one process starts to operate in critical section and does not release it then it may lead to deadlock, therefore, a single process is allowed to enter at a time.

Eg.1 Whenever a person enters the bathroom, he/she locks the bathroom so that no other person can enter it. Any other person who wants to use the bathroom needs to wait outside till the person comes out.

Eg.2 Tube light with multiple switches. If you switch on a particular switch it may or may not power on the

**Q31. Three required condition for solving for critical section problem.**

**Ans**.

**Mutual Exclusion**:

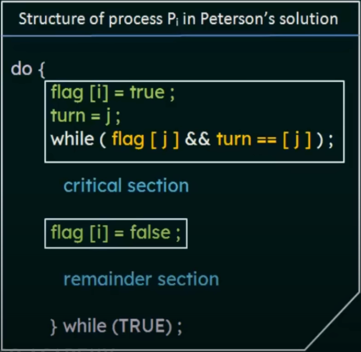
If a process is executing in its critical section, then no other process is allowed to execute in the critical section.

**Progress**:

When no process is executing in its critical section, and there exists a process that wishes to enter its critical section, it should not have to wait indefinitely to enter it. Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

**Bounded Waiting:**

A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.



\*\* using single flag we can implement Mutual exclusion but not Progress

\*\*Petersen’s solution🡪 Alternates the execution of critical and reminder section by the two processes.

Let us have pi that wants to enter the critical section but pj also wants to access it. The while loop here will stop pi from entering the critical section until its condition is false. The condition if false when pj is done its work in critical section and sets flag[j] = false.

**Q32. What is race condition?**

**Ans.** A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

Race condition occurs when two or more threads can access the shared data and try to change it at the same time which can cause inconsistency. We must take care of it or we might get stuck in Deadlocks.

**Q33. Solution to race condition.**

**Ans.**

1. **By making it atomic and executing at once**. We have atomic int in C++. This makes it thread safe.
2. **Mutual exclusion. Using mutex**. This implements locks in which if t1 is executing t2 is not allowed to access critical section.
3. **Semaphores.**

\*\*mutex is internally binary semaphores

**Q33. What are semaphores?**

**Ans.** A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signalled by another thread. This is different than a mutex as the mutex can be signalled only by the thread that is called the wait function.

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

wait(S){

   while (S<=0);

   S--;

}

signal(S){

   S++;

}

There are two types of semaphores: Binary Semaphores and Counting Semaphores.

* **Binary Semaphores:**They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of the mutex semaphore to 0 and some other process can enter its critical section.
* **Counting Semaphores:**They can have any value and are not restricted over a certain domain. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

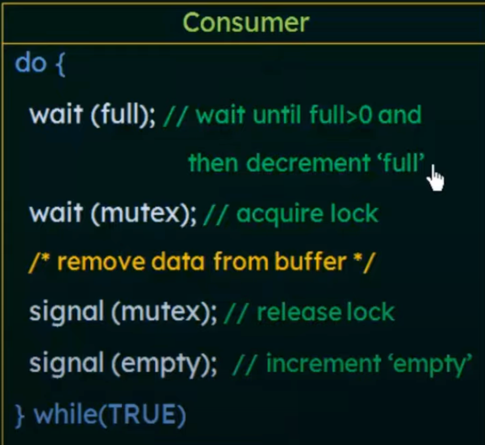
**Q35. Consumer and Producer problem.**

**Ans.**

The producer should produce data only when the buffer is not full. In case it is found that the buffer is full, the producer is not allowed to store any data into the memory buffer. Data can only be consumed by the consumer if and only if the memory buffer is not empty. In case it is found that the buffer is empty, the consumer is not allowed to use any data from the memory buffer

To solve it we will use 3 semaphores.

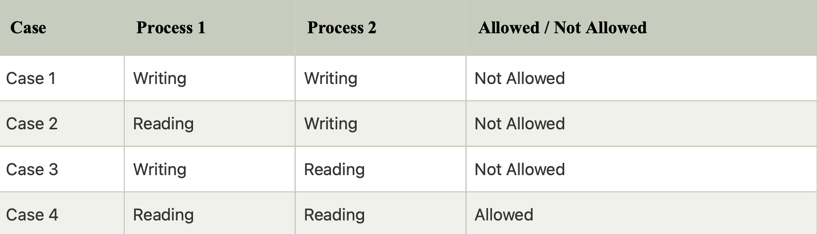
1. m(Mutex) 🡪 A binary semaphore which is used to acquire and release locks. (mutual exclusion)
2. empty 🡪 Counting semaphore, keeping count of empty slots in the buffer.
3. full 🡪 Counting semaphore, keeping count of filled slots in the buffer.

