National Textile University

Department of Computer Science

Subject:

Operating System

Submitted To:

Sir Nasir Mehmood

Submitted By:

Hafsa Tayyab

Registration No:

23-NTU-CS-1163

Lab No:

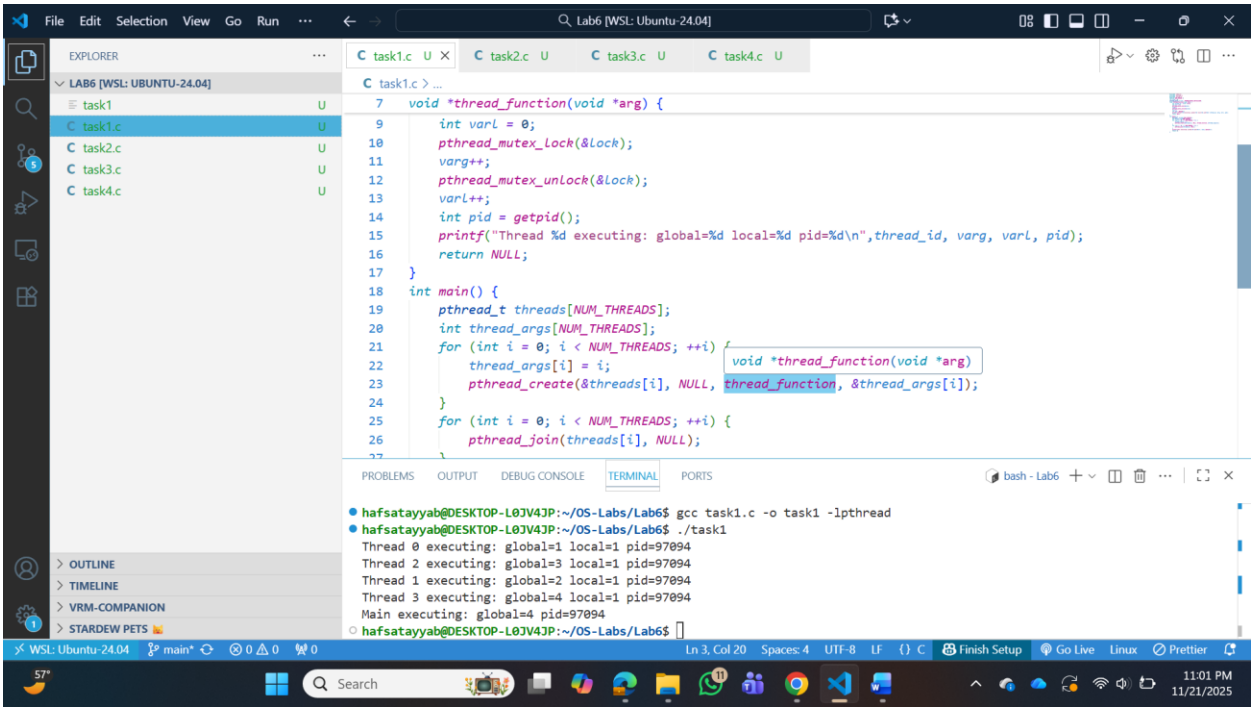
6

Semester:

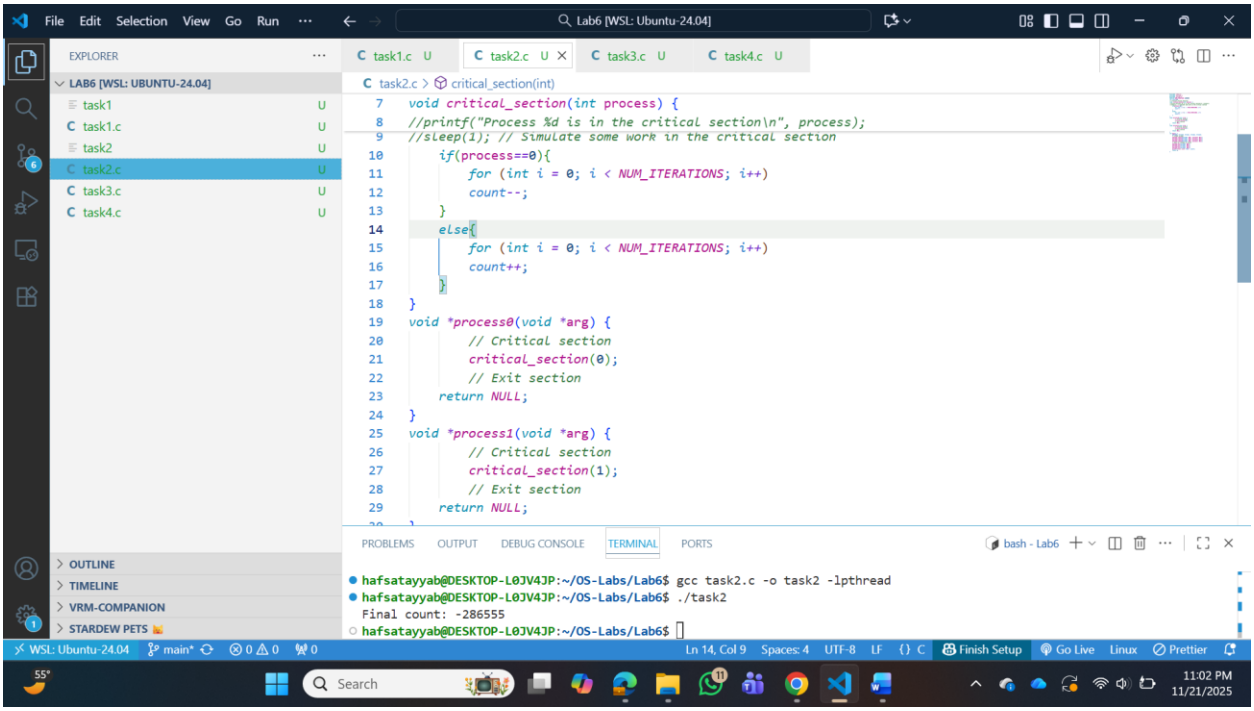
5th

Lab 6: Synchronization

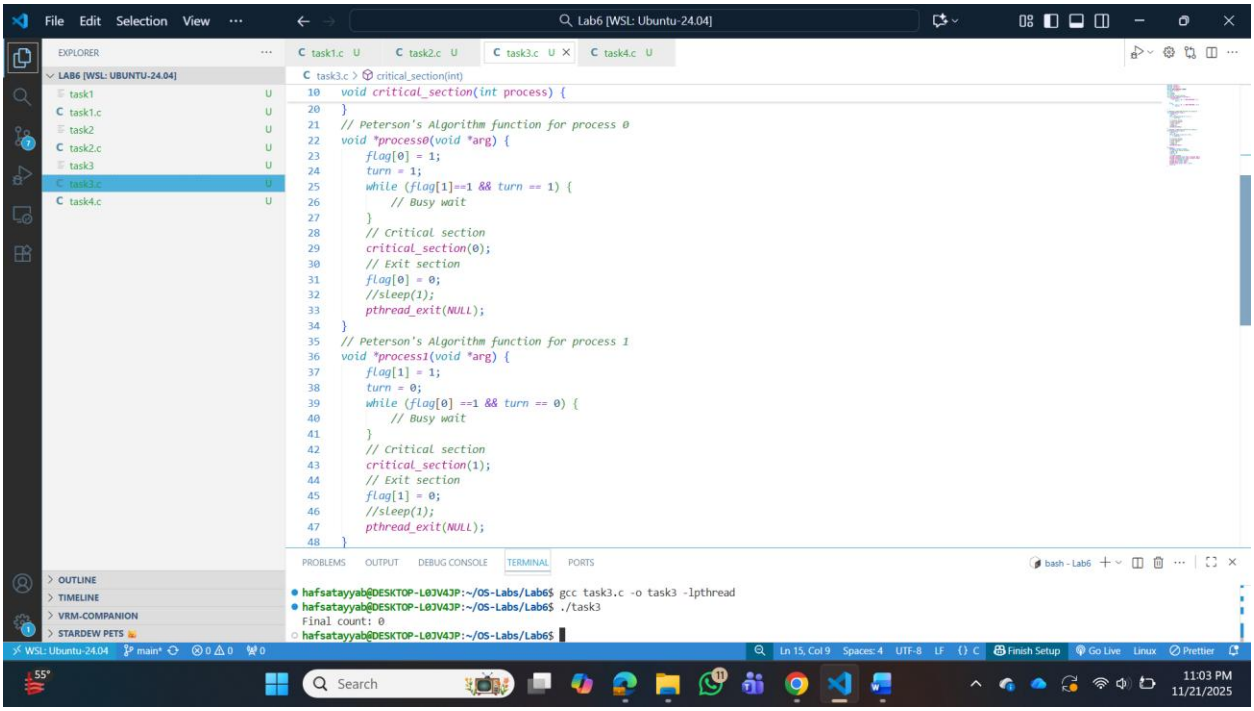
Program 1: Fixing Race Condition using Mutex



