



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:

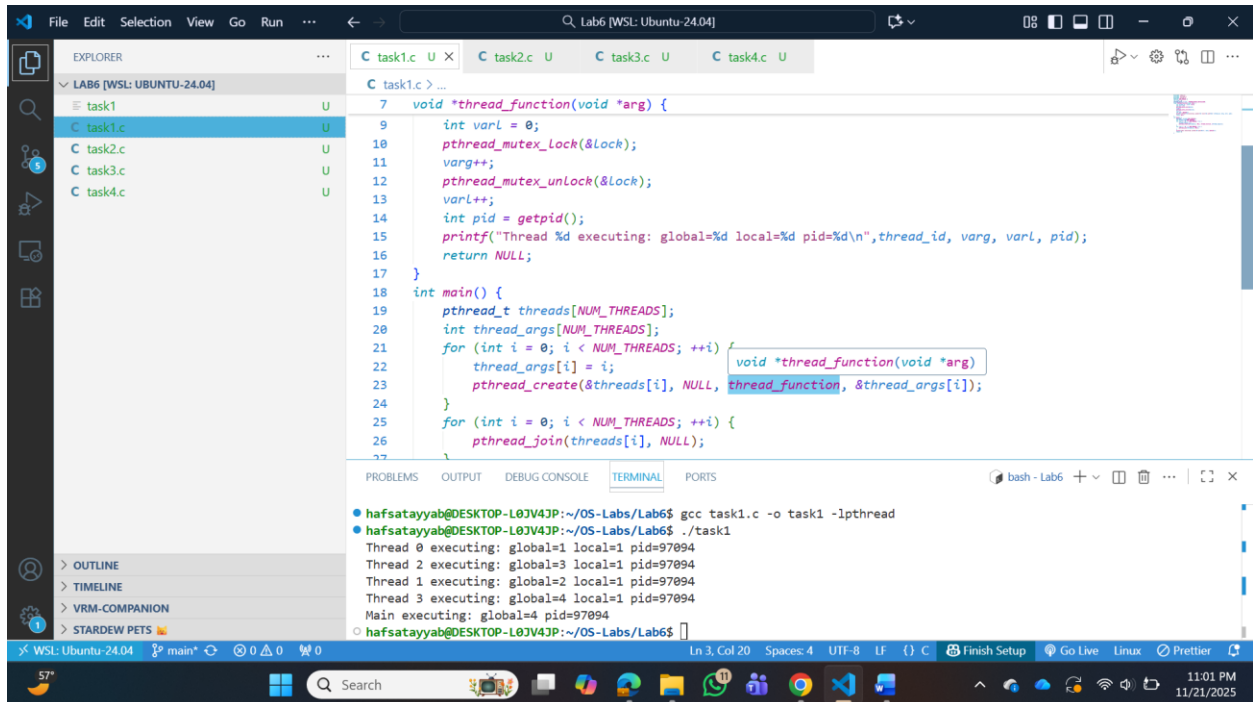
6

Semester:

