National Textile University

Department of Computer Science

Subject:
Operating System

Submitted to:
Sir Nasir Mehmood

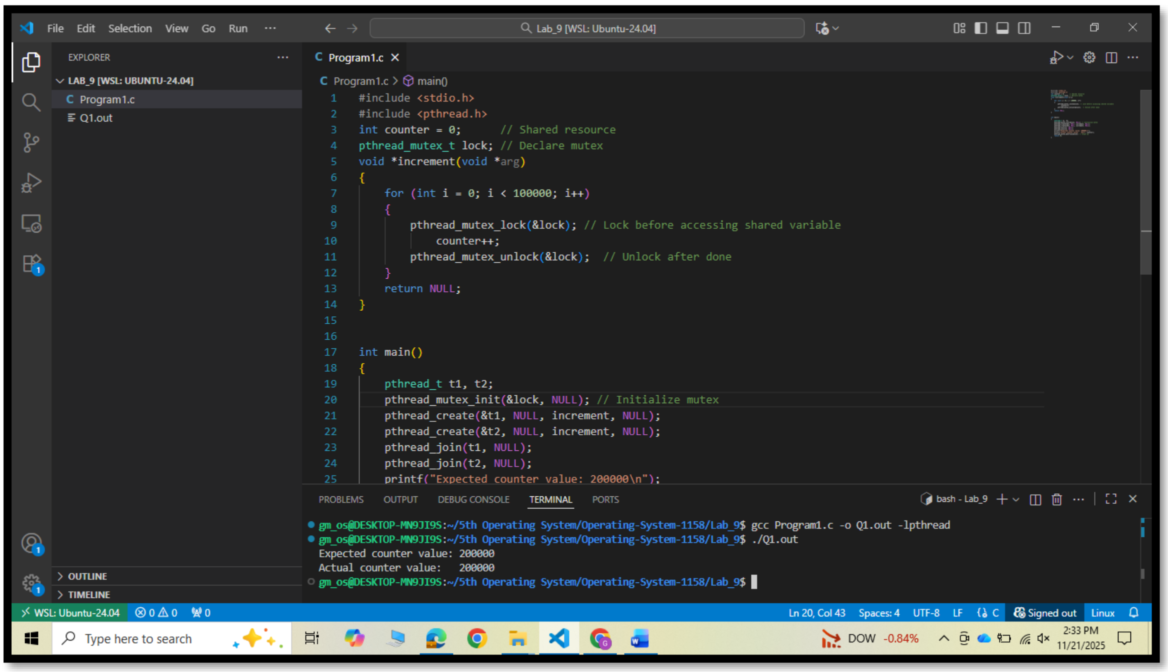
Submitted by:
Ghulam Mohyuddin

Reg number:
23-NTU-CS-1158

Lab number:
9th

Semester:
5th

Program#1: Fixing Race Condition using Mutex

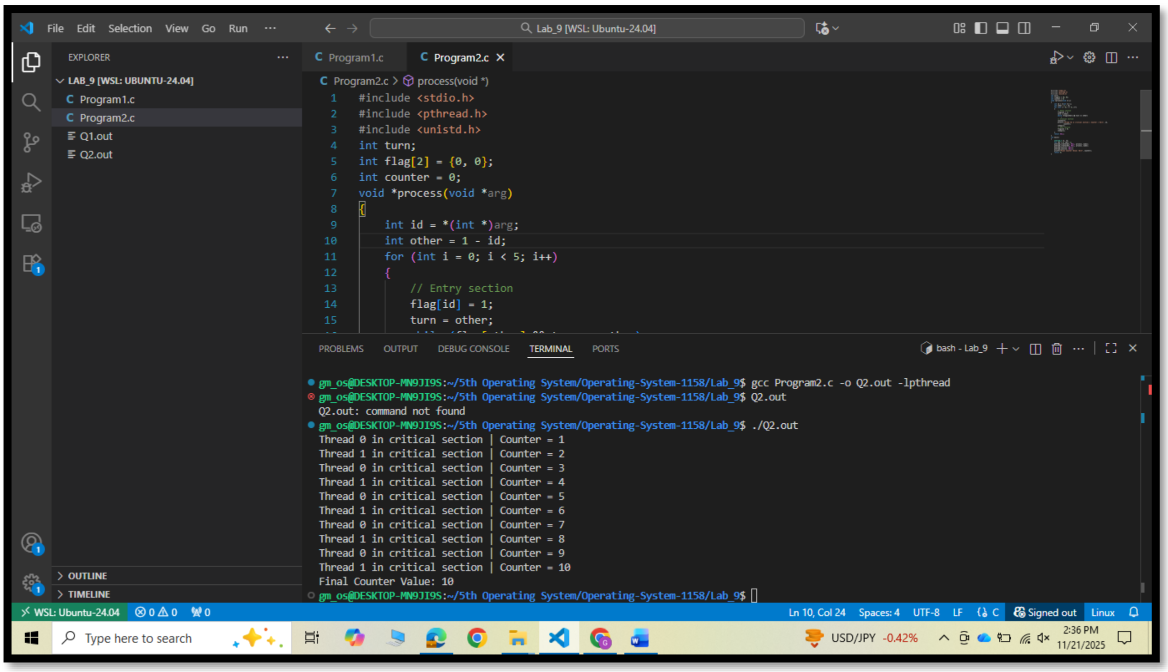


```c
#include <stdio.h>
#include <pthread.h>
int counter = 0;       // Shared resource
pthread_mutex_t lock; // Declare mutex
void *increment(void *arg)
{
    for (int i = 0; i < 100000; i++)
    {
        pthread_mutex_lock(&lock); // Lock before accessing shared variable
            counter++;