****

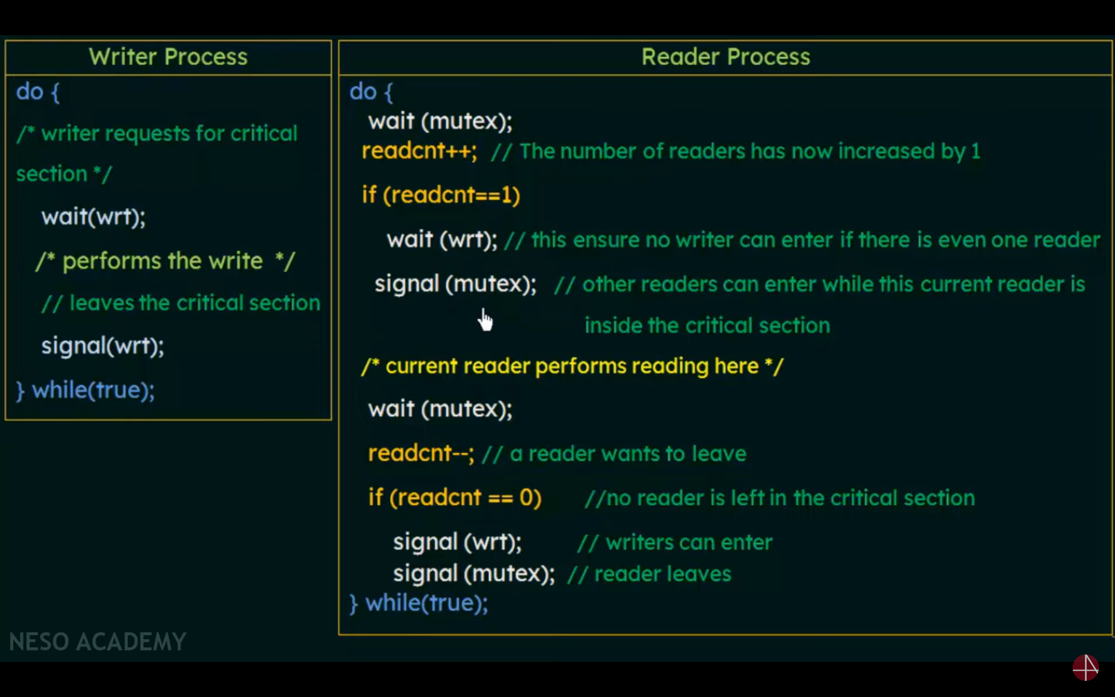
When mutex is locked then consumer is not allowed to change the buffer and hence the mutual exclusion is ensured.

**Q36. Readers and writers’ problem.**

**Ans.**

**Two semaphores and an integer are used in this 🡪**

1. **Mutex 🡪** to acquire a lock, semaphore initialized with 1, used to ensure mutual exclusion when read count is updated.
2. **Write 🡪** semaphore initialized with 1, common to both reader and writer.
3. **Read\_count🡪** an integer keeps count of active readers.

****

**Q37. What are Deadlocks?**

**Ans.**

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

**Deadlock can arise only if all these 4 conditions satisfy** 🡪

1. **Mutual Exclusion:** Two or more resources are non-shareable (Only one process can use at a time)
2. **Hold and Wait**: A process is holding at least one resource and waiting for resources.
3. **No Pre-emption:** A resource cannot be taken from a process unless the process releases the resource.
4. **Circular Wait:** A set of processes waiting for each other in circular form.

**Q38. How to handle Deadlocks?**

**Ans.**

1. Use a protocol to prevent the deadlock
2. Allow the system to fall into deadlock , then detect and remove it.
3. **Deadlock ignorance**🡪Pretend that deadlock never occur in our system.

**Q39. How to prevent the deadlock?**

**Ans.**

Same as the reasons that cause deadlocks. Try to prevent them and deadlock will be prevented.

1. Mutual exclusion
2. Use locks (mutual exclusion) only for non-sharable resource.
3. Sharable resources like Read-Only files can be accessed by multiple processes/threads.
4. However, we can’t prevent DLs by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.
5. Hold & Wait
6. To ensure H&W condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it doesn’t hold any other resource.
7. Protocol (A) can be, each process has to request and be allocated all its resources before its execution.
8. Protocol (B) can be, allow a process to request resources only when it has none. It can request any additional resources after it must have released all the resources that it is currently allocated.
9. No pre-emption
10. If a process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the process is currently holding are pre-empted. The process will restart only when it can regain its old resources, as well as the new one that it is requesting. (Live Lock may occur).
11. If a process requests some resources, we first check whether they are available. If yes, we allocate them. If not, we check whether they are allocated to some other process that is waiting for additional resources. If so, pre-empt the desired resource from waiting process and allocate them to the requesting process.
12. Circular wait
13. To ensure that this condition never holds is to impose a proper ordering of resource allocation.
14. P1 and P2 both require R1 and R1, locking on these resources should be like, both try to lock R1 then R2. By this way which ever process first locks R1 will get R2.