5th

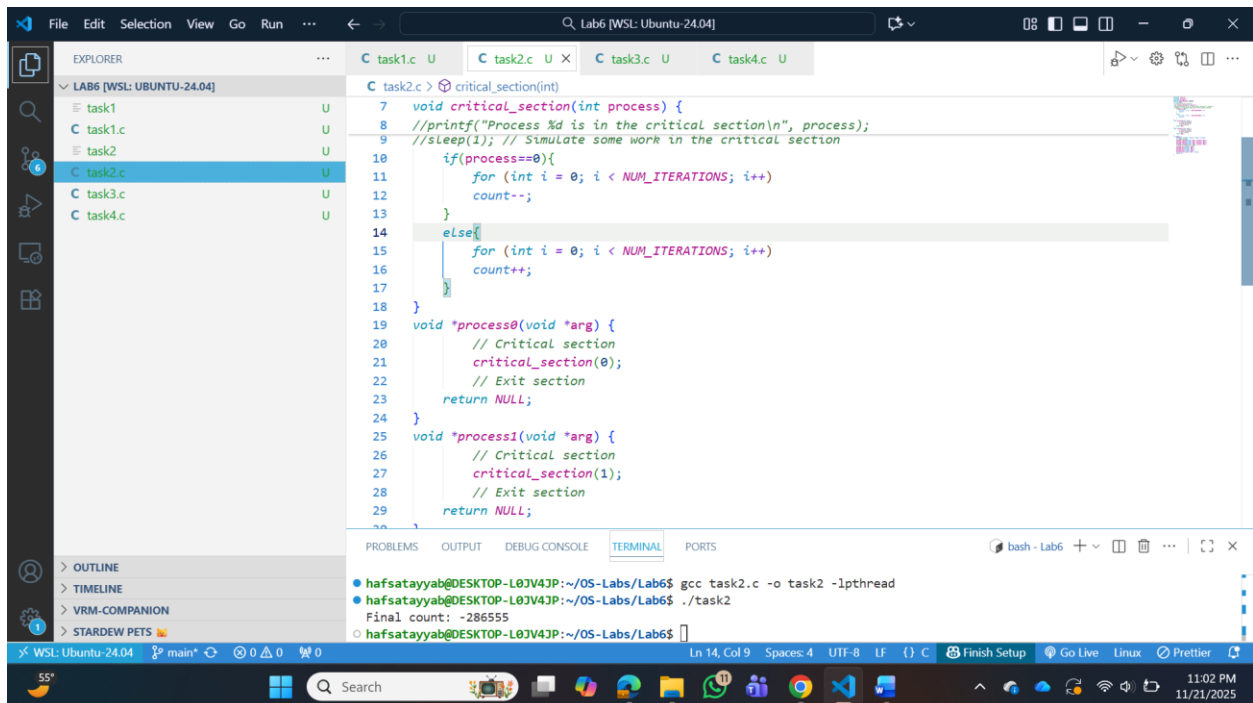
Lab 6: Synchronization

Program 1: Fixing Race Condition using Mutex



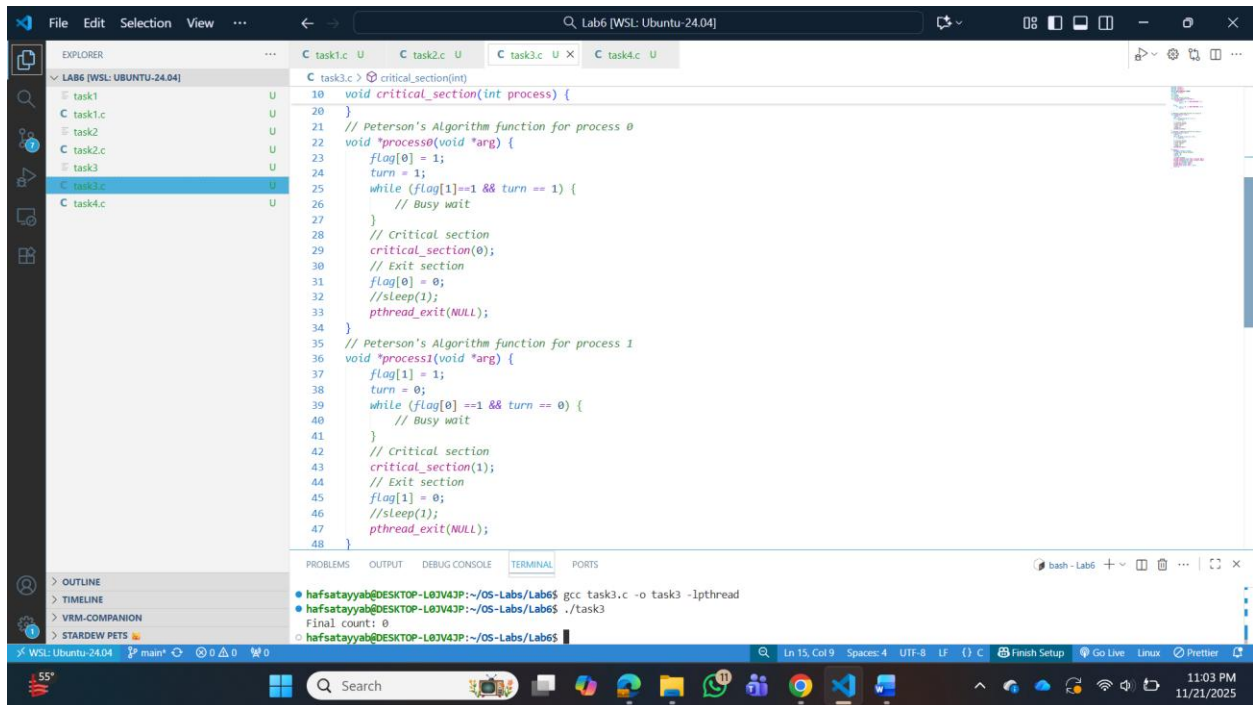
```
File Edit Selection View Go Run ... Lab6 [WSL: Ubuntu-24.04]
EXPLORER
LAB6 [WSL: UBUNTU-24.04]
task1
task1.c
task2.c
task3.c
task4.c
C task1.c > ...
7 void *thread_function(void *arg) {
9     int varl = 0;
10    pthread_mutex_lock(&lock);
11    varl++;
12    pthread_mutex_unlock(&lock);
13    varl++;
14    int pid = getpid();
15    printf("Thread %d executing: global=%d local=%d pid=%d\n", thread_id, varl, pid);
16    return NULL;
17 }
18 int main() {
19    pthread_t threads[NUM_THREADS];
20    int thread_args[NUM_THREADS];
21    for (int i = 0; i < NUM_THREADS; ++i) {
22        thread_args[i] = i;
23        pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
24    }
25    for (int i = 0; i < NUM_THREADS; ++i) {
26        pthread_join(threads[i], NULL);
27    }
28 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Lab6
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ gcc task1.c -o task1 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ ./task1
Thread 0 executing: global=1 local=1 pid=97894
Thread 2 executing: global=3 local=1 pid=97894
Thread 1 executing: global=2 local=1 pid=97894
Thread 3 executing: global=4 local=1 pid=97894
Main executing: global=4 pid=97894
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$
```

