CS 307 PA 3 REPORT
Name and Surname: Görkem Yar ID: 27970

IMPORTANT NOTE!!!!: I used sabanci.flow to run my programming assignment with the command “g++ -lpthread -o rideshare rideshare.cpp -std=c++11”. Please, use this command and flow while grading my homework.

The synchronization mechanisms that I used:

pthread_barrier for creating a phase to my program

pthread_lock for operations that should not be interleaved

semaphore as conditional variable (team semaphores) and as barrier (all semaphore)

My program:

First thing that about the homework that I discovered is if there are 5 threads that have been looking for a car then there is a valid combination. For instance, 4 A 1B or 3A 2B is valid or vice versa. I thought there should be a general semaphore that should prevent threads for searching for a car if there are 4 threads already in a car. To this end, I created “all” semaphore with initial value 4.

Also, I created two more semaphores for the teams since the threads should wait in one of these semaphores while waiting for a valid combination. Name of these semaphores are “home” and “rival” with initial value 0. Additionally, I created two integer variables to keep track of the number of threads that are waiting in these semaphores. Which are named as “fanCount” and “rivalCount”.

Also, I have created a barrier for second phase of the program and initialized it with 4.

Apart from these there is also a lock.

Note: If there is not a valid combination in first 4 threads my program will send signal to all semaphore, and it will wake up one more thread. Since this thread is the fifth thread there should be a valid combination.

My main pseudecode:

```
int main(argv){
    take fan and rival counts from argv
    if counts are valid
        create threads
        wait for threads to join
    print("The main terminates")
}
```

My pseudo code for audience function:

```
audienceThread(){
    Sem_wait(all) // prevent more then 4 threads to start phase 1
    //phase 1 started
    Acquire_lock
    Print(thread_id" looking for a car")
    Increment own team by 1.

    If (total waiting thread number is 4 and there is no valid combination){
        sem_signal(all) // since we could not find a valid combination in  4 threads
        release lock
        sem_wait(team_semaphore)
        // wait for a valid combination in this threads semaphore
    }
    else if(there is a valid combination){
        make this thread captain since it is the last one
        wake up the waiting threads in team semaphores
            (be careful if number of threads in rival semaphore is odd
            do not wake up all)
        release lock
    }
    else {
        // this is the case when there is no valid combination
        // and number of threads is less than 4
        release lock
        sem_wait(team_semaphore)
    }

    // phase 2 started:
    // now 4 threads are came here
    print(thread_id, "I have a spot in the car")

    barrier_wait // wait for all of the threads to hit barrier

    // phase 3 started;

    if (the thread is captain){
        print(thread_id, "I am the captain")
        send 4 signals to all semaphore to wake up threads that are waiting
        for phase 1
        // be careful if there is a sleeping thread in team semaphores
        // send 3 signals
    }
    // end of the execution
}
```

The logic behind my program as follows:

Phase 1:
There can be 4 or 5 threads that entered this region. In this region they will print we are looking for a car. One of them will become captain and wake up the other threads. If there are 5 threads in this region captain will not wake up the fifth one.
Phase 2:
There can be exactly 4 threads in this region.
They will print they have a spot in the car.
Phase 1 cannot start during a Phase 2 because a semaphore called “all” prevents that.
Phase3:
There can be exactly 4 threads and only 1 captain. Captain thread will print I am the captain.
Release the “all” barrier to wake up threads and start a new Phase 1
Other threads will do nothing.