COP 4520 Spring 2020

Programming Assignment 1

Note 1:

Please, submit your work via Webcourses.

Submissions by e-mail will not be accepted.

Due date: Monday, January 27th by 11:59 PM

Late submissions are not accepted.

Note 2:

This assignment is individual.

You can use a programming language of your choice for this assignment.

If you do not have a preference for a programming language, I would recommend C++.

Problem 1 (40 points)

Your non-technical manager assigns you the task to find all primes between 1 and 10^8. The assumption is that your company is going to use a parallel machine that supports eight concurrent threads. Thus, in your design you should plan to spawn 8 threads that will perform the necessary computation. Your boss does not have a strong technical background but she is a reasonable person. Therefore, she expects to see that the work is distributed such that the computational execution time is approximately equivalent among the threads. Finally, you need to provide a brief summary of your approach and an informal statement reasoning about the correctness and efficiency of your design. Provide a summary of the experimental evaluation of your approach. Remember, that your company cannot afford a supercomputer and rents a machine by the minute, so the longer your program takes, the more it costs. Feel free to use any programming language of your choice *that supports multi-threading* as long as you provide a ReadMe file with instructions for your manager explaining how to compile and run your program from the command prompt.

Required Output:

Please print the following output to a file named primes.txt:

<execution time> <total number of primes found> <sum of all primes found>
<top ten maximum primes, listed in order from lowest to highest>

Notes on Output:

1. Zero and one are neither prime nor composite, so please don't include them in the total number of primes found and the sum of all primes found.

2. The execution time should start prior to spawning the threads and end after all threads complete.

Grading policy:

General program design and correctness: 50%

Efficiency: 30%

Documentation including statements and proof of correctness, efficiency, and experimental evaluation: 20%

Additional Instructions:

Cheating in any form will not be tolerated. Please, submit your work via webcourses.

- In addition to being parallel, your design should also make use of an efficient algorithm for finding prime numbers.

Problem 2 (60 points)

The Dining Philosophers problem was invented by E. W. Dijkstra, a concurrency pioneer, to clarify the notions of deadlock and starvation freedom. Imagine five philosophers who spend their lives just thinking and feasting. They sit around a circular table with five chairs. The table has a big plate of rice. However, there are only five chopsticks (in the original formulation forks) available (see Figure 1 of Chapter 1, Exercise 1 from the textbook). Each philosopher thinks. When he gets hungry, he sits down and picks up the two chopsticks that are closest to him. If a philosopher can pick up both chopsticks, he can eat for a while. After a philosopher finishes eating, he puts down the chopsticks and again starts to think.

1. Write a program (Version 1) to simulate the behavior of the philosophers, where each philosopher is a thread and chopsticks are shared objects. Notice that you must prevent a situation where two philosophers hold the same chopstick at the same time.

2. Write a program (Version 2) that modifies Version 1 so that it never reaches a state where philosophers are deadlocked, that is, it is never the case that each philosopher holds one chopstick and is stuck waiting for another to get the second chopstick.

3. Write a program (Version 3) so that no philosopher ever starves.

4. Write a program (Version 4) to provide a starvation-free solution for any number of philosophers *N*.

Keep all 4 versions of your program and save them in separate files or separate folders.

Use **multi-threading** in your solution. You can choose any programming language supporting threads for your implementation. Document your solution well. Keep all 4 versions of your solution into 4 separate files or folders. Provide a README.txt file with detailed instructions on how to compile and run your program from the command prompt.

Grading policy:

General program design and correctness: 60%

Efficiency: 20%

Documentation including statements and proof of correctness, efficiency, and experimental evaluation: 20%

Additional Instructions:

Cheating in any form will not be tolerated.

- **You should submit four program versions and all documentation in one archive file (zip, tarball, etc.).**
- Once a philosopher has chosen his seat, he can't move to a new position
    - As such he can only use the chopsticks to his left and right.
- Your design must implement the chopsticks as shared variables.
- The philosophers should only interact with each other through the chopsticks.
- Output Format:
    - When a philosopher goes from eating to thinking he should output:
        - "%d is now thinking.\n"
    - When a philosopher goes from thinking to hungry he should output:
        - "%d is now hungry.\n"
    - When a philosopher goes from hungry to eating he should output:
        - "%d is now eating.\n"
    - Two adjacent philosophers should never eat at the same time.
- The programs should run continuously until the letter 'n' is pressed.
- The last executable (starvation-free) should accept a command-line argument that represents the number of philosophers.