



National Textile University

Department of Computer Science

Subject:
Operating System

Submitted to:
Sir Nasir Mehmood

Submitted by:
Gulam Rasool

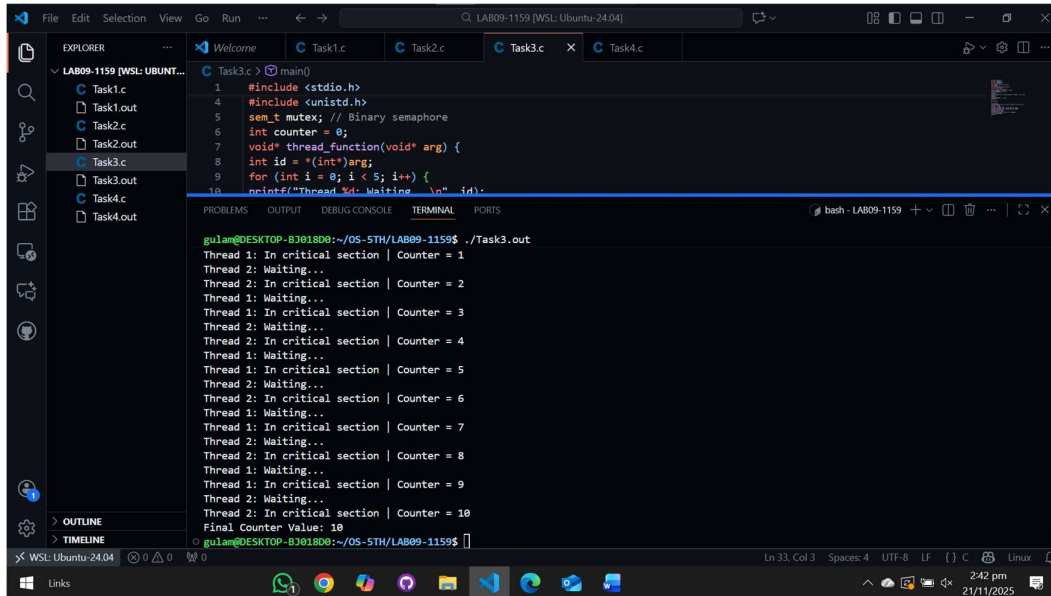
Reg number:
23-NTU-CS-1159

Lab no:
09

Section:
BSSE-(A) – 5th

LAB-09

Program-3:

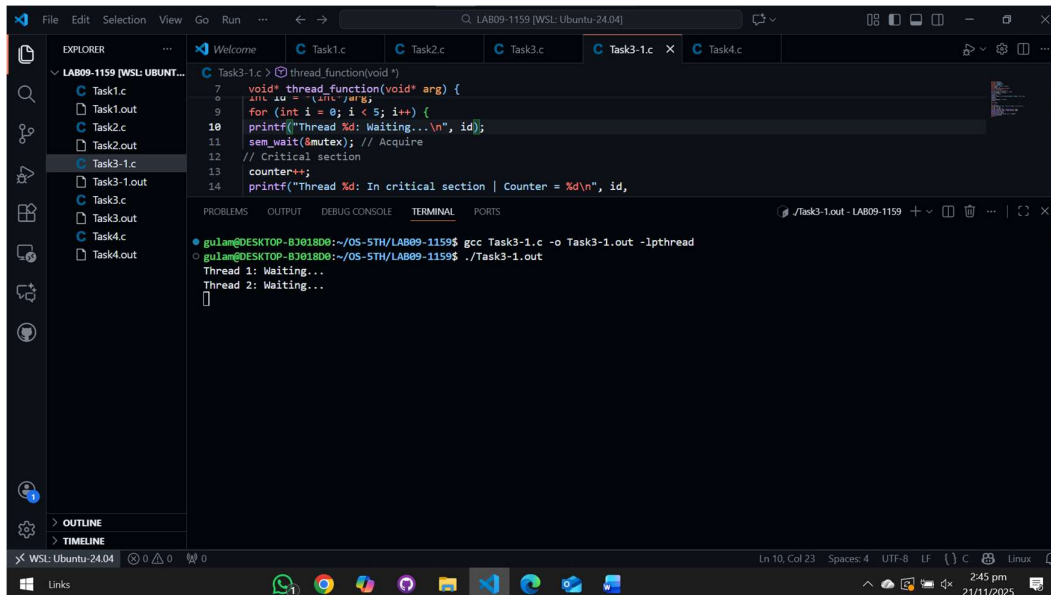


```
1 #include <stdio.h>
2 #include <unistd.h>
3 sem_t mutex; // Binary semaphore
4 int counter = 0;
5 void* thread_function(void* arg) {
6     int id = *(int*)arg;
7     for (int i = 0; i < 5; i++) {
8         printf("Thread %d: Waiting. \n", id);
9     }
10 }
```

```
gulam@DESKTOP-B701BD0:~/OS-5TH/LAB09-1159$ ./Task3.out
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: In critical section | Counter = 10
Final Counter Value: 10
gulam@DESKTOP-B701BD0:~/OS-5TH/LAB09-1159$
```

