CSC112 Lab3: Hofstra Bus System

Objectives

Get familiar with pthread library usage in Linux. Use synchronization to solve real-world problems. (This assignment only works on Linux. Please refer to "Running Linux on Windows or MacOS" on course homepage.)

Hofstra Bus System

Write a concurrent C program that emulates the behavior of the Hofstra bus system. Each student and each bus is modeled by a thread. When a bus arrives at the station, it should signal and wake up all the student threads waiting at the station to board the bus, and then goes to sleep waiting in a loop. When each student boards the bus in ticket order, the student thread should signal and wake up the bus thread so it can continue to wait for the next student to board. The boarding process ends if no more students are waiting or no seats are available on the bus. Use condition variables for the waiting and signaling operations. (This is one possible system design. Feel free to design your system in a different way as long as it works correctly.)

You are asked to write synchronization functions that will guarantee orderly boarding. In `hofbus_incomplete.c`, `struct station`, has been defined along with a function `station_init(struct station *station)`. You need to implement two functions in the file `hofbus_incomplete.c`, and rename the file as `hofbus.c` before submitting it.

When a bus arrives at the station and has opened its doors, it invokes the function:

```
station_load_bus(struct station *station, int count)
```

where `count` indicates how many seats are available on the bus. The function must not return until the bus is boarded successfully (either the bus is full or all waiting students have boarded).

When a student arrives at the bus station, the student will receive a ticket. The student with a smaller ticket number would get aborded earlier. After that, the student would start waiting by invoking the function:

```
station_wait_for_bus(struct station *station, int myticket, int myid)
```

The ticket number and student id would be passed to the function. This function must not return until a bus is in the station (i.e., a call to `station_load_bus` is in progress) and it's the student's turn and there are enough free seats on the bus for this student to boardr. The function should return the student's boarding turn. For example, if you are the 3rd student who gets boarded a bus at that station, the function should return 3.

You must write your solution in C using the Pthread functions for locks and condition variables:

`pthread_mutex_init, pthread_cond_init, pthread_mutex_lock, pthread_mutex_unlock, pthread_cond_wait, pthread_cond_signal, pthread_cond_broadcast`

Use only these functions (e.g., no semaphores or other synchronization primitives).

- You may not use more than a single lock in each struct station.
- You may assume that there is never more than one bus in the station at once and that all buses (and all students) are going to the same destination.
- Your code must allow multiple students to board simultaneously.
- Your code must not result in busy waiting.

Tester

The testing program `hofbus-tester.c` is provided to execute your concurrent functions with some test cases. It would be helpful to read through the comments to understand those test cases.

The taskset command in Linux is a utility used to set or retrieve the CPU affinity of a running process or to launch a new process with a specified CPU affinity. CPU affinity refers to binding a process to specific CPU cores, ensuring it only executes on those cores. The command “taskset -c 0 ./hofbus-tester” runs ./hofbus-tester on CPU core 0, including all of its threads. Although the program should work with or without taskset theoretically, it may simplify your debugging task by binding to a fixed core.

The following part shows how to compile and run the tester in a command line environment with the Pthread library. The output should not contain any error messages.

```
# gcc -o hofbus-tester hofbus-tester.c -pthread
# taskset -c 0 ./hofbus-tester
student 7 got ticket 1 and start waiting
student 8 got ticket 2 and start waiting
student 9 got ticket 3 and start waiting
student 10 got ticket 4 and start waiting
student 11 got ticket 5 and start waiting
student 12 got ticket 6 and start waiting
student 13 got ticket 7 and start waiting
student 14 got ticket 8 and start waiting
student 15 got ticket 9 and start waiting
student 16 got ticket 10 and start waiting
student 17 got ticket 11 and start waiting
student 18 got ticket 12 and start waiting
student 19 got ticket 13 and start waiting
student 20 got ticket 14 and start waiting
student 21 got ticket 15 and start waiting
student 22 got ticket 16 and start waiting
student 23 got ticket 17 and start waiting
student 24 got ticket 18 and start waiting
student 25 got ticket 19 and start waiting
student 26 got ticket 20 and start waiting
student 27 got ticket 21 and start waiting
student 28 got ticket 22 and start waiting
student 6 got ticket 23 and start waiting
student 29 got ticket 24 and start waiting
student 30 got ticket 25 and start waiting
student 5 got ticket 26 and start waiting
student 4 got ticket 27 and start waiting
student 3 got ticket 28 and start waiting
student 2 got ticket 29 and start waiting
student 1 got ticket 30 and start waiting
Bus entering station with 4 free seats
student 7 with ticket 1 has boarded, the turn is 1
student 8 with ticket 2 has boarded, the turn is 2
student 9 with ticket 3 has boarded, the turn is 3
student 10 with ticket 4 has boarded, the turn is 4
Bus entering station with 2 free seats
student 11 with ticket 5 has boarded, the turn is 5
student 12 with ticket 6 has boarded, the turn is 6
Bus entering station with 0 free seats
Bus entering station with 8 free seats
student 13 with ticket 7 has boarded, the turn is 7
student 14 with ticket 8 has boarded, the turn is 8
student 15 with ticket 9 has boarded, the turn is 9
student 16 with ticket 10 has boarded, the turn is 10
student 17 with ticket 11 has boarded, the turn is 11
student 18 with ticket 12 has boarded, the turn is 12
student 19 with ticket 13 has boarded, the turn is 13
student 20 with ticket 14 has boarded, the turn is 14
```

```
Bus entering station with 3 free seats
student 21 with ticket 15 has boarded, the turn is 15
student 22 with ticket 16 has boarded, the turn is 16
student 23 with ticket 17 has boarded, the turn is 17
Bus entering station with 8 free seats
student 24 with ticket 18 has boarded, the turn is 18
student 25 with ticket 19 has boarded, the turn is 19
student 26 with ticket 20 has boarded, the turn is 20
student 27 with ticket 21 has boarded, the turn is 21
student 28 with ticket 22 has boarded, the turn is 22
student 6 with ticket 23 has boarded, the turn is 23
student 29 with ticket 24 has boarded, the turn is 24
student 30 with ticket 25 has boarded, the turn is 25
Bus entering station with 0 free seats
Bus entering station with 0 free seats
Bus entering station with 3 free seats
student 5 with ticket 26 has boarded, the turn is 26
student 4 with ticket 27 has boarded, the turn is 27
student 3 with ticket 28 has boarded, the turn is 28
Bus entering station with 4 free seats
student 2 with ticket 29 has boarded, the turn is 29
student 1 with ticket 30 has boarded, the turn is 30
Looks good!
```

Hint:

1. The `station` structure could be used to track a) the number of free seats; b) the number of waiting students; c) the turn of the next student to board. The number of free seats in a station should be zero after the bus leaves.
2. Locks or conditional variables can be defined inside the `station` structure to
   a) make sure one student gets boarding at a time; b) let a student wait if there are no free seats; c) wake up the waiting students when a bus with free seats arrives; d) let the arriving bus wait if there are more waiting students to fill the available seats.

What to submit:

Submit only the `hofbus.c` file and a screenshot of the output. The `hofbus-tester.c` is not expected to be modified. If you do so for some reason, please upload the tester as well and explain the changes you made and why.