Lab 6: Dining Philosophers

Objectives

- Study and solve the dining philosophers problem.
- Gain experience with a multi-threaded program using mutex locks.

You may complete this lab individually or with one colleague from this class using pair programming rules (see the course Canvas page).

1 Getting Started

Read about the dining philosophers problem in Ch. 31.6.

2 Dining Philosophers

The provided code diningphilosophers.c gives an implementation of a variation of dining philosophers problem. In it, there are five philosophers sitting around a table and five forks. Each fork is shared by two philosophers. Each philosopher thinks for a random amount of time and then chooses to eat. In order to eat, the philosopher has to pick up two forks. First, she picks up the fork to her right (if available), and then picks up the fork to her left (if available). However, these philosophers are very impatient and do not wait for their forks to be available. Your job is to make them wait using mutex locks (the book discusses solution with semaphores. In this lab you will only be using locks).

1. Compile and execute the given code as is. Note, since this code uses pthreads, you should compile the code with `gcc -pthread philosophers.c`. The exact execution of the code will be random, but you should notice that most of the philosophers eventually starve (meaning that they could not eat in 20 seconds).

2. You need to alter the given code so that a philosopher properly waits for a fork to become available when picking it up. To do this, create a mutex lock for each fork (of type `pthread_mutex_t`). Now in the function `pickup_forks`, the philosopher should lock a fork before picking it up (using `pthread_mutex_lock`). In the function `return_forks`, the philosopher should unlock both forks (using `pthread_mutex_unlock`). Now compile and execute this code. There should be a lot less printed out, since the philosophers are now waiting for their forks. However, you may encounter a deadlock in which each philosopher has one fork but is waiting for her second fork. When this happens, the program stops printing anything. You may have to run it a few times to encounter this behavior.

Lab 6: Dining Philosophers

3. There's a solution to this problem of deadlock. Instead of having each philosopher grab the fork on the right and then the left, some philosophers should instead grab the fork on the left first. Alter the code so that philosophers of even index grab the left fork first, while the other still grab the right fork. Compile and execute this code. If done properly, the program will continuously print updates on the philosophers, with all five philosophers eating periodically.

Submission

Submit `diningphilosophers.c` to Canvas.