FIT3143 Lab Week 3

Lecturers: ABM Russel (MU Australia) and Vishnu Monn (MU Malaysia)

THREADS

OBJECTIVES

- The purpose of this lab is to introduce you to POSIX Threads
- Practice POSIX Threads

INSTRUCTIONS

- Download and set up the Linux VM [Refer to Lab Week 1]
- Setup eFolio (including Git) and share with tutor and partner [Refer to Lab Week 1]

TASK

DESCRIPTION:

- Become familiar with POSIX Threads

WHAT TO SUBMIT:

1. Screenshot of the running programs and git repository URL in the eFolio. The eFolio should also include analysis and discussions based on the requirements of the lab question.
2. Code in the Git.

EVALUATION CRITERIA

- This Lab-work is part of grading
- Code compile without errors (2), well commented (2), lab-work questions fully answered (4), analysis/report is well formatted (2) = 10 marks

LAB ACTIVITIES

1. POSIX Threads

a) Write a serial C program to search for prime numbers which are less than an integer, *n*, which is provided by the user. The expected program output is a list of all prime numbers found, which is written into a single text file (e.g., *primes.txt*).

For instance, if the user inputs *n* as 10 on the terminal, the prime numbers written to the text file are: 2, 3, 5, 7.

Hint: It is known that for a given number, prime numbers would only exist for values less than or equal to the square root of the given number. As such, you can make use of the sqrt() of math.h to optimize your code.

b) Measure the time required to search for prime numbers less than an integer, *n* when *n = 1,000,000* (i.e., t_s). Calculate the theoretical speed up using Amdahl's law when *p = 4*, with *p = number of processes (or threads)*.

Amdahl's law, s(p):

$$s_{theory}(p) = \frac{1}{r_s + \frac{r_p}{p}}$$

where r_p is parallel ratio (parallelizable portion) of the algorithm, r_s is the serial ratio (non-parallelizable portion) of the algorithm and *p* is the number of processes (or threads).

c) Write a parallel version of your serial code in C utilizing POSIX Threads. Here, design and implement a parallel partitioning scheme which distributed the workload among the threads. Compare the performance of the serial program (t_s) in part (a) with that of the parallel program (t_p) in part (b) for a fixed number of processors or threads (e.g., *p* = *4*). Calculate the actual speed up, *S(p)*. Analyze and compare the actual speed up with the theoretical speed up for an increasing number of threads and/or *n*. You can either tabulate your results or plot a chart when analyzing the performance of the serial and parallel programs.