        pthread_mutex_unlock(&lock);  // Unlock after done
    }
    return NULL;
}


int main()
{
    pthread_t t1, t2;
    pthread_mutex_init(&lock, NULL); // Initialize mutex
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Expected counter value: 200000\n");
```

```
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program1.c -o Q1.out -lpthread
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q1.out
Expected counter value: 200000
Actual counter value:   200000
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

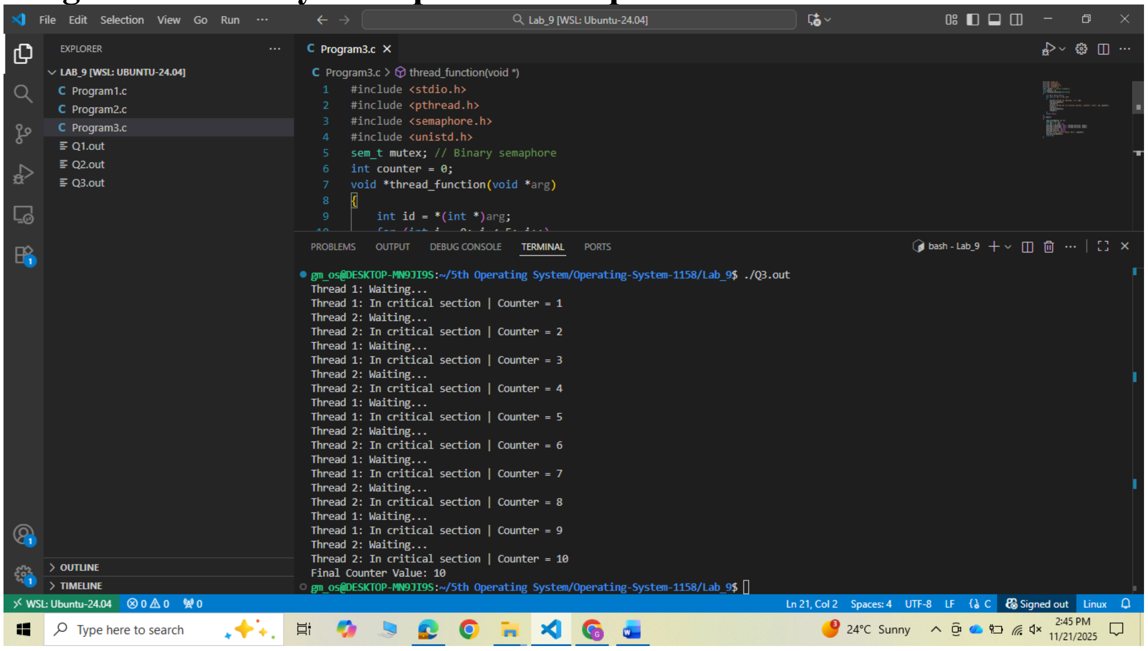
Program#2: Peterson’s Algorithm Simulation



