CS307-PA3

Uğur Günal 30558

Pseudo Code:

```
Barrier barrier  #initialized to 4 in main
Mutex lock1
Mutex lock2
Semaphore semA = 0
Semaphore semB = 0
int waitingA = 0
int waitingB = 0
int CarID = 0
int PassangerCount = 0

function thread_function(TeamName):
   print "Thread ID:", get_thread_ID(), ", Team:", TeamName, ", looking for a car"
   lock1.acquire
   if TeamName is "A":
      waitingA += 1
   else if TeamName is "B":
      waitingB += 1
   if waitingA >= 2 and waitingB >= 2:
      lock2.acquire
      release 2 semA
      release 2 semB
      waitingA -= 2
      waitingB -= 2
   else if waitingA >= 4:
      lock2.acquire
      release 4 semA
      waitingA -= 4
   else if waitingB >= 4:
```

```
      lock2.acquire
      release 4 semB
      waitingB -= 4
   lock1.release
   if TeamName is "A":
      semA.acquire
   else if TeamName is "B":
      semB.acquire
   barrier.wait
   lock1.acquire
   PassangerCount += 1
   print "Thread ID:", get_thread_ID(), ", Team:", TeamName, ", found a spot in a car"
   if PassangerCount % 4 == 0:
      print "Thread ID:", get_thread_ID(), ", Team:", TeamName, ", I am the captain and driving the car with ID", CarID
      CarID += 1
      lock2.release
   lock1.release
   return
```

Overview:

My C++ program provides a rideshare service for fans of two different football clubs. A single car will be used for transportation, accommodating exactly four people for ridesharing. It is not allowed to share the ride with one person from club A and three people from club B. Only 4-0 and 2-2 ridesharing combinations are allowed.

Description:

In my program, I utilize two semaphores (semA and semB) , two mutexes (lock1 and lock2) and one barrier to implement the desired ridesharing mechanism and outputs. The program begins by checking the validity of terminal input, ensuring that the number of fans from each team is even and that the total fan

number is divisible by 4. If these conditions are satisfied, the program continues; otherwise, it terminates after printing 'main terminates.' Then program creates Threads vector and initialize barrier to 4.

The program creates a thread for each fan and pushes these threads into the Threads vector. In the program, a single thread function is used, and the team that the fans support is passed as a parameter to the thread.

Thread Implementation:

The thread starts by printing the thread ID and “looking for a car.” It then acquires lock1, ensuring that waitingA and waitingB are incremented by only one thread at a time. Additionally, if there is a suitable rideshare combination, only one thread will wake up other threads. After acquiring the lock, the thread increments either waitingA or waitingB based on the team it belongs to.

The thread then checks if there is an available rideshare combination by inspecting waitingA and waitingB. Three variations are considered for a correct rideshare: 2-2, 4-0, 0-4. If any of these combinations holds true, the thread acquires lock2. This lock ensures that the next rideshare combinations do not interweave with the current one.

The thread wakes up the waiting threads, decrementing waitingA and waitingB by the number of threads it wakes up in each team. Finally, the thread releases lock1 and waits on semA or semB, depending on its team.

If a waiting thread is awakened by another thread through the process explained above, it first waits for other 3 threads for passing the barrier. After passing the barrier thread acquire lock1 to prevent multiple threads from concurrently becoming a captain. Then thread increments PassengerCount by 1 Subsequently, it prints the thread ID and the message “found a spot in the car.” Since there should be one captain for every four threads (PassengerCount % 4 == 0), if this condition holds true, the thread becomes the captain. It then prints 'I am the captain,' followed by the carID. The carID is then incremented for the

next combination, and lock2 is released for the next rideshare combinations. Finally lock1 is released.

After all threads are created, main waits for all threads to terminate by iterating through the Threads vector, and finally, it prints “Main terminates.”