Ali Cenker Yakışır – PA3 Report

In this assignment, I have used 2 different thread method for 2 group (A and B). These methods are exactly same except A and B characters (in variables, printings etc.)

There are 2 semaphores for each group as well as 1 common semaphore which is used like mutex lock. Also, number of sleeping threads` count of both groups being held at global scope. Common semaphore initialized to 1 and other waitingA and waitingB semaphores are initialized to 0 (it will be explained in the next parts).

Firstly, a thread gets the common semaphore (named as critSec) and prints its init. Then it checks is there enough sleeping threads to fill the car. If it couldn't find, it goes to sleeping statement directly since group semaphores are initialized to 0. Before going to sleep, it also increases the common semaphore, in other words, releases the common lock.

If some other thread finds enough people to fill the car, this thread will be lead thread which also drives the car. It wakes the 3 sleeping thread and updates groups sleeping count. It also hasn't released common lock (semaphore) yet. The threads at before the ***init*** cannot be able to run because of the common lock(semaphore). Yet, the newly awaken treads won`t be affected. Since it won`t face any other wait() of the critSec. This way only 4 of the threads that shares the car will be able to run. They will print that they found a car. After printing, they face a pthread_barrier (this was initialized to 4 by driver before waking the others). All 4 of the threads must be reached to barrier in order to exceed barrier. It means that their ***mid*** printings must be done to pass the barrier. After the barrier, only the driver thread will print that it is the driver. Also, driver thread releases the critSec semaphore. So, other non-sleeping threads can do their initialization while getting the critSec semaphore.

I think initializing groups semaphore by 0 is the good solution since every non-driver thread will go to sleep directly. Also, there was a need of one common lock in order to prevent that some threads init printings will be intervene a full groups printing. Also, comparisons must be protected by common lock. Another thing is that barrier really works well in this assignment since program needs to wait all mid printings to complete before the printing of the driver and releasing the lock.