```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
int turn;
int flag[2] = {0, 0};
int counter = 0;
void *process(void *arg)
{
    int id = *(int *)arg;
    int other = 1 - id;
    for (int i = 0; i < 5; i++)
    {
        // Entry section
        flag[id] = 1;
        turn = other;
```

```
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program2.c -o Q2.out -lpthread
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ Q2.out
Q2.out: command not found
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q2.out
Thread 0 in critical section | Counter = 1
Thread 1 in critical section | Counter = 2
Thread 0 in critical section | Counter = 3
Thread 1 in critical section | Counter = 4
Thread 0 in critical section | Counter = 5
Thread 1 in critical section | Counter = 6
Thread 0 in critical section | Counter = 7
Thread 1 in critical section | Counter = 8
Thread 0 in critical section | Counter = 9
Thread 1 in critical section | Counter = 10
Final Counter Value: 10
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Program#3: Binary Semaphore Example

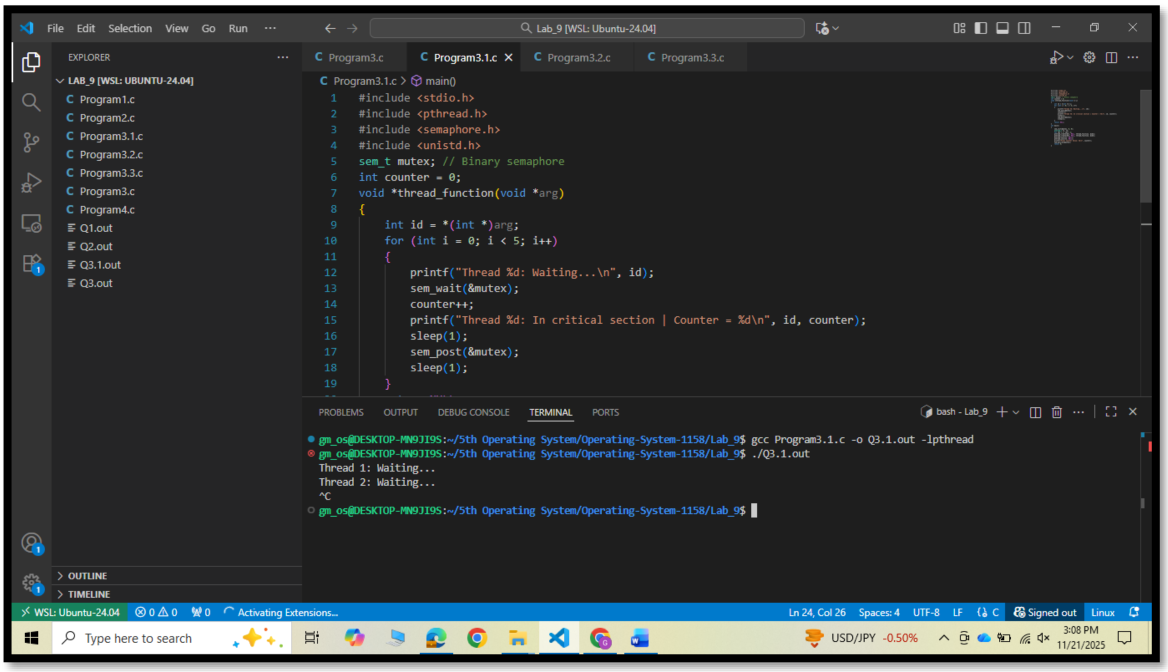


```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void *thread_function(void *arg)
{
    int id = *(int *)arg;
```

```
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q3.out
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 2
Thread 1: Waiting...
Thread 1: In critical section | Counter = 3
