Project: Pthreads Synchronization

In this project, you will learn the basics of multi-threaded programming, and synchronizing multiplethreads using locks and condition variables. You will use the `pthreads` thread API in this assignment.

Task: Your Own Synchronization Tool

In this part, you will implement your own synchronization pattern, much like the producer-consumer as discussed in class. You can come up with your own toy/real-life problem as well.

Your problem should have at least two different agents/threads, e.g., producers and consumers, each doing different things, and requiring some kind of synchronization between them. The synchronization requirement should be more complex than simple patterns like “thread 2 must run after thread 1” which are given as test cases in part C. Your program should spawn multiple threads to create these different agents, with some randomness in their start times, in order to achieve different interleavings of threads during testing. Each agent/thread should do some dummy work in its start function and print some message when it does the work, e.g., producer prints something when it produces and consumer prints something when it consumes. Without proper synchronization, the threads may function incorrectly, e.g., consumer may consume from an empty buffer. But with correct synchronization added, your print statements should indicate that the threads are synchronized correctly as expected. You must define the expected correct behavior of the various threads and demonstrate in your solution that the correct behavior is indeed achieved by looking at the output.

You will provide two different solutions to your synchronization problem, one using condition variables and the other using semaphores. For condition variables, you will use the CV and mutex abstractions from the `pthreads` API. For semaphores, you will use your own semaphore (zemaphore) abstraction implemented by you in part C above. You should develop, test, and demonstrate the CV and sempahore solution in separate C/C++ files. You will use condition variables/mutexes/semaphores as shared global variables in your program, that are available for use by all threads. During your testing, you should run your programs multiple times, with different interleavings of the threads, to demonstrate that your solution is correct.

There is no starter/template code provided for this part of the assignment.

Submission instructions

For part D, you will submit all code written by you.

- Place these files and any other files you wish to submit in your submission directory, with the directory name being your roll number (say, 12345678).
- Tar and gzip the directory using the command `tar -zcvf 12345678.tar.gz 12345678` to produce a single compressed file of your submission directory. Submit this tar gzipped file on Moodle.