National Textile University

Department of Computer Science

Subject:

Operating Systems

Submitted To:

Sir Nasir Mehmood

Submitted By:

Alishba Riasat

Registration No:

23-NTU-CS-1135

Lab No:

6

Semester:

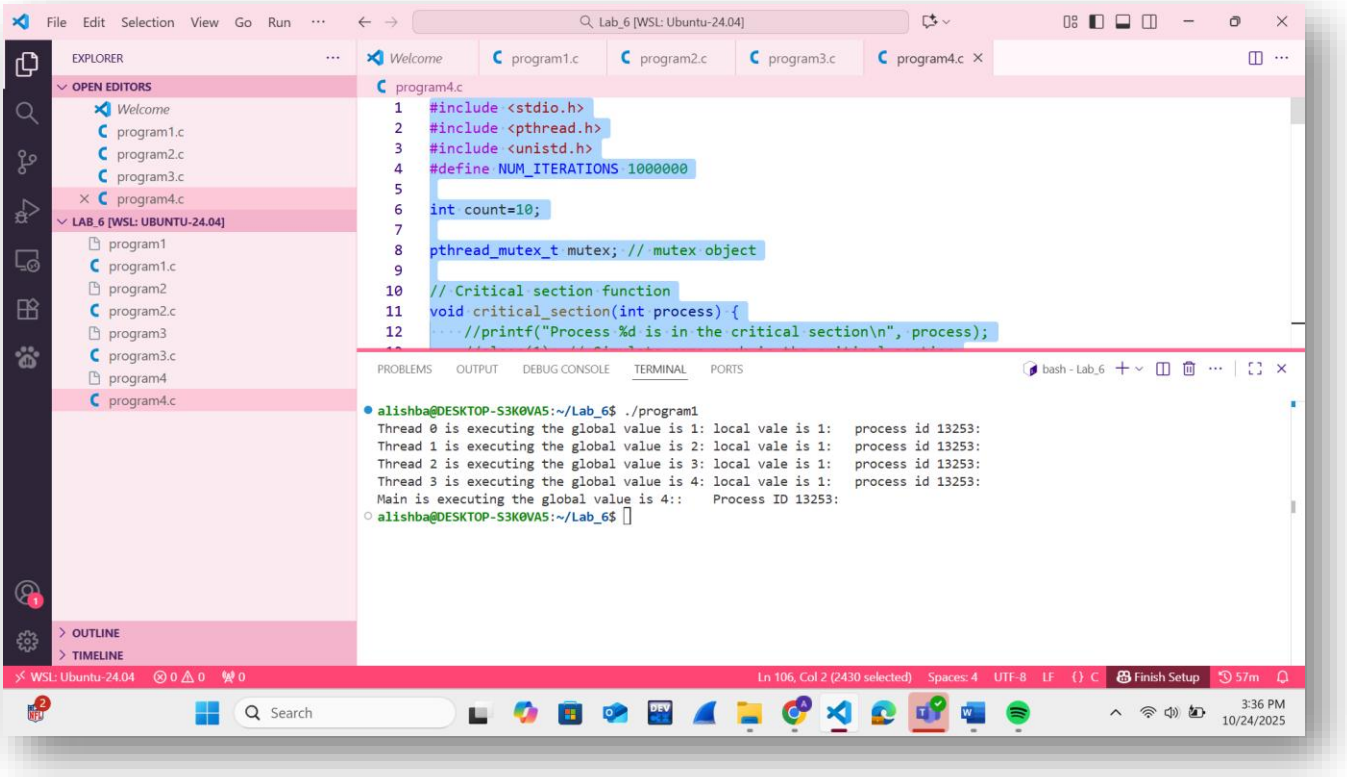
5th

Program 1:

