Mert Kılıçaslan

Riding to a Soccer Game (PA – 3) Report

CS307 – Operating Systems

Flow of the threads as a Pseudo Code:

```
global int  counterA = counterB = counterBarrier = 0
void* rideshare(void* args){

    char teamLetter = fromArgsTakeChar
    thread_barrier_wait(wait_all_threads)
    print("Looking for a car", teamLetter)

    while(true){

        if(counterA, counterB == (4-0) or (0-4) or (2-2)){

            if(teamLetter == 'A'){
                mutexLock
                counterA++
                mutexUnlock
            }

            else if(teamLetter == 'B'){
                mutexLock
                counterB++
                mutexUnlock
            }

            print("Found a spot in a car", teamLetter)
            thread_barrier_wait(wait_four_thread)

            mutexLock
            counterBarrier++

            if(counterBarrier == 4){
                print("I am the captain", teamLetter)
                counterA = counterB = counterBarrier = 0
            }
            mutexUnlock

            thread_barrier_wait(wait_all_threads)
            break_the_loop

        }
    }
}
```

Description of the Synchronization Mechanisms:

In the implementation of the code, I use 4 Mutexes, 2 Pthread Barriers, and logical if/while conditions. First of all, my program starts by getting the correct team numbers from terminal input, if they are not correct. (i.e. they are not even number or sum of them doesn't divisible by four), the main thread immediately terminates and finishes the execution. If all is well, the main thread creates two barriers that can hold respectively four threads and the total thread count. Then after it proceeds to create a specified number of threads for team A and team B, and send them to the rideShare function. After that, it starts to wait for its children to finish their work, in the pthread_join.

In the rideshare function, To make the synchronization more effective we have an optional pthread barrier that holds all threads to reset the elapsed time while they enter the function. After the last thread to complete the total count for barrier comes, all thread prints "looking for a car" output and enters an unconditional while loop. Inside the loop, they meet an if a condition that checks thread validity for forming 2-2, 4-0, 0-4 bands (i.e. While true, all threads try to enter the if the condition that checks the validity). If it is a valid thread, it enters the condition and increases the global team counter by one (since it is a critical section, mutexes are used for counter increments, otherwise there might be a race condition), that are initialized as zero in global, to update the if condition to check upcoming threads. Then they prints out the "found a spot in car" output. After a valid thread enters the condition validator and increases the counter by one, it waits at the barrier holding the four threads to create a valid band. When the fourth valid thread comes, the barrier will be crossed and as a result, there will be a band with four threads. One of these threads must be captain, therefore we have another counter that allows us to find the last or 'the one thread'. The last thread resets all the counters with mutexes, including the counter to detect the last thread, and then prints the "I am the captain" output. After this point, threads that are in the same band wait at the barrier that holds all the thread count (same as the first barrier) to make sure all thread have finished their outputs and execution in order. When the last thread arrives at the last barrier, the loop will be broken, and all of them return to the main function. They finished their execution and they wake up the main thread after the children finish their execution, the main thread finish its execution as well.