Program 2: Without Peterson's Algorithm



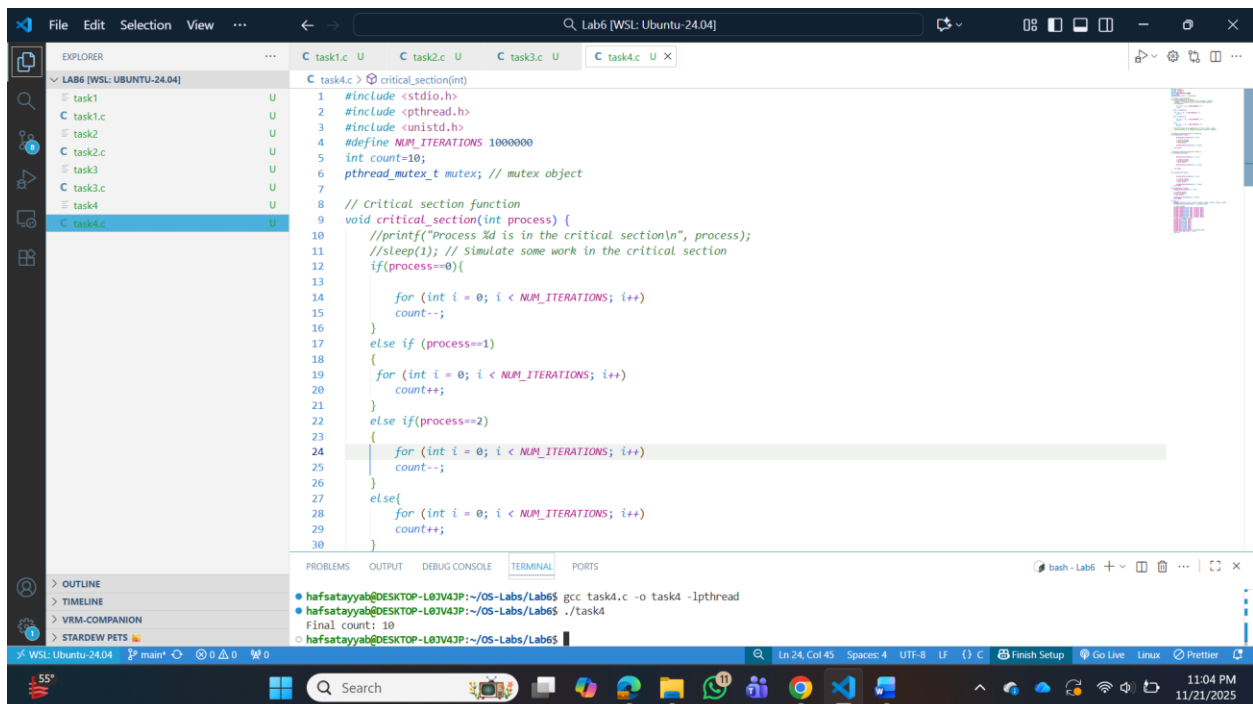
```
File Edit Selection View Go Run ... Lab6 [WSL: Ubuntu-24.04]
EXPLORER
LAB6 [WSL: UBUNTU-24.04]
task1
task1.c
task2.c
task3.c
task4.c
C task2.c > critical_section(int)
7 void critical_section(int process) {
8     //printf("Process %d is in the critical section\n", process);
9     //sleep(1); // Simulate some work in the critical section
10    if(process==0){
11        for (int i = 0; i < NUM_ITERATIONS; ++i)
12            count--;
13    }
14    else{
15        for (int i = 0; i < NUM_ITERATIONS; ++i)
16            count++;
17    }
18 }
19 void *process0(void *arg) {
20     // Critical section
21     critical_section(0);
22     // Exit section
23     return NULL;
24 }
25 void *process1(void *arg) {
26     // Critical section
27     critical_section(1);
28     // Exit section
29     return NULL;
30 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Lab6
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ gcc task2.c -o task2 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$ ./task2
Final count: -286555
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab6$
```

