



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:

```
gulam@DESKTOP-BJ018D0:~/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: Waiting...
Thread 2: In critical section | Counter = 10
Final Counter Value: 10
gulam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$
```

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

Program-3.1:

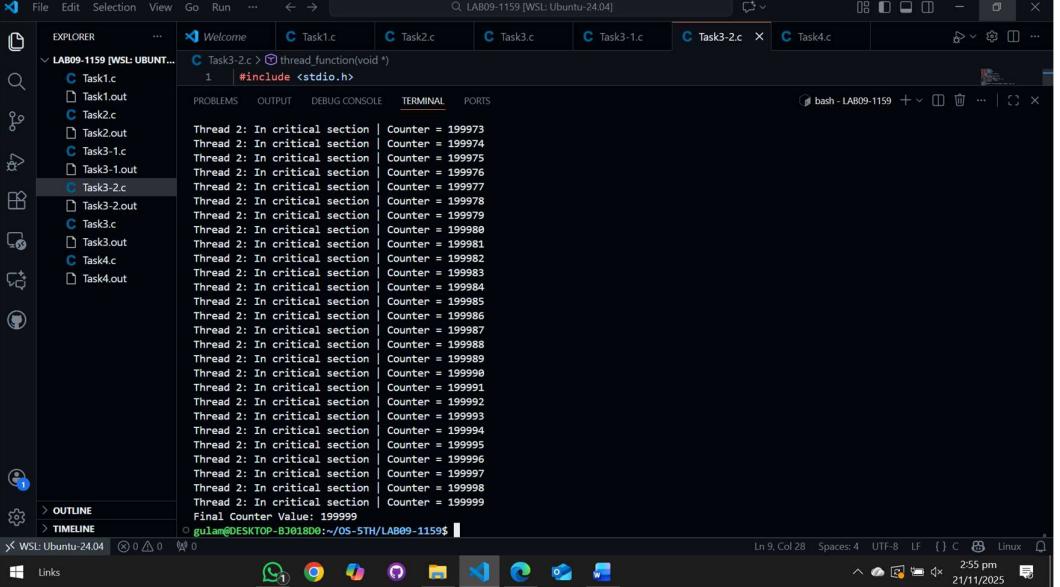
The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The title bar indicates the window is titled 'LAB09-1159 [WSL: Ubuntu-24.04]'. The VS Code interface includes an Explorer sidebar on the left listing files like Task1.c, Task1.out, Task2.c, Task2.out, Task3.c, Task3.out, Task3-1.c, Task3-1.out, Task4.c, and Task4.out. The main editor area displays a C code snippet for a thread function. The code uses a semaphore named 'sem' to manage access to a shared resource. The terminal tab at the bottom shows the command-line output of running the program, which results in both threads waiting indefinitely.

```
void* thread_function(void* arg) {
    int id = (int)arg;
    for (int i = 0; i < 5; i++) {
        printf("Thread %d: Waiting...\n", id);
        sem_wait(&sem); // Acquire
    }
    // Critical section
    counter++;
    printf("Thread %d: In critical section | Counter = %d\n", id,
           counter);
}
```

```
gulam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$ gcc Task3-1.c -o Task3-1.out -lpthread
gulam@DESKTOP-BJ018D0:~/OS-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:



The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The title bar indicates the workspace is "LAB09-1159 [WSL: UBUNTU]". The Explorer sidebar on the left lists files: Task1.c, Task2.c, Task3.c, Task3-1.c, Task3-1.out, Task3-2.c, Task3-2.out, Task3.c, Task3.out, Task4.c, and Task4.out. The Task3-2.c file is selected. The Terminal tab is active, displaying the output of a program. The output shows multiple instances of Thread 2 entering a critical section and incrementing a counter from 199973 to 199999. The final output line is "Final Counter Value: 199999". The status bar at the bottom right shows the date and time as "21/11/2025" and "2:55 pm".

```
#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 Synchronized and run concurrently without wait.**
- **The value Changed and not remain same to actual answer.**

Program-4:

The screenshot shows the Visual Studio Code interface running in WSL. The Explorer sidebar lists files: Task1.c, Task1.out, Task2.c, Task2.out, Task3-1.c, Task3-1.out, Task3-2.c, Task3-2.out, Task3.c, Task3.out, Task4.c, and Task4.out. The Task4.c file is open in the editor, displaying C code that includes stdio.h, defines a semaphore, initializes a counter to 0, and prints thread IDs and resource status. The terminal window shows the command to compile Task4.c and run it, followed by its execution output which shows four threads waiting for a resource, acquiring it, and then releasing it, with the counter value remaining at 0.

```
#include <stdio.h>
sem_t resource_semaphore;
int counter=0;
void* decrement(void* arg) {
    int thread_id = *(int*)arg;
    printf("Thread %d: Waiting for resource...\n", thread_id);
    sem_wait(&resource_semaphore); // Wait: decrement counter
}
void* increment(void* arg) {
    int thread_id = *(int*)arg;
    printf("Thread %d: Acquired resource!\n", thread_id);
    sem_post(&resource_semaphore); // Post: increment counter
}
int main() {
    sem_init(&resource_semaphore, 1, 1);
    pthread_t threads[4];
    for(int i=0; i<4; i++) {
        pthread_create(threads[i], NULL, decrement, (void*)&i);
    }
    for(int i=0; i<4; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Counter: %d\n", counter);
}
```

```
gulam@DESKTOP-BJ018D0:~/OS-5TH/LAB09-1159$ gcc Task4.c -o Task4.out -lpthread
gulam@DESKTOP-BJ018D0:~/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
```

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

Program 4.1:

- 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()