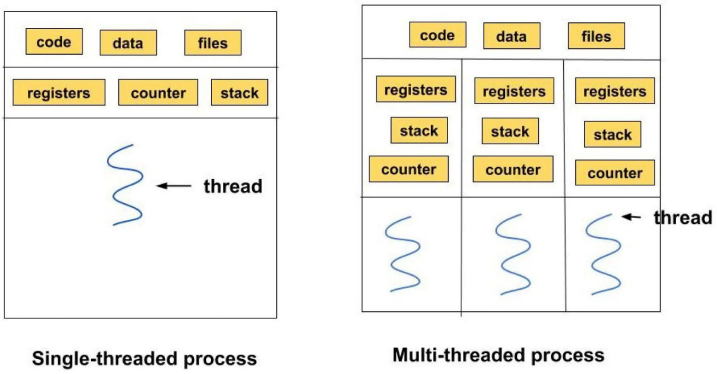
Operating Systems (CS F372)
Tutorial Sheet 6
Threads

In this tutorial sheet, we shall explore the concepts of threads and multithreading.



What is a thread, and how is it different from a process? (Read)

A thread is a single sequence stream within a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. Threads are not independent from each other, unlike processes. As a result, threads share with other threads their code section, data section, and OS resources like open files and signals. But, like processes, a thread has its own program counter (PC), a register set, and a stack space.

When to use multithreading?

In computational tasks where the inherent computation is trivial, such as finding a single element in an array or calculating the sum of array elements, non-threaded execution often outperforms threaded execution. This is primarily because the computational overhead of creating and managing threads can outweigh any potential performance gains. In such scenarios, the simplicity of non-threaded processes allows them to efficiently handle these lightweight operations.

However, the advantage of threading becomes prominent when dealing with computationally intensive tasks that can be parallelized effectively. For instance, in applications involving complex data processing, large-scale matrix operations, or simulations of concurrent systems, threading can offer significant performance enhancements.

You can use the time.h library to measure the runtimes of your programs and compare the execution times between threaded and non-threaded programs.

Note: Multithreading is often used in programs with shared data structures or files involved. Once some threads are created, we do not have control over which thread executes when, and hence, at times, it might be necessary to control or restrict access to the shared resource. This is where the concept of synchronization comes in. We shall explore this concept in the upcoming sheets.

Questions

Note: Use the -pthread flag during compilation. Ex. `gcc thread1.c -o thread1 -pthread`

1. The purpose of this exercise is to familiarize you with the basics of multithreading in C and how threads can be used to execute concurrent tasks.

   Write a C program that utilizes multithreading to spawn some fixed number of threads. Each thread should print a message indicating its index(1st thread, 2nd thread, etc.).

   Read: What pthread_join *does not* do

2. This exercise aims to illustrate how multithreading can be used to efficiently fill a data structure in parallel, demonstrating its advantages in scenarios involving data manipulation and generation.

   Develop a C program that creates a pattern of asterisks in a pyramid shape using multithreading. Each thread should be responsible for printing a row of asterisks, where the number of asterisks in each row corresponds to the thread's index. For example, the first thread prints a single asterisk, the second thread prints two asterisks, and so on. The number of lines in the pyramid is taken as a command-line argument.

3. This exercise aims to demonstrate the concept of dividing a file-reading task into multiple threads, each responsible for a portion of the file.

   Develop a C program capable of efficiently reading and processing integers from a text file. The file contains integers, each residing on a separate line. Your task involves creating three distinct threads, each responsible for reading and processing exactly one-third of the integers from the file concurrently(for simplicity, assume total_numbers%3 = 0). As integers are processed, display their values alongside the thread number (0, 1, or 2). The file name and the number of integers in the file are taken as a command-line argument.

   Hint: Use the **fseek** function.