Thread 2: Waiting...
Thread 2: In critical section | Counter = 4
Thread 1: Waiting...
Thread 1: In critical section | Counter = 5
Thread 2: Waiting...
Thread 2: In critical section | Counter = 6
Thread 1: Waiting...
Thread 1: In critical section | Counter = 7
Thread 2: Waiting...
Thread 2: In critical section | Counter = 8
Thread 1: Waiting...
Thread 1: In critical section | Counter = 9
Thread 2: Waiting...
Thread 2: In critical section | Counter = 10
Final Counter Value: 10
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Description:
As the Semaphore value is 1, so it run the Wait, and then decrease S by 1 and enter to critical section.

Program#3.1

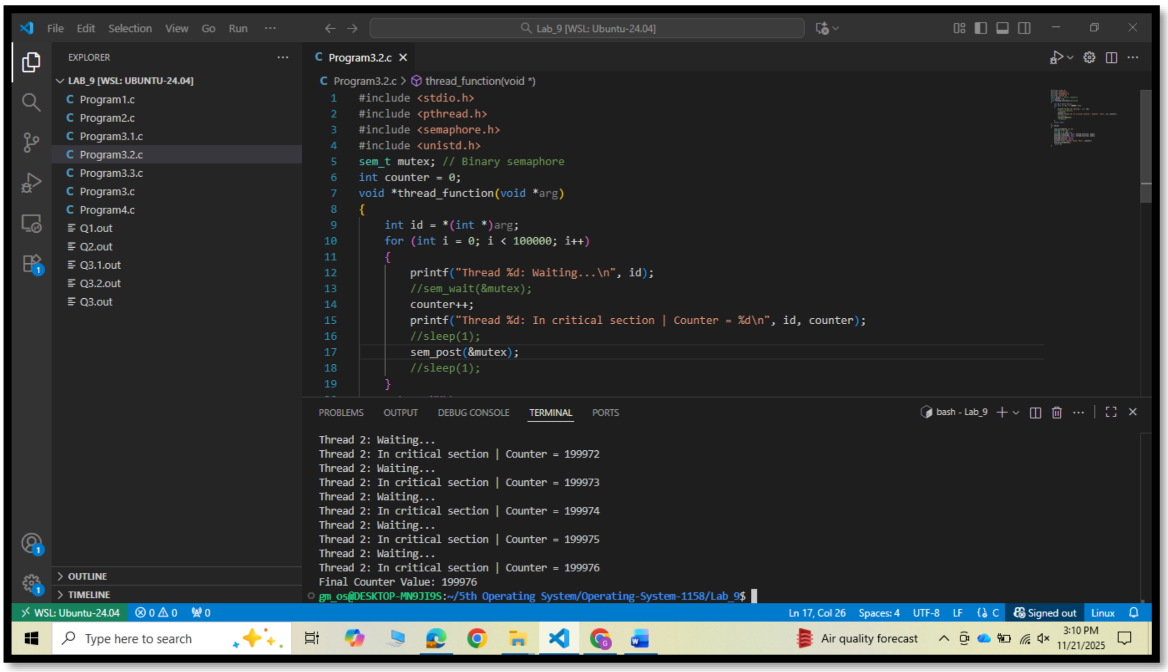


```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void *thread_function(void *arg)
{
    int id = *(int *)arg;
    for (int i = 0; i < 5; i++)
    {
        printf("Thread %d: Waiting...\n", id);
        sem_wait(&mutex);
        counter++;
        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
        sleep(1);
        sem_post(&mutex);
        sleep(1);
    }
```

```
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program3.1.c -o Q3.1.out -lpthread
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q3.1.out
Thread 1: Waiting...
Thread 2: Waiting...
^C
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Description:
As S is zero, so the condition of Wait is not true, so we All the process are in waiting.

