PseudoCode

```
// Global variables
int numTeamA;         // Number of fans in Team A
int numTeamB;         // Number of fans in Team B
sem_t teamASem;       // Semaphore for Team A fans
sem_t teamBSem;       // Semaphore for Team B fans
pthread_barrier_t barrier;  // Barrier to synchronize fans forming groups
pthread_mutex_t mutex;      // Mutex for protecting shared resources
int carID = 0;            // Counter for car IDs
int fanA = 0;             // Counter for Team A fans looking for a car
int fanB = 0;             // Counter for Team B fans looking for a car
int carCountA = 0;         // Counter for Team A fans in a car
int carCountB = 0;         // Counter for Team B fans in a car
```

//

```
// Function of Team A
function teamAfan(threadID):
acquire mutex
print "Thread ID: " + threadID + ", Team: A, I am looking for a car"
increment fanA
if fanA >= 2 and fanB >= 2:
release mutex

for i in range(2):
release teamBSem
```

```
for i in range(2):
release teamASem

decrement fanA by 2
decrement fanB by 2

else if fanA == 4:
for i in range(4):
release teamASem

decrement fanA by 4
release mutex

else:
release mutex
wait for teamASem

barrier.wait
acquire mutex
print "Thread ID: " + threadID + ", Team: A, I have found a spot in a car"
increment carCountA

if carCountA + carCountB == 4:
print "Thread ID: " + threadID + ", Team: A, I am the captain and driving the car with ID " + carID
increment carID
```

reset carCountA and carCountB to 0

release teamASem

reset fanA and fanB

release mutex

//

// Function of Team B

function teamBfan(threadID):

acquire mutex

print "Thread ID: " + threadID + ", Team: B, I am looking for a car"

increment fanB

if fanA >= 2 and fanB >= 2:

release mutex

for i in range(2):

release teamBSem

for i in range(2):

release teamASem

decrement fanA by 2

decrement fanB by 2

```
else if fanB == 4:
for i in range(4):
release teamBSem

decrement fanB by 4
release mutex

else:
release mutex
wait for teamBSem

barrier.wait
acquire mutex
print "Thread ID: " + threadID + ", Team: B, I have found a spot in a car"
increment carCountB

if carCountA + carCountB == 4:
print "Thread ID: " + threadID + ", Team: B, I am the captain and driving the car with ID " + carID
increment carID
reset carCountA and carCountB to 0
release teamBSem

reset fanA and fanB
release mutex
```

//

```
// Main function
function main(argc, argv):

    numTeamA = parseInteger(argv[1])
    numTeamB = parseInteger(argv[2])

    // Check validity
    if numTeamA % 2 != 0 or numTeamB % 2 != 0 or (numTeamA + numTeamB) % 4 != 0:
        print "Main terminates..."
        return 1

    initializeSemaphore(teamASem, 0)
    initializeSemaphore(teamBSem, 0)
    initializeMutex(mutex)
    initializeMutex(mutex2)
    initializeBarrier(barrier, 4)

    threads = createArray(numTeamA + numTeamB)
    threadIDs = createArray(numTeamA + numTeamB)

    for i in range(numTeamA + numTeamB):
        threadIDs[i] = i
        if i < numTeamA:
            createThread(threads[i], teamAfan, threadIDs[i])
        else:
            createThread(threads[i], teamBfan, threadIDs[i])
```

```
  for i in range(numTeamA):
    release teamASem
  for i in range(numTeamB):
    release teamBSem

  for i in range(numTeamA + numTeamB):
    joinThread(threads[i])

  destroySemaphore(teamASem)
  destroySemaphore(teamBSem)
  destroySemaphore(captainSem)
  destroyMutex(mutex)
  destroyMutex(mutex2)

  print "The main terminates."
  return 0
```

//

Explanation

Program has 2 semaphores for two teams, one mutex and one barrier. Firstly, I check whether the given fan numbers are correct by checking if they are even and their total is divisible by 4. After that I initiliaze barrier and mutex and semaphores. Then I create two vectors to keep track of the threads and thread IDs. Then I create threads of team A and team B and join them accordingly. Lastly I destory barrier, semaphores and mutex.

//

Thread Functions

My two distinc functions for 2 teams works the same logically. Firstly, I get the ID of the thread. After that, lock the mutex and print “I am looking for a car”. Then thread checks the number of fans, if there are more than or equal to 2 fanA and fanB, then wakes up two thread from each team and subtracts 2 from numA and numB fan numbers each. If we are in the Fan A function, it checks if thread itself is the 4th thread and if so, wakes up 4 team A threads and reduces 4 from Fan B number. Similarly if we are in Fan B function and thread itself is the 4th thread, it wakes up 4 team B threads and reduces 4 from fan B number. This way we can make sure only 2 A and 2 B, 4 A or 4 B combinations will occur. If none of the conditions are valid, it goes in the else condition and waits a thread. This system allows us to block other threads enter the race for the same car and wakes up only the necessary threads.

After total of 4 threads wakes up and starts to race, within locks they update the carCounts by incrementing one and couts “I found a car”. CarCount allows us to choose a captain. I chose the last coming thread to become the captain for the ease. After checking if carCount + carCount == 4, if that is the case it prints “ I am the captain” , resets the counters, increment the carID – so that we can print of which car the current thread is becoming the captain - , wakes up a thread and unlocks the mutex. Then returns NULL from the function.

After all threads get created and find a spot, shell program destroys the mutex, barrier and sems and prints “Main terminates…”