- When SEM is initialized to 1 :
- The Threads wait for each other until find resources free .

Program-3.1:



The screenshot shows a Visual Studio Code editor window with a C program named `Task3-1.c`. The program uses a semaphore to control access to a critical section. Two threads, `Thread 1` and `Thread 2`, are created and both attempt to enter the critical section. The terminal output shows that both threads are in a waiting state.

```
7 void* thread_function(void* arg) {
8     int id = (int)arg;
9     for (int i = 0; i < 5; i++) {
10        printf("Thread %d: Waiting...\n", id);
11        sem_wait(&mutex); // Acquire
12        // Critical section
13        counter++;
14        printf("Thread %d: In critical section | Counter = %d\n", id,
```

```
gulin@DESKTOP-8701BD0: ~/05-5TH/LAB09-1159$ gcc Task3-1.c -o Task3-1.out -lpthread
gulin@DESKTOP-8701BD0: ~/05-5TH/LAB09-1159$ ./Task3-1.out
Thread 1: Waiting...
Thread 2: Waiting...
[]
```

- When SEM is initialized to 0:
- No thread/Function enter the Critical Function Both wait because already in Waiting both wait.

Program 3-2:

```
1 #include <stdio.h>

Thread 2: In critical section | Counter = 199973
Thread 2: In critical section | Counter = 199974
Thread 2: In critical section | Counter = 199975
Thread 2: In critical section | Counter = 199976
Thread 2: In critical section | Counter = 199977
Thread 2: In critical section | Counter = 199978
Thread 2: In critical section | Counter = 199979
Thread 2: In critical section | Counter = 199980
Thread 2: In critical section | Counter = 199981
Thread 2: In critical section | Counter = 199982
Thread 2: In critical section | Counter = 199983
Thread 2: In critical section | Counter = 199984
Thread 2: In critical section | Counter = 199985
Thread 2: In critical section | Counter = 199986
Thread 2: In critical section | Counter = 199987
Thread 2: In critical section | Counter = 199988
Thread 2: In critical section | Counter = 199989
Thread 2: In critical section | Counter = 199990
Thread 2: In critical section | Counter = 199991
Thread 2: In critical section | Counter = 199992
Thread 2: In critical section | Counter = 199993
Thread 2: In critical section | Counter = 199994
Thread 2: In critical section | Counter = 199995
Thread 2: In critical section | Counter = 199996
Thread 2: In critical section | Counter = 199997
Thread 2: In critical section | Counter = 199998
Thread 2: In critical section | Counter = 199999
Final Counter Value: 199999
```

- **Without Wait:**
- **The System is not Synchronzied and run concurrently without wait.**
- **The value Changed and not remain same to actual answer.**

