EECS 3221

Assignment 3 **(Version 1.1)**

Implement a Solution to *Dining Philosophers Problem* Using Thread Management and Semaphores

The purpose of this assignment is to explore the concept of **Semaphores in Linux** and their applications.

What to Submit

Submit your work to eClass as a single **.C** file. Make sure to include your name and student number **on top of the source code file**.

Similar to assignment 1, make sure to comment your code and follow proper coding style and indentation.

Your source file should include a comment block describing the purpose of your program and how it works. You should also add a line of comment for every method definition, and system call.

Ensure your program does not produce warnings or errors when executing. Finally, always verify that you are submitting the most complete and tested version of your code to eClass.

Task

Implement a solution to the **Dining Philosophers** problem using semaphores, while avoiding possible deadlocks. You should use **POSIX C** thread management and Semaphores to implement your solution.

Preparation

Review the lecture notes and the manual page for **POSIX thread management** and **POSIX semaphores**.

Specifically, you should check thread management methods such as pthread_create, and semaphore management methods including sem_wait, sem_post and sem_init.

Requirements

1- Write a C program that generates a synchronization solution for exactly **5 philosophers**.

2- For each philosopher **create a separate thread**. Ensure each thread is created correctly by checking for return value of the system calls.

3- The main thread should **wait** for the philosopher threads to complete their execution. Once all threads have exited, the main thread should **destroy any semaphores** and exit.

4- For every philosopher, the philosopher **begins in thinking mode.** Then the philosopher will try to use the semaphore to check to see if their **left** chopstick and **the** right chopsticks are available. If available, the philosopher should begin **eating** for **exactly 5 seconds**. After that, the philosopher should **put down the left and then the right chopsticks**. Next the philosopher will **relax for exactly 5 seconds**, after which the philosopher thread should exit.

5- During the phases of thinking, eating and relaxing, the thread should **print out a message** saying 'Philosopher [id] is in [state] mode', where [id] is the id of the philosopher, and the [state] indicate the philosopher's state.

6- When a philosopher picks up a chopstick, the thread should **print out a message** saying 'Philosopher [id] picked up [left/right] chopstick [cid]', where [id] is the philosophers id, [left/right] should indicate either left or right chopstick, and [cid] is the id of the chopstick.

7- Ensure only **4 philosophers** can eat at a time to avoid deadlocks. Note, the order by which philosophers eats is not important, as long as all **5** philosophers eventually eat.

Your program takes no command line arguments. All values can be added directly to the source code. Your program should work in a Linux like environment such as the lab environment available at Lassonde building.