**Q40. What are Logical address and physical addresses?**

**Ans.**

**Logical address** also known as a virtual address, is an address generated by the CPU during program execution. It is the address seen by the process and is relative to the program’s address space. The process accesses memory using logical addresses, which are translated by the operating system into physical addresses. Ranges 🡪 0 to MAX

**Physical address:** A physical address is the actual address in main memory where data is stored. It is a location in physical memory, as opposed to a virtual address. Physical addresses are used by the memory management unit (MMU) to translate logical addresses into physical addresses.

Ranges 🡪 0+R to R+MAX (R is the base value)

**Q41. What are the allocation methods of physical memory?**

**Ans.**

**Contiguous Memory Allocation🡪**

1. Data is stored in single contiguous block of memory.
2. Main memory is divided into equal or different size parts.
3. Internal and External fragmentation.
4. Size of the fragment is an issue.

**Dynamic partitioning 🡪**

1. In this technique, the partition size is not declared initially. It is declared at the time of process loading.
2. No internal fragmentation but suffers external fragmentation.
3. Size of the process doesn’t matter.

\*\*internal fragmentation can be cured by dynamic partitioning.

**Q42. What is fragmentation**

**Ans.**

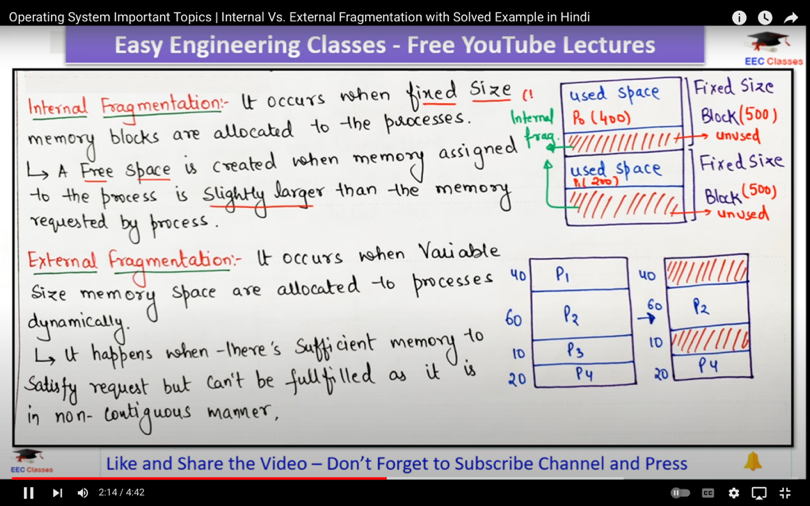
Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused.

\*\* it is a problem in which a program cannot be assigned a memory even if there is space left due to fragmentation**.**

**Q43. Internal vs External fragmentation**

**Ans.**

|  |  |
| --- | --- |
| **Internal Fragmentation** | **External fragmentation** |
| 1.Internal fragmentation occurs when there is unused space within a memory block. | 1.External fragmentation occurs when a storage medium, such as a hard disc or solid-state drive, has many small blocks of free space scattered throughout it. |
| 2.Internal fragmentation occurs when memory is divided into fixed-sized partitions. | 2.External fragmentation occurs when memory is divided into variable size partitions based on the size of processes. |
| 3.The solution of internal fragmentation is the best-fit block. | 3.The solution to external fragmentation is compaction and paging. |

****

**Q44. What is defragmentation?**

**Ans.** Dynamic partitioning suffers from external fragmentation. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is called defragmentation.

\*\* to manage free space OS uses free list a Linked List storing address of free spaces.

**Q45. What is paging?**

**Ans.**

Paging is a memory management scheme that eliminates the need for a [contiguous allocation](https://www.geeksforgeeks.org/difference-between-contiguous-and-noncontiguous-memory-allocation/) of physical memory. The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging.

Ek chiz ko store karne ke liye uske size ka container nahi chaiye ( no need of contiguous allocations).

1. The basic purpose of paging is to separate each procedure into pages.
2. The physical memory is divided into fixed-size blocks called page frames, which are the same size as the pages used by the process. The process’s logical address space is also divided into fixed-size blocks called pages, which are the same size as the page frames.
3. When a process requests memory, the operating system allocates one or more page frames to the process and maps the process’s logical pages to the physical page frames.

Each process has its different page table.

Walkie talkie as an example

Virtual memory creates an illusion of large main memory

Paging vs segmentation