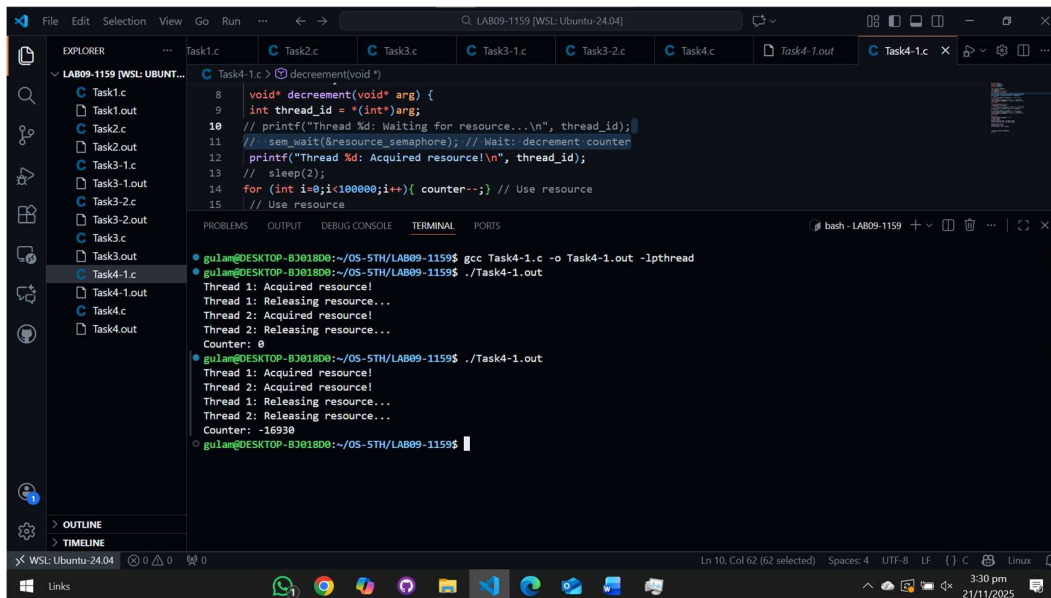
Program-4:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     sem_t resource_semaphore;
6     int counter=0;
7     void* decreement(void* arg) {
8         int thread_id = *(int*)arg;
9         printf("Thread %d: Waiting for resource...\n", thread_id);
10        sem_wait(&resource_semaphore); // Wait: decrement counter
11    }
```

```
gulam@DESKTOP-8701BD0:~/OS-5TH/LAB09-1159$ gcc Task4.c -o Task4.out -lpthread
gulam@DESKTOP-8701BD0:~/OS-5TH/LAB09-1159$ ./Task4.out
Thread 1: Waiting for resource...
Thread 1: Acquired resource!
Thread 2: Waiting for resource...
Thread 2: Acquired resource!
Thread 3: Waiting for resource...
Thread 3: Acquired resource!
Thread 4: Waiting for resource...
Thread 4: Acquired resource!
Thread 2: Releasing resource...
Thread 1: Releasing resource...
Thread 4: Releasing resource...
Thread 3: Releasing resource...
Counter: 0
gulam@DESKTOP-8701BD0:~/OS-5TH/LAB09-1159$
```

- Two functions with increment and decrement:
- Counter value remain same we used semaphore Synchronization.

Program 4.1:



The screenshot shows a Visual Studio Code editor window with a C program named `Task4-1.c` and its output in the terminal. The program uses a semaphore to manage a shared resource, but it contains a race condition where threads do not wait for the semaphore to be released before decrementing the counter.

```
8 void* decrement(void* arg) {
9     int thread_id = *(int*)arg;
10    // printf("Thread %d: Waiting for resource...\n", thread_id);
11    // sem_wait(&resource_semaphore); // Wait: decrement counter
12    printf("Thread %d: Acquired resource!\n", thread_id);
13    // sleep(2);
14    for (int i=0; i<100000; i++) { counter--; } // Use resource
15    // Use resource
```

The terminal output shows the execution of the program, demonstrating a race condition where the counter is decremented by multiple threads simultaneously, resulting in a final value of -16990 instead of 0.

```
guilam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$ gcc Task4-1.c -o Task4-1.out -lpthread
guilam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$ ./Task4-1.out
Thread 1: Acquired resource!
Thread 1: Releasing resource...
Thread 2: Acquired resource!
Thread 2: Releasing resource...
Counter: 0
guilam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$ ./Task4-1.out
Thread 1: Acquired resource!
Thread 2: Acquired resource!
Thread 1: Releasing resource...
Thread 2: Releasing resource...
Counter: -16990
guilam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$
```

- Two separate functions without Wait.
- When Threads don't wait
- Race condition occurs and change the value from the actual value.

Task_3:

Difference between Mutex and Semaphore.

Mutex	Semaphore
Lock & Unlock System	It is Integer Based
Strict Ownership the one which lock can unlock.	Not restrict any other thread can wait() and post()