Code:

```
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 4
int varg=0;

void *thread_function(void *arg) {
    int thread_id = *(int *)arg;

    int varl=0;
    varg++;
    varl++;
    printf("Thread %d is executing the global value is %d: local
vale is %d:   process id %d:  \n", thread_id,varg,varl,getpid());
    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_args[NUM_THREADS];


    for (int i = 0; i < NUM_THREADS; ++i) {
        thread_args[i] = i;
        pthread_create(&threads[i], NULL, thread_function,
&thread_args[i]);
    }

    for (int i = 0; i < NUM_THREADS; ++i) {
        pthread_join(threads[i], NULL);
    }
    printf("Main is executing the global value is %d::    Process ID
%d:  \n",varg,getpid());

    return 0;
}
```

Output:



Program 2 (Without Peterson Algorithm):

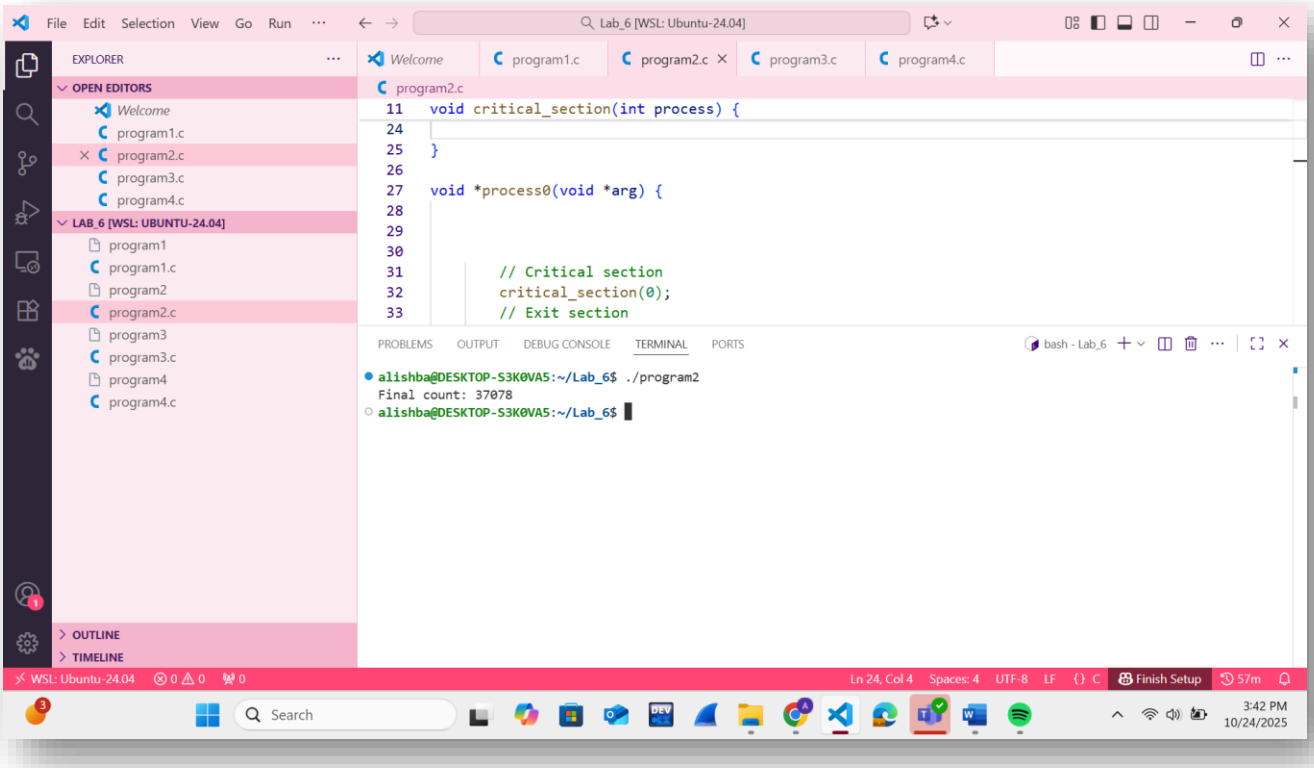
Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

int count=10;



// 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
    {
        for (int i = 0; i < NUM_ITERATIONS; i++)
        count++;
    }

