

Lecture 05.01 Processing threads

threads A processing *thread* is a sequences of instructions to be executed by the CPU. One or more threads typically comprise a *process*. Threads can share the same memory space and other resources, but processes are typically independent.

processes

scheduler The computer operating system's *scheduler* controls the execution of processes and threads. For most modern processors, each *core* can handle two threads by sharing a core, which swaps back-and-forth between the threads—called *simultaneous multithreading* (SMT) or time-slicing.¹ A schematic of this process is shown in Figure 05.1.

CPU cores

simultaneous multithreading

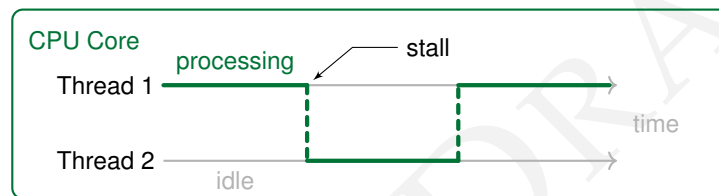


Figure 05.1: a schematic of simultaneous multithreading (SMT) in a single CPU core.

stalls Although the core is not actually simultaneously processing the threads, there is frequently an overall speedup by exploiting *stalls* in the thread such as *cache misses*: when a thread requires data not available in the CPU caches and must wait for the data from some relatively slow source, such as the main memory. When there is a cache miss in one thread, the other can execute in what would have otherwise been stalled processing time. Frequently, a single program will make use of multiple threads.

cache misses

Pthreads UNIX-based operating systems, such as the NI Linux Real-Time OS of the myRIO, have a standardized (IEEE POSIX 1003.1c) threading language in C called *POSIX threads* (Pthreads). This widely used interface is incorporated into header files of the C library provided with the myRIO. This is how we will implement threading in our programs from [Lab Exercise 05](#), on.

The ARM Cortex A-9 processor of the myRIO has two cores and is capable of handling multiple threads.

¹Intel uses the term “hyperthreading” for SMT.