2.1 Scheduler and threads

Threads: To introduce the explanation about the scheduler, it is important to differentiate user-level thread (ULT also mentioned as "threads" in the literature) and kernel-level threads (KLT). A KLT will be attributed processor usage by the operating system's (OS) scheduler, competing against other processes on the system. It is said to be a one-to-one mapping model (1:1). The ULT hoverer, are scheduled by a VM or other mechanism, these types of threads are mapped in a many-to-many or many-to-one models (M:N, M:1). For these threads, they are competing between each other (in a M:1 model), scheduled by another mechanism than the OS scheduler. [2]

The scheduler: It is a mechanism that manages how programs interact with the processor. It has different ways to achieve its function, but its primary goal is to address processes' access to the CPU in an efficient manner. The scheduling disciplines for a thread to be managed by the scheduler are: first come first serve, round robin, shortest first etc...

When the CPU is single-threaded then the scheduler switches processes on the CPU fast enough to give the illusion that the computer handles everything in parallel, which it doesn't. A scheduler on a multi-threaded machine will spread the different processes among the multiple cores and logical processor available, the scheduler has to be fast in its decisions and fair so that all the processes end up being executed. To help, there could also be a scale to differentiate importance between threads, so the scheduler take this parameter into account with the 'nice' value for Linux processes for instance. To make sure all the tasks are executed and efficiently distributed, a scheduler is needed so all cores and logical processor are continuously being used in the most effective manner.

Some important details about the scheduler for this study, is that the scheduler manages multiple queues, specially a waiting queue and a competing queue. A thread that is doing computation on a core is running, and when it gets lock, it will be placed in the waiting queue until it receives an unlock signal, which will place it in the competing queue to "acquire" a core again and to let it continue its computation. This mechanism along with choosing which thread has to run, is not instant and can cause delay in program execution.

About pthread and Linux scheduler precisely: Posix Thread is an interface followed to implement threads in a consistent manner between OS. On Linux, this is the Native POSIX Thread Library (NPTL) that implements pthread by doing a "clone()" system call from the Linux standard library's when starting a thread.

When a thread is started it is a 1:1 correspondence with a schedulable entities of the kernel. This means that each thread created by a pthread on Linux

will be directly handled by the Linux scheduler with all the other processes which make them compete for the CPU at the same level as every other process of the Linus System. The default status of a process is SCHED_OTHER where threads execution are time-sliced and fairly shared on the CPU.

The default "mode" given to processes can differ between OS, and on Linux a process scheduling policy can be modified to SCHED_FIFO or SCHED_RR (with root authority, as it could result in starvation of other processes if only FIFO threads are created).