Introduction#

A system can achieve concurrency by employing one of the following multitasking models:

- Preemptive Multitasking
- Cooperative Multitasking

Preemptive Multitasking#

In preemptive multitasking, the operating system preempts a program to allow another waiting task to run on the CPU. Programs or threads can't decide how long for or when they can use the CPU. The operating system’s scheduler decides which thread or program gets to use the CPU next and for how much time. Furthermore, scheduling of programs or threads on the CPU isn’t predictable. A thread or program once taken off of the CPU by the scheduler can't determine when it will get on the CPU next. As a consequence, if a malicious program initiates an infinite loop, it only hurts itself without affecting other programs or threads. Lastly, the programmer isn't burdened to decide when to give up control back to the CPU in code.

Cooperative Multitasking#

Cooperative Multitasking involves well-behaved programs to voluntarily give up control back to the scheduler so that another program can run. A program or thread may give up control after a period of time has expired or if it becomes idle or logically blocked. The operating system’s scheduler has no say in how long a program or thread runs for. A malicious program can bring the entire system to a halt by busy waiting or running an infinite loop and not giving up control. The process scheduler for an operating system implementing cooperative multitasking is called a cooperative scheduler. As the name implies, the participating programs or threads are required to cooperate to make the scheduling scheme work.

Cooperative vs Preemptive#

Early versions of both Windows and Mac OS used cooperative multitasking. Later on preemptive multitasking was introduced in Windows NT 3.1 and in Mac OS X. However, preemptive multitasking has always been a core feature of Unix based systems.