**1) What are the possible state transitions of a process?**

- The process can be in any one of the following three possible states.

+ Running (actually using the CPU at that time and running).

+ Ready (runnable; temporarily stopped to allow another process run).

+ Blocked (unable to run until some external event happens).

**2) What are the differences between a thread and a process?**

- A process is a program under execution i.e an active program. A thread is a lightweight process that can be managed independently by a scheduler. Processes require more time for context switching as they are more heavy. Threads require less time for context switching as they are lighter than processes.

**3) User-level threads vs. Kernel-level threads? Know the pros and cons of each type.**

- A thread is a lightweight process that can be managed independently by a scheduler. It improves the application performance using parallelism.

A thread shares information like data segment, code segment files etc. with its peer threads while it contains its own registers, stack, counter etc.

The two main types of threads are user-level threads and kernel-level threads. A diagram that demonstrates these is as follows

***User - Level Threads***

The user-level threads are implemented by users and the kernel is not aware of the existence of these threads. It handles them as if they were single-threaded processes. User-level threads are small and much faster than kernel level threads. They are represented by a program counter(PC), stack, registers and a small process control block. Also, there is no kernel involvement in synchronization for user-level threads.

***Advantages of User-Level Threads***

Some of the advantages of user-level threads are as follows −

* User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed.
* User-level threads can be run on any operating system.
* There are no kernel mode privileges required for thread switching in user-level threads.

***Disadvantages of User-Level Threads***

Some of the disadvantages of user-level threads are as follows −

* Multithreaded applications in user-level threads cannot use multiprocessing to their advantage.
* The entire process is blocked if one user-level thread performs blocking operation.

***Kernel-Level Threads***

Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel. The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads.

***Advantages of Kernel-Level Threads***

Some of the advantages of kernel-level threads are as follows −

* Multiple threads of the same process can be scheduled on different processors in kernel-level threads.
* The kernel routines can also be multithreaded.
* If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel.

***Disadvantages of Kernel-Level Threads***

Some of the disadvantages of kernel-level threads are as follows −

* A mode switch to kernel mode is required to transfer control from one thread to another in a process.
* Kernel-level threads are slower to create as well as manage as compared to user-level threads.

**4) Understand the concepts of race condition and mutual exclusion**

Race Conditions: two or more processes are reading and writing on shared data and the final result depends on who runs precisely when

Mutual exclusion : making sure that if one process is accessing a shared memory, the other will be excluded form doing the same thing

***+ Locked variables, strict alternation, Peterson's algorithm, disabling interrupts, semaphore, monitor. Know which one is software or hardware solution; which one relies on busy waiting.***

- One is software or hardware solution: Locked variables, strict alternation, Peterson's algorithm

- One relies on busy waiting : disabling interrupts, semaphore, monitor.

***+ Semaphores. Understand what a semaphore is; its up and down operations; its principles to allow mutual exclusion among several processes or threads.***

- **Semaphore** is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) Wait, and 2) Signal for the process synchronization.

A semaphore either allows or disallows access to the resource, which depends on how it is set up.

- **P operation is also called wait, sleep, or down operation, and V operation is also called signal, wake-up, or up operation**. Both operations are atomic and semaphore(s) is always initialized to one.

- To provide mutual exclusion for use of a resource such as a linked list, the processes **create a single semaphore that has an initial count of 1**. Before accessing the shared resource, a process calls wait on the semaphore, and calls signal after it has com- pleted access.

***+ What is the preemption and non-preemption in the context of process scheduling?***

- Key Differences Between Preemptive and Non-Preemptive Scheduling: In preemptive scheduling, the CPU is allocated to the processes for a limited time whereas, in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to the waiting state.

***+ How the FIFO, SJF (SJRF) and Round-Robin scheduling algorithms work?***

- First come first serve (FCFS) scheduling algorithm simply **schedules the jobs according to their arrival time**. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU.

- Shortest Job First (SJF) is an algorithm in which **the process having the smallest execution time is chosen for the next execution**. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

- Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a quantum. Once a process is executed for a given time period, it is preempted and other process executes for a given time period. Context switching is used to save states of preempted processes.