CS 5600 Project 2

1- Overview

This is the second team project which involves using the semaphores and/or mutex locks to avoid deadlock in a multi-threaded program.

A department store consists of two floors that are connected by narrow stairs that cannot fit more than one person on any given stair. The stairs can become deadlocked if both a first-floor customer is trying to go up to the second floor, while a second-floor customer is trying to go down to the first floor. Customers in such situation refuse to backup leading to deadlock.

The store manager is hiring you to design a C program using POSIX synchronization, that

1. prevents deadlock
2. prevents starvation (the situation in which first-floor customers prevent second-floor customers from using the stairs, or vice versa)
3. and allows more than one customer to use the stairs in the same direction in an "efficient" way that you determine.

- Represent first-floor customers and second-floor customers as separate threads. Once a customer is on the stairs, the associated thread will sleep for a random period of time, representing the time to go over all the 13 stairs between the two floors.
- Design your program so that you can create several threads representing first-floor customers and second-floor customers, and test it using different possible scenarios.
- Find the Turnaroud time and Response time for each customer

2- Documenting your project

In a **readme.md** file, include the following:
1. Your group number, and the name of the students in your group
2. A short description of the project.
3. Explain how you implemented the project:
   - List the functions you wrote, and the purpose of each function
   - Explain how you tested your project and list the test cases you used
   - Explain how you are guaranteeing that your code is free of deadlock and starvation.
   - Find the average Turnaroud time and Response time of the examples you run, and explain using these performance measures how you adjusted your project to make your design "efficient".
   - Explain how we can compile, run and test your code.
   - List the contributions of each student in your group

3- Submitting your work

You should place all relevant files to the project in a folder and create a zip file by zipping the folder itself. You should not include object files or executable binary files that are generated during the compilation process. The zip file should be submitted via Canvas. Only one of the team members may submit the work: if one member uploaded the file on Canvas, the other team member will automatically see the same submission. Call your files Project2_group#.