}

void *process0(void *arg) {



        // Critical section
        critical_section(0);
        // Exit section



    return NULL;
}

void *process1(void *arg) {



        // Critical section
        critical_section(1);
        // Exit section




    return NULL;
}

int main() {
    pthread_t thread0, thread1, thread2, thread3;
```

```c
    // Create threads
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process0, NULL);
    pthread_create(&thread3, NULL, process1, NULL);

    // Wait for threads to finish
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);


    printf("Final count: %d\n", count);

    return 0;
}
```

Output:



Program 3 (With Peterson Alogorithm):

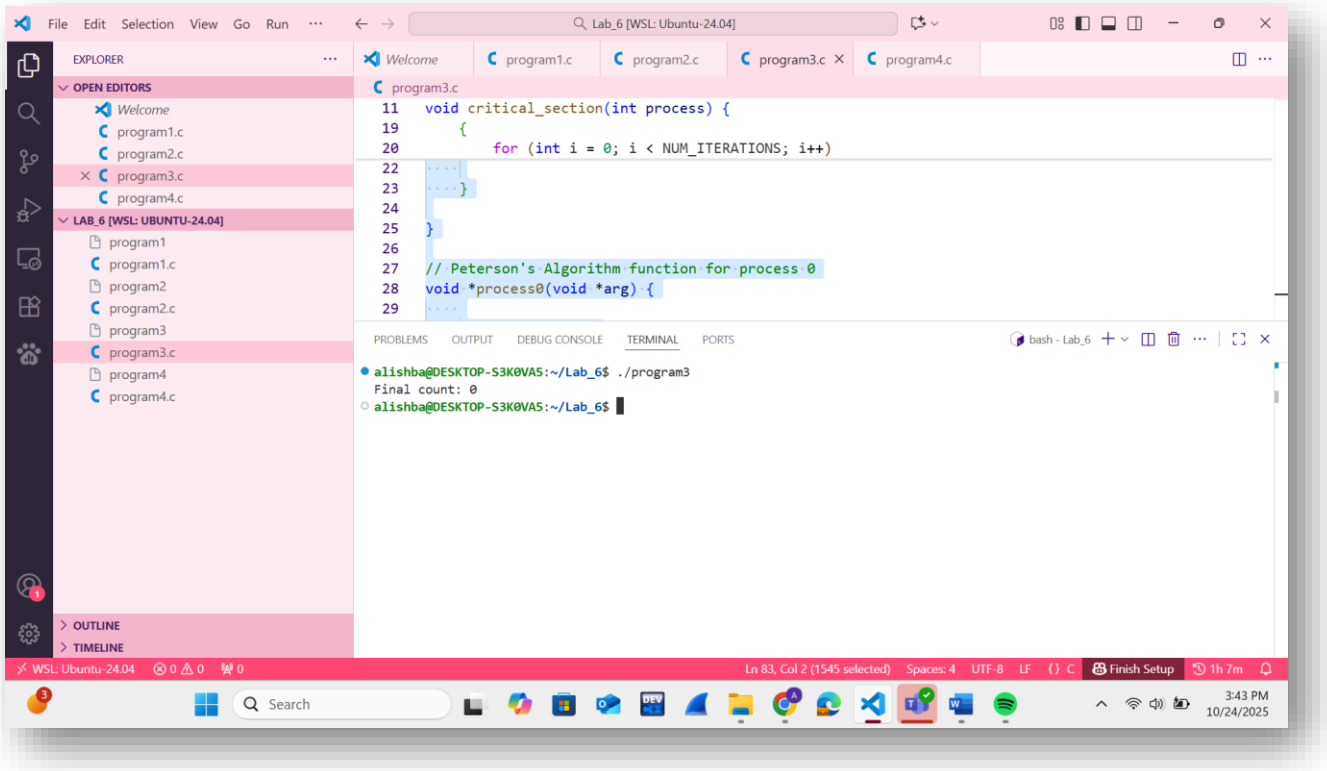
Code:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 100000
// Shared variables
int turn;
int flag[2];
int count=0;

// Critical section function
void critical_section(int process) {

    if(process==0){

        for (int i = 0; i < NUM_ITERATIONS; i++)
            count--;
    }
    else
    {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count++;

    }

}

// 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);
}

int main() {
    pthread_t thread0, thread1;

    // Initialize shared variables
    flag[0] = 0;
    flag[1] = 0;
    turn = 0;


    // Create threads
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);

    // Wait for threads to finish
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);
```

```
    printf("Final count: %d\n", count);

    return 0;
}
```

