OS Assignment 3

Lead TAs: Hamza, Mati and Zafir

Good News

- This assignment can be done in pairs now.
- However, it is not compulsory to do it in pairs. You may do it individually if you want to
- Google sheet: https://docs.google.com/spreadsheets/d/1b42FBULvZcgRfbcif5MBelX2J_r-XRO7jFd_iKSFlfs/edit?usp=sharing

Background - The Critical Section

The critical section is a section of the code in which a process can access shared resources.

Only one process or thread can access the critical section at a time.

Three Conditions:

Mutual Exclusion: Only one process can be in the critical section at one time

Progress: When no process in critical section, one of the waiting process must be given access of the critical section without delay

Bounded waiting: Bound on the number of times other processes can enter critical section while I wait

Pthreads

- We will be using Pthreads in multiple places throughout this assignment
- Single code and data segment for all the threads within a process – we need this power of threads and will use it to our advantage for this assignment.
- We will be using the following two functions:
  - **int pthread_create(pthread_t *restrict** *thread*, **const pthread_attr_t *restrict** *attr*, **void *(****start_routine***)(void *), void *restrict** *arg***);**
  - **int pthread_join(pthread_t** *thread*, **void *****retval***);**

Semaphores

We will be using semaphores as our only synchronization construct for this assignment. No other constructs allowed.

We will use the following three functions:

int sem_init(sem_t *_sem_, int _pshared_, unsigned int _value_);	**int sem_post(sem_t *_sem_);**	**int sem_wait(sem_t *_sem_);**

Some useful resources

- **Pthread Creation:** https://man7.org/linux/man-pages/man3/pthread_create.3.html
- **Pthread Joining:** https://man7.org/linux/man-pages/man3/pthread_join.3.html
- **Semaphore Initialization:** https://man7.org/linux/man-pages/man3/sem_init.3.html
- **Semaphore waiting:** https://man7.org/linux/man-pages/man3/sem_wait.3.html
- **Semaphore release:** https://man7.org/linux/man-pages/man3/sem_post.3.html

Part 1 (Problem Statement)

- You have to design an elevator controller for a university
- The elevator will move from the ground floor to the top floor and back to the ground floor again
- You may assume that the maximum number of floors within the university will be 20 (Your program should work for all values <= 20)
- Your elevator will keep on running till there are no more people waiting for the elevator and no one is remaining inside the elevator
- Your elevator must move in order e.g (2,3) will be dropped before (1,4)
- Your elevator can hold upto a maximum of n number of people, which has been provided to you.
- Three key things to consider: Max_floor_no, elevator_capacity and total number of people

Part 1
(Implementation)

- The following function declarations have been provided to you in the skeleton code. You need to implement them:
  - **InitializeP1(numFloors, maxNumPeople):** numFloors is the total number of floors within the university. maxNumPeople is the max number of people that can be in the elevator at a given point in time. In this function, you will initialize all the global variables and semaphores that you declare. You are allowed to declare as many of them as required.
  - **goingFromToP1(*args):** Function called by every person as they they call the lift. (Already being called in test cases) Args are the details of the person such as person id, their source floor and their destination floor
  - In addition to this, you need to make a main elevator function as well that will be called as a thread within the startP1 function in part1.c. This main elevator function will keep on running till no one is waiting for it and all the current people have been dropped

Part 2 (problem statement)

- This part is like part one except that instead of having one elevator, you will now have 5 trains running concurrently.
- You will have 5 stations and 5 trains. Initially train 0 will be at station 0, train 1 at station 1 and so on. Assume that all passengers will be waiting at the stations when trains start moving.
- Each passenger object will have a to and from station number.
- Only one train can be at a station at a time.
- Trains will stop at a station, pick up passengers and then drops them at their respective destinations.
- For each passenger when it gets off at its destination, you will have to print its (train number, station from, station to) for example 3 1 2
- The order of these print statements is very important. For example, a person going from station 2 to 3 should be printed first then a person going from station 1 to 5.
- The trains will keep running in a loop until no more passengers are left.
- For more information read the assignment manual.

Part 2 (implementation)

- You are given starting code in part2.h and part2.c
- Initialize 5 threads one for each train in the startP2() function. Make sure each train starts at the correct station.
- The threads should run until all passengers are dropped at their destination.
- Each passenger is represented by a thread running the goingFromToP2(*void * user_data*) function.
- Use semaphores to synchronize the running of the trains so that the passengers are picked up and dropped off in the correct order.
- Initialize semaphores and variables in initializeP2(*int numStations*, *int maxNumPeople*) function

Part 3

- Traffic lights: rule for crossing the intersection
- Similar to previous parts, but have added few rules
- 1. There are four traffic lights, one for each direction (North, South, East, West). Only one traffic light becomes green at any given time; the remaining traffic light will be red.
- 2. A car going straight or turning left must be in the left lane.
- 3. A car turning right must be in the right lane.
- 4. A car going straight or turning right cannot cross the intersection if the corresponding traffic light is red and must wait for the traffic light to go green. The traffic light for going straight or turning right becomes green at the same time.
- 5. A car may turn left on Red light if there is no car waiting in front of it for green traffic light.
- 6. A traffic light remains green until 5 waiting cars from each lane cross the intersection or turn right. In case there are less than 5 waiting cars in each lane, the traffic light becomes red as soon as all the waiting cars cross the intersection or turn right.
- 7. Each car is represented by a thread. The car thread shouldn't exit until the car has crossed the intersection or made the turn.

Part 3

- Car->from ...(north,south,east,west)
- Car->to ...(north,south,east,west)
- Car->lane ...(left,right)
- Car->user_id ...(i)
- Maintain the number of cars in each direction+lane
- Use semaphores to ensure the cars follow the rules so there is no accident
- Output format: from to lane
- If there is a car going from SOUTH to NORTH, from lane LEFT, print SOUTH NORTH LEFT
- Hint: understand the **test3** function in main.c to find out how to use directions and lanes.

GOOD LUCK!