In a computer with single core processor, only one task is said to be running at any point of time.

Now, I have doubts. When I’m hearing music, at the same time I’m converting some video into mobile format and at same time editing my document in my single core Pentium processor. So more than one task is running. But I didn’t feel as if only one task or process was running.

I see that all the three tasks are running without interruption and I even think they run in parallel. If only one task was running at any point and also if CPU switched between various processes or tasks, then when I would hear music without interruption and if CPU switched one task to other, another task would have to wait for its turn and that time the music playback or video conversion may stop, but I didn’t feel that. So help me on this topic. Thanks in advance.

Multithreading means exactly that, running multiple threads. This can be done on a uni-processor system, or on a multi-processor system.

On a single-processor system (a system which has a CPU with only one core), when running multiple threads, the actual observance of the computer doing mulitple things at the same time (i.e., multi-tasking) is an illusion, because what's really happening under the hood is that there is a software scheduler performing time-slicing on the single CPU. So only a single task is happening at any given time, but the scheduler is switching between tasks fast enough so that you never notice that there are multiple processes, threads, etc., contending for the same CPU resource.

On a multi-processor system (a system which has a CPU with multi cores, or a system which has multi CPUs), the need for time-slicing is reduced. The time-slicing effect is still there, because a modern OS could have hundred's of threads contending for two or more processors, and there is typically never a 1-to-1 relationship in the number of threads to the number of processing cores available. So at some point a thread will have to stop and another thread start on a CPU that the two threads are sharing. This is again handled by the OS's scheduler. That being said, with a multiprocessors system, you can have two things happening at the same time, unlike with the uni-processor system.

In the end, the two paradigms are really somewhat orthogonal in the sense that you will need multithreading whenever you want to have two or more tasks running asynchronously, but because of time-slicing, you do not necessarily need a multi-processor system to accomplish that. If you are trying to run multiple threads, and are doing a task that is highly parallel (i.e., trying to solve an integral), then yes, the more cores you can throw at a problem, the better. You won't necessarily need a 1-to-1 relationship between threads and processing cores, but at the same time, you don't want to spin off so many threads that you end up with tons of idle threads because they must wait to be scheduled on one of the available CPU cores. On the other-hand, if your parallel tasks requires some sequential component, i.e., a thread will be waiting for the result from another thread before it can continue, then you may be able to run more threads with some type of barrier or synchronization method so that the threads that need to be idle are not spinning away using CPU time, and only the threads that need to run are contending for CPU resources.