ASSIGNMENT # 5

NAME : SHAHERYAR ASHFAQ

ROLL NO : **20P-0128**

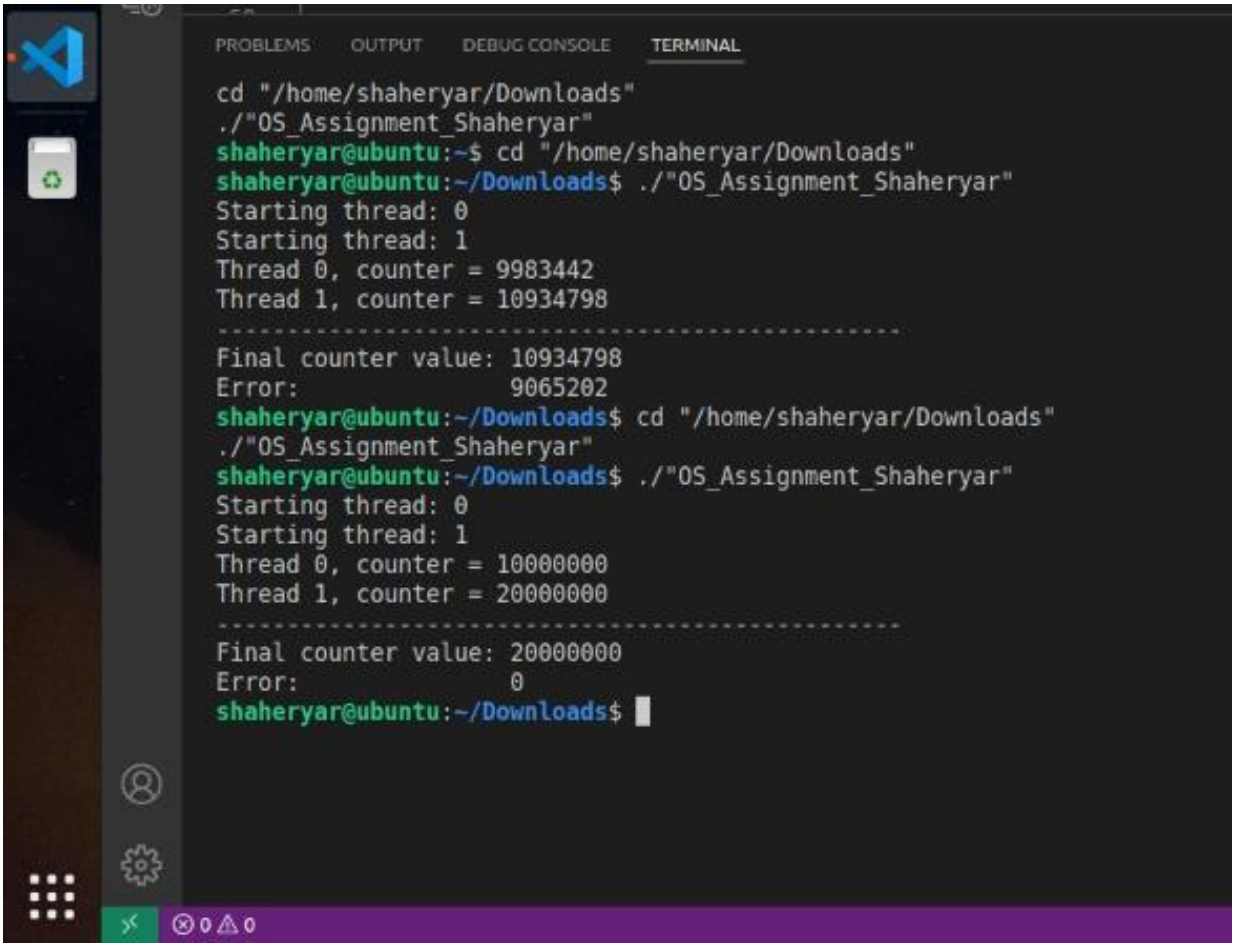
SECTION : BS-CS **4-B**

SUBJECT : OPERATING SYSTEM

```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<pthread.h>
#include<semaphore.h>
#include<sys/types.h>
#include<errno.h>

#define NUM_RUNS 10000000

#define THREAD_NUMS 2

void* handler(void *ptr);

int counter;

sem_t mutex;        // Creating semaphore

int main() {

    int i[THREAD_NUMS];
    pthread_t th[THREAD_NUMS];

    sem_init(&mutex, 0, 1); // Initializing semaphore with starting value of 1

    i[0] = 0;
    i[1] = 1;

    for(int j = 0; j < THREAD_NUMS; j++) {
        if (pthread_create(&th[j], NULL, &handler, &i[j]) != 0) {
            perror("Failed to create thread");
        }
    }
```

```c
    for (int j = 0; j < THREAD_NUMS; j++) {
        if (pthread_join(th[j], NULL) != 0) {
            perror("Failed to join thread");
        }
    }

    printf("-----------------------------------------------\n");
    printf("Final counter value: %d\n", counter);
    printf("Error:               %d\n", (NUM_RUNS * 2 - counter));

    sem_destroy(&mutex);

    return 0;
}

void* handler(void *ptr) {

    int iter = 0;
    int *thread_num = malloc( sizeof(int) );
    thread_num = (int *) ptr;

    printf("Starting thread: %d\n", *thread_num );

    sem_wait(&mutex);

    while(iter < NUM_RUNS) {
        counter++;
        iter++;
    }

    sem_post(&mutex);

    printf("Thread %d, counter = %d\n", *thread_num, counter);

    pthread_exit(0);
}
```

OUTPUT



Explanation

Locks are one synchronization technique. A lock is an abstraction that allows at most one thread to *own* it at a time. *Holding a lock* is how one thread tells other threads: “I’m changing this thing, don’t touch it right now.”