CS4375: Theory of Operating Systems **Instructor: Dr. Deepak Tosh**

Spring 2025 **Due Date:** 03/31/2025, Midnight

Assignment-4 (Concurrency)

The goal of this assignment is to allow you exploring the concurrency abstractions like threads, locks, and condition variables, which are vastly used in real life software applications. Here we will build a Master-Worker Thread pool that would allow you to understand how real-world systems like webservers and database servers work.

Building a Master-Worker Thread Pool

In this part, you will implement a simple master and worker thread pool, a pattern that occurs in many real-life applications. In this simple program, the master threads produce numbers continuously and place them in a buffer, and worker threads will consume them, i.e., print these numbers to the screen. This kind of structure is commonly used in multi-threaded architecture for web servers where the server has one or more master threads and a pool of worker threads. When a new connection arrives from a web client, the master accepts the request and hands it over to one of the workers. The worker then reads the web request from the network socket and writes a response back to the client. Your simple master-worker program is similar in structure to such applications, albeit with much simpler request processing logic at the application layer.

Given elements

You are given with a skeleton **master-worker.c** program. This program takes **4 command line arguments**:

1. M - How many numbers to "produce"
2. N - Maximum size of the buffer in which the produced numbers should be stored (N)
3. C - The number of worker threads to consume these numbers (use up to 50)
4. P - The number of master threads to produce numbers. (use up to 20)

The skeleton code spawns **P** master threads, that produce the specified number of integers from 0 to M − 1 into a shared buffer array. The main program waits for these threads to join, and then terminates.

Note: The skeleton code given to you does not have any logic for synchronization, which is to be done by you.

Tasks

You must modify this skeleton code **master-worker.c** in the following ways.

1. You must add code to spawn the required number of worker threads, and write the function to be run by these threads.
2. This function will remove/consume items from the shared buffer and print them to screen.
3. You must add logic to correctly synchronize the producer and consumer threads in such a way that every number is produced and consumed **exactly once**.
4. Producers **must not try to produce when the buffer is full**, and consumers **should not consume from an empty buffer**.

5. The main thread must call **pthread_join** on the master and worker threads and terminate itself once all threads have joined.
6. Your solution must only use **pthreads condition variables** for **waiting** and **signaling**: busy waiting is not allowed.

Assumptions:

a. While you need to ensure that all C workers are involved in consuming the integers, it is not necessary to ensure perfect load balancing between the workers.
b. Once all M integers (from 0 to M − 1) have been produced and consumed, all threads must exit.

If your code is written correctly, every integer from 0 to M − 1 will be produced exactly once by the master producer thread, and consumed exactly once by the worker consumer threads.

Hint: Try with small cases, for example, 1 producer and 2 consumer threads with small buffer size. Try to revisit the producer-consumer code that we learned in class.

To submit:

a. Programs with appropriate documentation and instruction to execute
b. A report on demonstrating your program's execution and correctness. Also mention any references that you used while developing your program.
c. Discussion on challenging aspects of this assignment.

Grading:

The assignment will be graded on following items:

1. Completeness and correctness on your responses, explanations, and observations.
2. Inclusion of appropriate evidence (in form of screenshots)
3. Clarity of Report
4. References (including links where you found some sample code).

References:

1. How to use POSIX Threads: https://hpc-tutorials.llnl.gov/posix/
2. Little book of semaphores - https://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf