**AYÇA ELİF AKTAŞ – 27802**

```
semaphore mutex = 1; // Binary semaphore
semaphore fanA_queue = 0; // Counting semaphore
semaphore fanB_queue = 0; // Counting semaphore
semaphore captainSem = 1; // Binary semaphore
barrier barrier = 4; // Initialize the barrier with a count of 4
barrier barrier2 = 4; // Initialize the barrier with a count of 4
barrier barrier3 = 4; // Initialize the barrier with a count of 4
int numA, numB;
int carCounter = 0;
int fanA = 0;
int fanB = 0;
bool isCaptain = false;

// Helper functions
void spot(pthread_t tid, char team)
{}
void drive(pthread_t tid, char team)
{}

// Fan A thread
void fanA_thread() {
   wait(mutex);
   fanA += 1;
   printf();
   if (fanA == 4) {
      signal(fanA_queue, 4);
      fanA = 0;
      isCaptain = true;
   } else if (fanA == 2 && fanB >= 2) {
      signal(fanA_queue, 2);
      signal(fanB_queue, 2);
      fanB -= 2;
      fanA = 0;
      isCaptain = true;
   } else {
      signal(mutex);
   }
   wait(fanA_queue);
   // Barrier synchronization
   wait(barrier);
   spot();
   wait(barrier2); // Signal for reaching this point
   wait(captainSem);
   if (isCaptain) {
      drive();
      carCounter += 1;
      isCaptain = false;
      signal(mutex);
   }
   signal(captainSem);
   wait(barrier3);
   return NULL;
}
```

```
// Fan B thread
void fanB_thread() {
   wait(mutex);
   fanB += 1;
   printf();
   if (fanB == 4) {
      signal(fanB_queue, 4);
      fanB = 0;
      isCaptain = true;
   } else if (fanB == 2 && fanA >= 2) {
      signal(fanB_queue, 2);
      signal(fanA_queue, 2);
      fanA -= 2;
      fanB = 0;
      isCaptain = true;
   } else {
      signal(mutex);
   }
   wait(fanB_queue);
   // Barrier synchronization
   wait(barrier);
   spot();
   wait(barrier2);

   wait(captainSem);
   if (isCaptain) {
      drive();
      carCounter += 1;
      isCaptain = false;
      signal(mutex);
   }
   signal(captainSem);
   wait(barrier3);
   return NULL;
}

// Main function
int main(int argc, char* argv[]) {
   // Input validation
   // Initialization of semaphores and barriers
   // Thread creation
   // Wait for threads to finish
   // Print termination message
   // Semaphore and barrier destruction
}
```

In my program, I primarily used semaphores and barriers for synchronization. I have used 4 Semaphores in my program. Mutex(Binary) Semaphore protects critical sections of code to ensure that only one thread can access shared resources, such as the fanA and fanB counters. This semaphore is essential for avoiding race conditions and ensuring correct updates to shared variables. Counting Semaphores (fanA_queue, fanB_queue) used to control the flow of threads. These semaphores help implement the logic of waiting for a specific number and combination of threads to reach a certain point before continuing with the rest of the program. When the number of fanA threads or/and fanB threads reach the desired combination these counting semaphores signal the right combination of threads to wake up and continue with the rest of the program. Barriers (barrier, barrier2, barrier3) used to synchronize multiple threads. After the right combination of threads wakes up the first barrier ensures that all threads print the first line of " I am looking for a car" and reach a certain point before proceeding, the second barrier ensures that all threads print the "I have found a spot in a car" before they enter the captain logic, and the third barrier ensures all threads complete their tasks before the main thread terminates. Conditional Logic is the “isCaptain” boolean variable this variable is used to determine whether a thread should act as a captain. When the last thread of a band comes it makes the isCaptain variable true for all threads because they all share the same memory. But only one thread is supposed to be a captain that is why the captain logic is guarded by the Mutex (Binary) semaphore to prevent race conditions. So that only one thread can be the captain and then the captain thread makes the isCaptain false so that no other thread can be a captain. The carCounter is protected by the mutex semaphore, ensuring that each band increments it in a mutually exclusive manner. This guarantees the uniqueness and consistency of car IDs.

The use of semaphores ensures thread safety, preventing race conditions and ensuring that shared resources are accessed in a mutually exclusive manner. Specifically, the use of mutex semaphores ensures critical section protection, while counting semaphores facilitates controlled thread synchronization. The combination of barriers and conditional logic ensures that threads execute in a specific order, meeting the specified correctness criteria for the order and interleaving of printed strings.