Program 3: With Peterson's Algorithm



```
File Edit Selection View ... Lab6 [WSL: Ubuntu-24.04]
EXPLORER
  LAB6 [WSL: UBUNTU-24.04]
    task1
    task2
    task3
    task4
    task1.c
    task2.c
    task3.c
    task4.c
  task3.c
task3.c
  10 void critical_section(int process) {
  20 }
  21 // Peterson's Algorithm function for process 0
  22 void *process0(void *arg) {
  23     flag[0] = 1;
  24     turn = 1;
  25     while (flag[1]==1 && turn == 1) {
  26         // Busy wait
  27     }
  28     // Critical section
  29     critical_section(0);
  30     // Exit section
  31     flag[0] = 0;
  32     //sleep(1);
  33     pthread_exit(NULL);
  34 }
  35 // Peterson's Algorithm function for process 1
  36 void *process1(void *arg) {
  37     flag[1] = 1;
  38     turn = 0;
  39     while (flag[0] ==1 && turn == 0) {
  40         // Busy wait
  41     }
  42     // Critical section
  43     critical_section(1);
  44     // Exit section
  45     flag[1] = 0;
  46     //sleep(1);
  47     pthread_exit(NULL);
  48 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
hafsatayyab@DESKTOP-L07V43P:~/OS-Labs/Lab6$ gcc task3.c -o task3 -lpthread
hafsatayyab@DESKTOP-L07V43P:~/OS-Labs/Lab6$ ./task3
Final count: 0
hafsatayyab@DESKTOP-L07V43P:~/OS-Labs/Lab6$
```

Program 4: With Mutex



```
File Edit Selection View ... Lab6 [WSL: Ubuntu-24.04]
EXPLORER
  LAB6 [WSL: UBUNTU-24.04]
    task1
    task2
    task3
    task4
    task1.c
    task2.c
    task3.c
    task4.c
  task4.c
task4.c
  1 #include <stdio.h>
  2 #include <pthread.h>
  3 #include <unistd.h>
  4 #define NUM_ITERATIONS 1000000
  5 int count=10;
  6 pthread_mutex_t mutex; // mutex object
  7
  8 // Critical section function
  9 void critical_section(int process) {
  10     //printf("Process %d is in the critical section\n", process);
  11     //sleep(1); // Simulate some work in the critical section
  12     if(process==0){
  13         for (int i = 0; i < NUM_ITERATIONS; i++)
  14             count--;
  15     }
  16     else if (process==1)
  17     {
  18         for (int i = 0; i < NUM_ITERATIONS; i++)
  19             count++;
  20     }
  21     else if (process==2)
  22     {
  23         for (int i = 0; i < NUM_ITERATIONS; i++)
  24             count--;
  25     }
  26     else{
  27         for (int i = 0; i < NUM_ITERATIONS; i++)
  28             count++;
  29     }
  30 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
hafsatayyab@DESKTOP-L07V43P:~/OS-Labs/Lab6$ gcc task4.c -o task4 -lpthread
hafsatayyab@DESKTOP-L07V43P:~/OS-Labs/Lab6$ ./task4
Final count: 10
hafsatayyab@DESKTOP-L07V43P:~/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)