Oytun Kuday Duran 28357 Report for PA 3

Flow of the threads and usage of mechanisms:

The threads get created and joined at the main inside for loop after the necessary checks. The main thread terminates after joins. There is a barrier at the start of thread function (named share) that helps all threads start at the same time. There is a semaphore named semGlob, It's initial value is 5 so that when 5 threads join, the rest waits. It is 5 because there must be a match in 5 threads (2-2 or 4-0). It wouldn't work if it was 4 (1-3 condition). The threads joined the function gets checked in some if statements whether they are able to fill a car or not. It is checked by using semaphore values. The SemA and SemB sephamores are used to keep the count of the number of the ridesharers and keeping them waiting. Their initial value is 0 since all threads enter the else statement needs to wait to be awaken when a matching combination is found. If there is no matching condition, they go to else statement and wait until a matching combination is found. I used a function i created named sem_unlock because If a thread sleeps, we need to unlock the global lock named 'lock2' since otherwise it would be a deadlock, if we unlock before the wait, there may be a context switch occur. This lock2 is used to protect critical sections and ensure concurrency while using multiple sem_post()'s. When a matching combination is found, if a thread gets into a if statement, firstly, it makes local variable driver true so that the last thread that completes a car-combination would be the driver. Then, it resets the barrier I used to print "captain statement or end statement" after the I have found a spot parts. Every time a car is filled, the barrier gets the value 4 since the captain output needs to be after 4 thread "I have found spot" outputs. Then, it wakes up regarding threads. We know which threads should be awaken because of value checks. We don't have a chance to wake up random threads since we used a global semaphore that ensures the unnecessary threads won't be continuing. After 3 threads that were sleeping wakes up and prints "I have found a spot", they go down to a barrier initialized to 4. After 4 threads printed the "I have found" statement and reached the last part of function, the driver (a.k.a. the last thread) prints that it is the captain because of the local bool variable we used. We wake up 4 global semaphores at the same time with lock2 so that the process continues and other newcomer threads doesn't interrupt the process. It was 5 at the start but we used 4 threads. 1 thread remaining combines with 4 newly awaken threads and they form a combination too. The last thread which enters an if statement and becomes captain/ driver doesn't sleep but it wakes up necessary threads to avoid a deadlock. Other threads wait at the barrier after outputting the necessary "i have a found" part. After 4 thread joins the barrier, the driver prints that he is captain and wakes up other threads.

It is correct since I carefully designed and implemented the system and output is accomodating with the correctness in PA pdf. If a thread starts processing, the lock2 locks the other threads so I can control the threads easily. The semaphores and barriers are also correct since my initializations give the correct output. For example, after checking if a car-filling condition is available, it wakes up the regarding threads with sem_post. Since we wait at the beginning of the code with a global semaphore, we don't have a chance to wake up a random thread to cause problems. The deadlock was avoided by a function I created named sem_unlock which unlocks mutex and waits a semaphore. Some of the semaphore struct and functions were taken from the lecture slides and modified slightly. I believe that my explanations about the flow of the code is enough to understand my locking and concurrency mechanisms.