Program#3.2

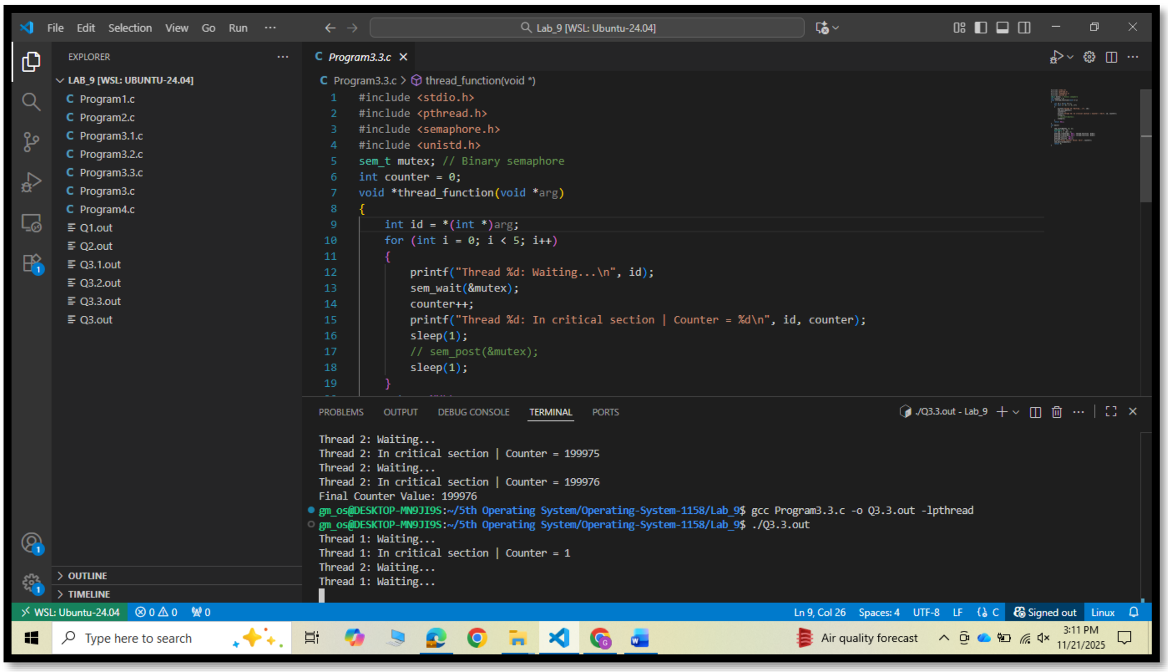


```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void *thread_function(void *arg)
{
    int id = *(int *)arg;
    for (int i = 0; i < 100000; i++)
    {
        printf("Thread %d: Waiting...\n", id);
        //sem_wait(&mutex);
        counter++;
        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
        //sleep(1);
        sem_post(&mutex);
        //sleep(1);
    }
```

```
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199972
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199973
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199974
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199975
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199976
Final Counter Value: 199976
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Description:
After changing loop iteration and, commenting Wait and Sleep, it provides the wrong value, the value of thread must be the iteration of loop but it comes wrong.

Program#3.3

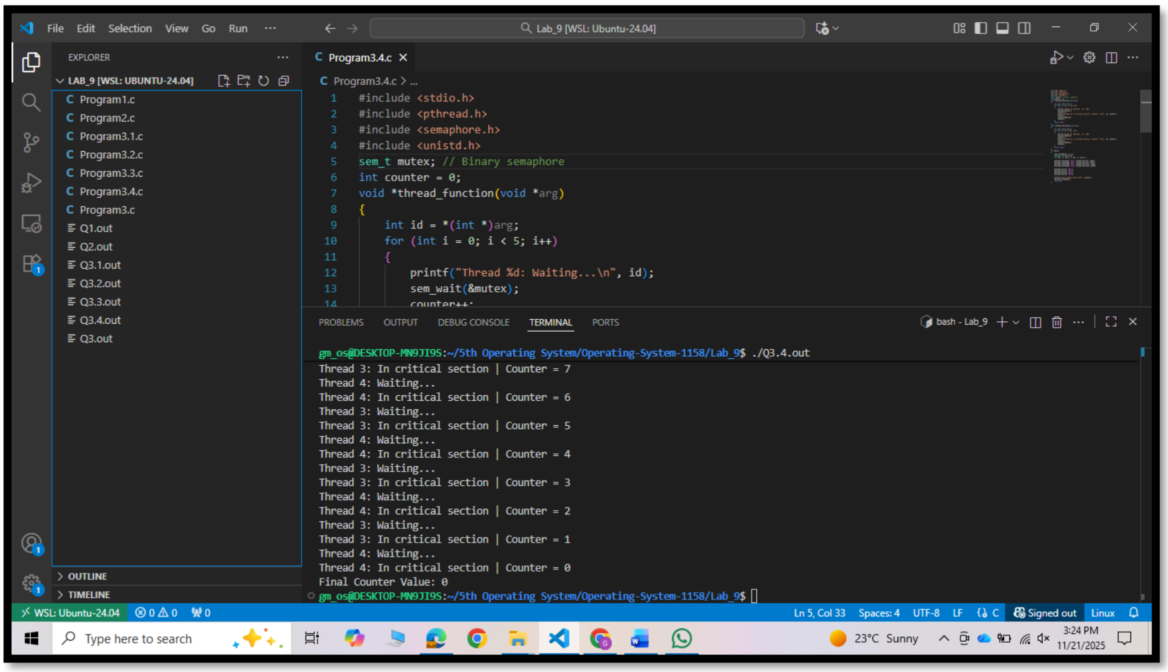


