What is a process?

A process is a program in execution. It is more than a program code called a text section. This concept works under all operating systems because all the task performs by the operating system needs a process to perform the task. The process executes when it changes the state. To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program

What is PCB?

A process control block (PCB) is a data structure used by computer operating systems to store all the information about a process. It is also known as a process descriptor. When a process is created (initialized or installed), the operating system creates a corresponding process control block.

Define different types of schedulers.

A scheduler is a system software that allows you to handle process scheduling. Three types of schedulers are 1) Long term 2) Short term 3)Medium-term.

The long-term scheduler regulates the program and selects processes from the queue and loads them into memory for execution.

Medium-term scheduling is a part of the swapping function. This is a decision on whether to add a process to those that are at least partially in the main memory and therefore available for execution.

A short-term scheduler is also known as a CPU scheduler. Its main objective is to boost the system performance according to certain set criteria.

Which schedule is responsible for the process mix and how?

Long-Term Scheduler. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming.

What is context switching in OS?

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

What are the advantage and disadvantages of too much context switching?

The disadvantage of context switching is that it requires some time for context switching, also called context switching time. Time is required to save the context of one process that is in the running state and then get the context of another process that is about to come into the running state.

The main advantage of context switching is, even if the system contains only one CPU, it gives the user an illusion that the system has multiple CPUs due to which multiple processes are being executed. Context switching is so fast that the user won't even realize that the processes are switched to and from.

Why do we use thread over process?

We can prefer multiple threads over multiple processes for two reasons:

Inter-thread communication (sharing data etc.) is significantly more straightforward to program Than inter-process communication.

Another advantage of using a thread over a process is related to context switching. Context switching between threads is much faster than context switching between processes. Context switching refers to how the system switches from running one process or thread to another running process or thread

How can a thread be utilized for task parallelism?

In many cases, you can achieve parallelism by forking a few threads to do the work. The Async API can help by propagating errors appropriately and cleaning up threads.

What are the types of threads?

There are two types of threads to be managed in a modern system: User threads and kernel threads. User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs. Kernel threads are supported within the kernel of the OS itself.

What are the limitations of the user thread?

Multithreaded applications in user-level threads cannot use multiprocessing to their advantage. The entire process is blocked if one user-level thread performs a blocking operation. User-level threads are invisible to the OS. They are not well integrated with the OS

How these limitations can be solved?

Solving this requires communication between the kernel and the user-level thread manager.