Lab 6. The Sleeping-Barber Problem

1. Problem Description

A barbershop consists of a waiting room with *n* chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Describe your multi-threaded solution to coordinate the barber and the customers. There should be one thread representing the barber and one additional thread for each new customer. Implement your solution using only Pthreads functions.

2. Design/implementation Hints

No sample code will be given for this project. However, here is a possible design logic you can reference. You might use a counter for time allocation of a haircut. For example, if the counter reaches a pre-defined value, the haircut is considered done. You can also generate a random number to simulate whether a new customer arrives or not. For example, if the random number is even, it means a new customer has arrived.

You are **required** to use Pthreads mutex and condition variables to implement this lab. Please first read relevant sections in https://computing.llnl.gov/tutorials/pthreads/. (Note that you CANNOT use *semaphore.h* functions, such as *sem_init*, *sem_wait*, and *sem_post*. If you really want to use semaphore.h functions, you need to justify why and get the instructor's approval beforehand.)

3. Submissions

- (70%) Well-commented source code
- (30%) A report that
  - has a diagram explaining inter-process communication.
  - explains how to compile and run your code.
  - includes screenshots of your running program.

4. Grading:

- **50 points**: Multi-threaded implementation: one thread representing the barber and one additional thread for each new customer.
- **20 points:** The code is well commented, compiled successfully, and ran as expected. You should also fully test your code and document the output.
- **30 points:** The report clearly describes inter-process communication (how different threads communicate with each other) and test plan (what you have done to verify the correctness of your code).