Program 2: Without Peterson's Algorithm



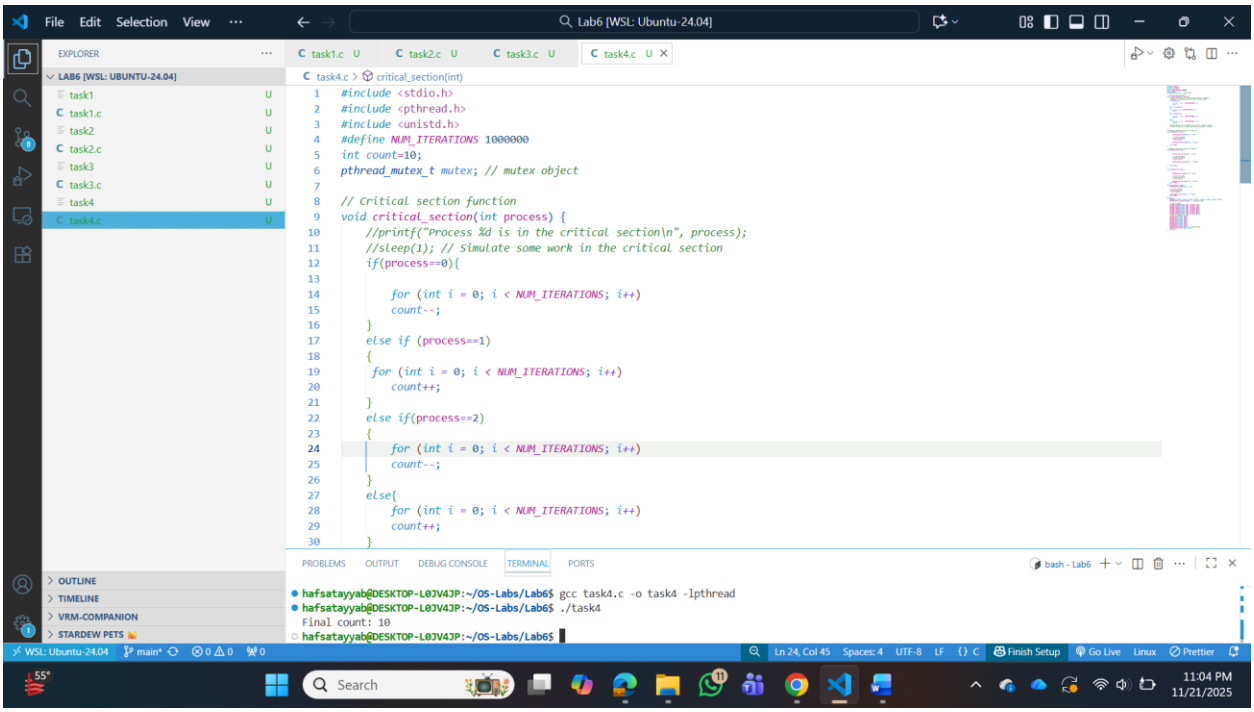
Program 3: With Peterson's Algorithm



```
// Peterson's Algorithm function for process 0
void *process0(void *arg) {
    flag[0] = 1;
    turn = 1;
    while (flag[1]==1 && turn == 1) {
        // Busy wait
    }
    // Critical section
    critical_section(0);
    // Exit section
    flag[0] = 0;
    //sleep(1);
    pthread_exit(NULL);
}
// Peterson's Algorithm function for process 1
void *process1(void *arg) {
    flag[1] = 1;
    turn = 0;
    while (flag[0] ==1 && turn == 0) {
        // Busy wait
    }
    // Critical section
    critical_section(1);
    // Exit section
    flag[1] = 0;
    //sleep(1);
    pthread_exit(NULL);
}
```

```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ gcc task3.c -o task3 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ ./task3
Final count: 0
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$
```

Program 4: With Mutex



```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000
int count=10;
pthread_mutex_t mutex; // mutex object

// Critical section function
void critical_section(int process) {
    //printf("Process %d is in the critical section\n", process);
    //sleep(1); // Simulate some work in the critical section
    if(process==0){

        for (int i = 0; i < NUM_ITERATIONS; i++)
        count--;
    }
    else if (process==1)
    {
     for (int i = 0; i < NUM_ITERATIONS; i++)
        count++;
    }
    else if(process==2)
    {
        for (int i = 0; i < NUM_ITERATIONS; i++)
        count--;
    }
    else{
        for (int i = 0; i < NUM_ITERATIONS; i++)
        count++;
    }
```

```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ gcc task4.c -o task4 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ ./task4
Final count: 10
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$
```

Aspect	Peterson's Algorithm	Mutex (pthread_mutex, OS lock)
Type	Software-only mutual exclusion	OS-supported + hardware-assisted lock
Thread Support	Only 2 threads (classic version)	Any number of threads
Implementation	Uses shared variables: flag[], turn	Uses atomic CPU instructions + kernel support
Correctness on Modern CPUs	Not guaranteed due to weak memory ordering	Guaranteed — includes memory barriers
Performance	Busy waiting (spinlock) → wastes CPU	Blocks when waiting → efficient
Fairness / Bounded Waiting	Guaranteed theoretically	Depends on implementation but generally fair
Reliability	Fragile: affected by compiler optimizations, caching	Very reliable and widely used
Scalability	Poor (only 2 threads, slow)	Excellent
Use Case	Educational purposes only	Real-world synchronization in programs
Ease of Use	Hard to implement correctly	Easy API: pthread_mutex_lock()
Memory Requirements	Few shared variables	Mutex structure (small but more overhead)
Kernel Involvement	None (pure userspace spinlock)	May block and involve scheduler
Where It Works	Only in strictly sequentially consistent models	Works everywhere (Linux, Windows, macOS, multicore CPUs)