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

There can be various events that lead to a state transition for a process. The possible state transitions are given below:

1. **Null -> New:**A new process is created for the execution of a process.
2. **New -> Ready:**The system will move the process from new to ready state and now it is ready for execution. Here a system may set a limit so that multiple processes can’t occur otherwise there may be a performance issue.
3. **Ready -> Running:** The OS now selects a process for a run and the system chooses only one process in a ready state for execution.
4. **Running -> Exit:**The system terminates a process if the process indicates that is now completed or if it has been aborted.
5. **Running -> Ready:**The reason for which this transition occurs is that when the running process has reached its maximum running time for uninterrupted execution. An example of this can be a process running in the background that performs some maintenance or other functions periodically.
6. **Running -> Blocked:** A process is put in the blocked state if it requests for something it is waiting. Like, a process may request some resources that might not be available at the time or it may be waiting for an I/O operation or waiting for some other process to finish before the process can continue.
7. **Blocked -> Ready:** A process moves from blocked state to the ready state when the event for which it has been waiting.
8. **Ready -> Exit:** This transition can exist only in some cases because, in some systems, a parent may terminate a child’s process at any time.

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

| **Comparison Basis** | **Process** | **Thread** |
| --- | --- | --- |
| Definition | 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. |
| Context switching time | 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. |
| Memory Sharing | Processes are totally independent and don’t share memory. | A thread may share some memory with its peer threads. |
| Communication | Communication between processes requires more time than between threads. | Communication between threads requires less time than between processes . |
| Blocked | If a process gets blocked, remaining processes can continue execution. | If a user level thread gets blocked, all of its peer threads also get blocked. |
| Resource Consumption | Processes require more resources than threads. | Threads generally need less resources than processes. |
| Dependency | Individual processes are independent of each other. | Threads are parts of a process and so are dependent. |
| Data and Code sharing | Processes have independent data and code segments. | A thread shares the data segment, code segment, files etc. with its peer threads. |
| Treatment by OS | All the different processes are treated separately by the operating system. | All user level peer threads are treated as a single task by the operating system. |
| Time for creation | Processes require more time for creation. | Threads require less time for creation. |
| Time for termination | Processes require more time for termination. | Threads require less time for termination. |

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

## **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

+ 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.

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

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

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