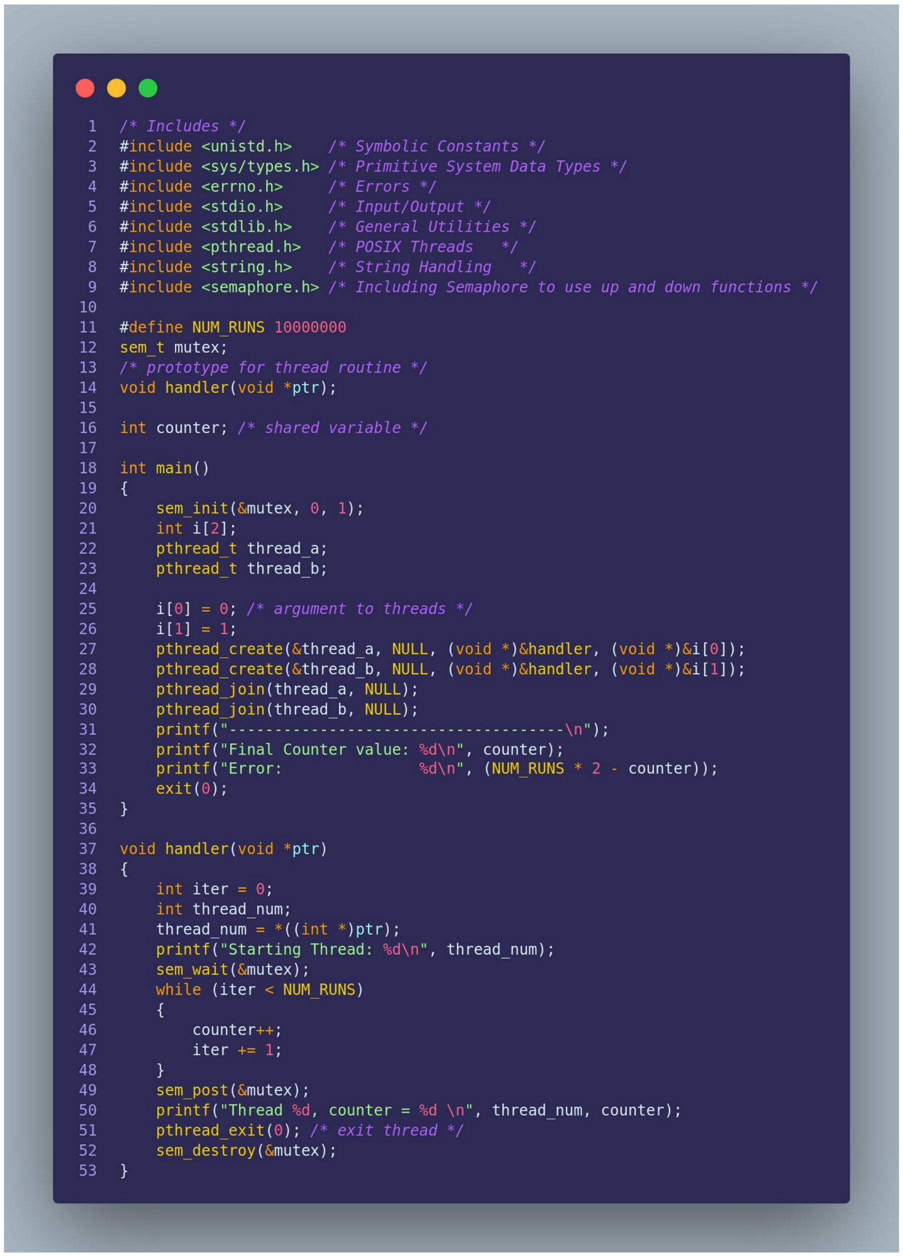
NAME : JAWAD AHMED
ROLL NO: 20P-0165
SECTION: BCS-4A

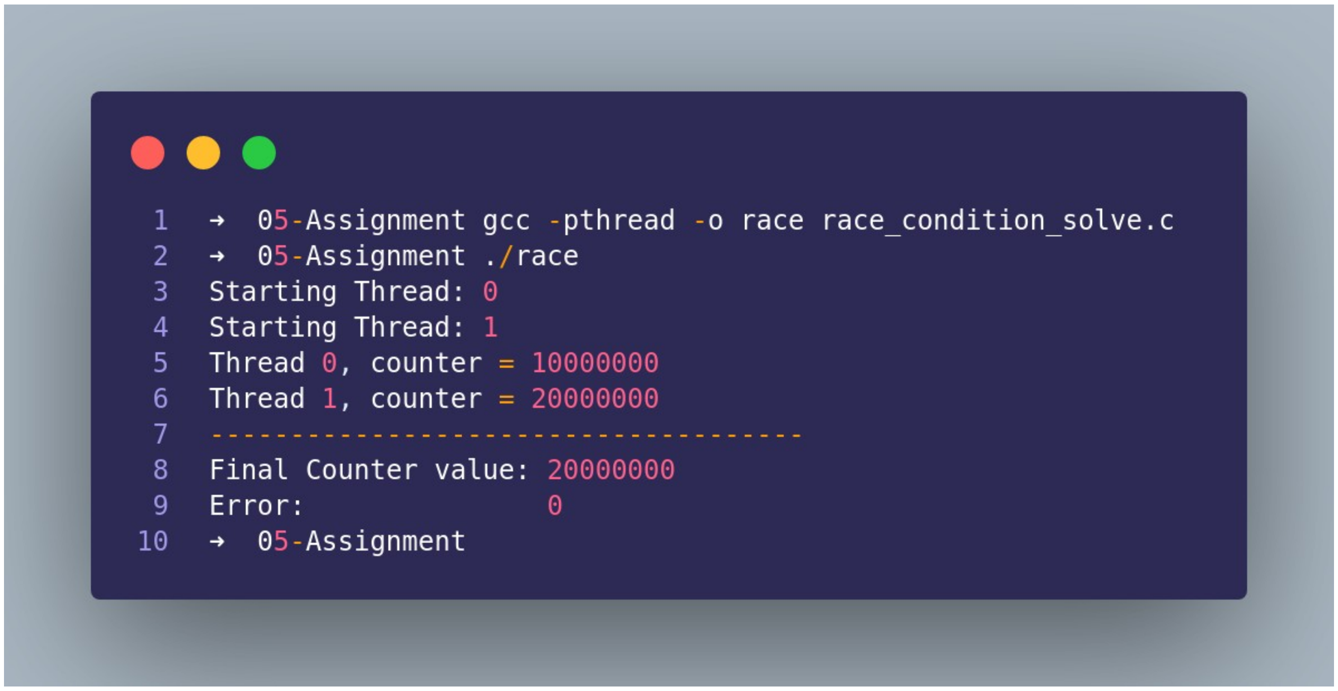
<u>ASSIGNMENT NO 5</u>

1:



```c
/* Includes */
#include <unistd.h>    /* Symbolic Constants */
#include <sys/types.h> /* Primitive System Data Types */
#include <errno.h>     /* Errors */
#include <stdio.h>     /* Input/Output */
#include <stdlib.h>    /* General Utilities */
#include <pthread.h>   /* POSIX Threads   */
#include <string.h>    /* String Handling   */
#include <semaphore.h> /* Including Semaphore to use up and down functions */

#define NUM_RUNS 10000000
sem_t mutex;
/* prototype for thread routine */
void handler(void *ptr);

int counter; /* shared variable */

int main()
{
    sem_init(&mutex, 0, 1);
    int i[2];
    pthread_t thread_a;
    pthread_t thread_b;

    i[0] = 0; /* argument to threads */
    i[1] = 1;
    pthread_create(&thread_a, NULL, (void *)&handler, (void *)&i[0]);
    pthread_create(&thread_b, NULL, (void *)&handler, (void *)&i[1]);
    pthread_join(thread_a, NULL);
    pthread_join(thread_b, NULL);
    printf("-------------------------------------\n");
    printf("Final Counter value: %d\n", counter);
    printf("Error:               %d\n", (NUM_RUNS * 2 - counter));
    exit(0);
}

void handler(void *ptr)
{
    int iter = 0;
    int thread_num;
    thread_num = *((int *)ptr);
    printf("Starting Thread: %d\n", thread_num);
    sem_wait(&mutex);
    while (iter < NUM_RUNS)
    {
        counter++;
        iter += 1;
    }
    sem_post(&mutex);
    printf("Thread %d, counter = %d \n", thread_num, counter);
    pthread_exit(0); /* exit thread */
    sem_destroy(&mutex);
}
```

2:



```
➜  05-Assignment gcc -pthread -o race race_condition_solve.c
➜  05-Assignment ./race
Starting Thread: 0
Starting Thread: 1
Thread 0, counter = 10000000
Thread 1, counter = 20000000
------------------------------------
Final Counter value: 20000000
Error:               0
➜  05-Assignment
```

1. The critical section of the code is the while loop counter increment instruction we want critical section code to be synchronized. So we used the locks on the while loop in this way we have eliminated the error. **SEMAPHORE** is the variable that store the information about how many turns to give to thread. In this case we have used the Binary Semaphore. When one thread acquire the lock then the other thread will not able to go that critical section and when it complete it's work control given back to the other thread in this way both threads synchronizely do their task.

2. There are different ways you can do this One way is that you can add the semaphore into the while loop in counter increment instruction but the problem with it is that every time to loop run semaphore executed and we don't want that.