Back in the old days a computer had a single CPU, and was only capable of executing a single program at a time.

Later came multitasking which meant that computers could execute multiple programs (AKA tasks or processes) at the same time. It wasn't really "at the same time" though. The single CPU was shared between the programs.

OS switch between the programs running, executing each one of them before switching

Challenges

Programs can no longer assume to have all the CPU time available, nor all memory or any other computer resources

We need to clear resources , so others can use.

It is even more challenging , that the threads are executing within the same program and are hence reading and writing the same memory simultaneously..

This can result in errors not seen in a single threaded program. Some of these errors may not be seen on single CPU machines, because two threads never really execute "simultanously"

If a thread reads a memory location while another thread writes to it, what value will the first thread end up reading? The old value? The value written by the second thread? Or a value that is a mix between the two? Or, if two threads are writing to the same memory location simultanously, what value will be left when they are done? The value written by the first thread? The value written by the second thread? Or a mix of the two values written? Without proper precautions any of these outcomes are possible. The behaviour would not even be predictable. The outcome could change from time to time.