```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void *thread_function(void *arg)
{
    int id = *(int *)arg;
    for (int i = 0; i < 5; i++)
    {
        printf("Thread %d: Waiting...\n", id);
        sem_wait(&mutex);
        counter++;
        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
        sleep(1);
        // sem_post(&mutex);
        sleep(1);
    }
```

```
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199975
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199976
Final Counter Value: 199976
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program3.3.c -o Q3.3.out -lpthread
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q3.3.out
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
```

Description:
After commenting Signal, the value of S in not increase so the value of Wait is false and all process goes on Waiting.

Program#3.3



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void *thread_function(void *arg)
{
    int id = *(int *)arg;
    for (int i = 0; i < 5; i++)
    {
        printf("Thread %d: Waiting...\n", id);
        sem_wait(&mutex);
        counter++;
```

```
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q3.4.out
Thread 3: In critical section | Counter = 7
Thread 4: Waiting...
Thread 4: In critical section | Counter = 6
Thread 3: Waiting...
Thread 3: In critical section | Counter = 5
Thread 4: Waiting...
Thread 4: In critical section | Counter = 4
Thread 3: Waiting...
Thread 3: In critical section | Counter = 3
Thread 4: Waiting...
Thread 4: In critical section | Counter = 2
Thread 3: Waiting...
Thread 3: In critical section | Counter = 1
Thread 4: Waiting...
Thread 4: In critical section | Counter = 0
Final Counter Value: 0
gm_os@DESKTOP-MN9JI9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Description:

- ❖ Create a new function in which decrementing the counter, so the value after whole execution of Thread is 0.
- ❖ Because in thread_function counter is incrementing by 1 in each iteration, and in thread_function1 counter is decrementing by 1 in each iteration, so after the whole execution of main the counter becomes 0 again.

Task#4

Mutex:	**Semaphore:**
Mutex used lock and unlock for entry in critical section, or for synchronization.	Semaphore uses Wait and Signal for synchronization.
Mutex allows only one thread to change the critical section at a time.	Semaphore use Wait to allow a thread to change if the condition becomes true.
Mutex ownership is the thread that lock it only unlock it.	Semaphore use Signal to enter another thread/process.