Output:



Program 4 (With Mutex):

Code:

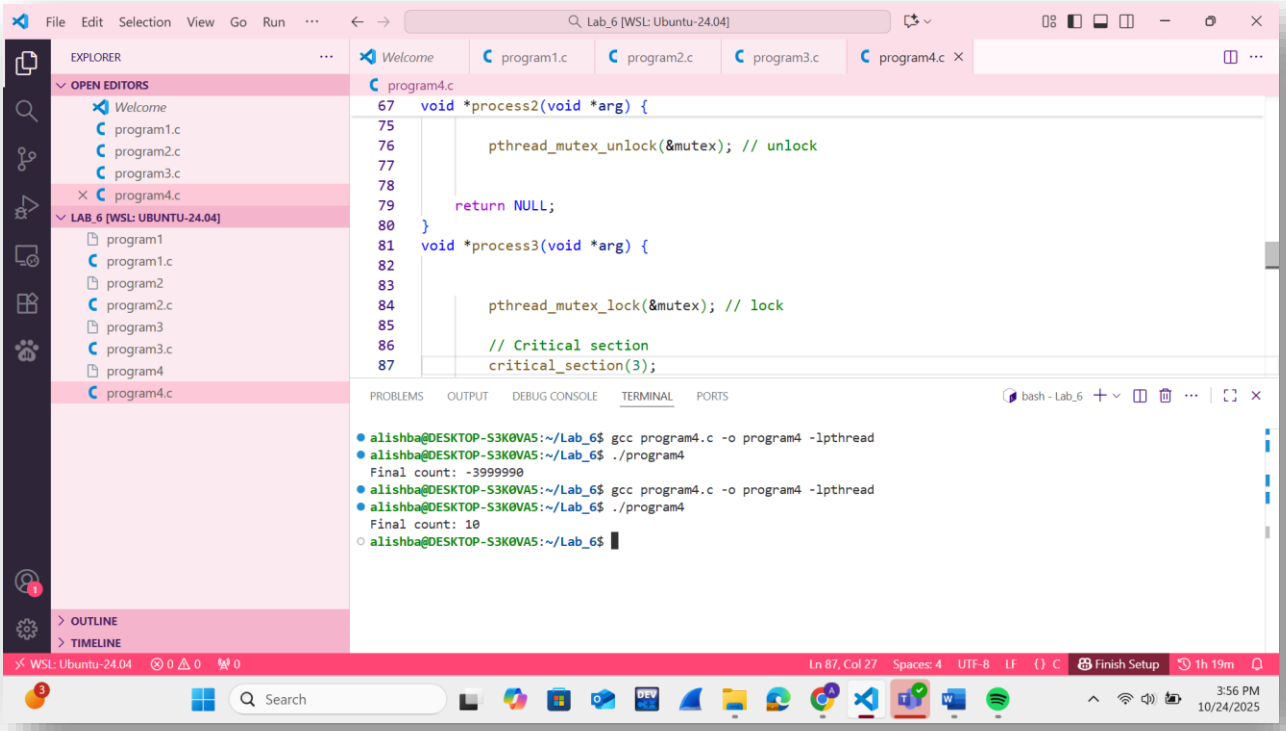
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

int count=10;

pthread_mutex_t mutex; // mutex object
```

```c
// 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++;
    }
    //printf("Process %d has updated count to %d\n", process,
count);
    //printf("Process %d is leaving the critical section\n",
process);
}

// Peterson's Algorithm function for process 0
void *process0(void *arg) {

        pthread_mutex_lock(&mutex); // lock

        // Critical section
        critical_section(0);
        // Exit section

        pthread_mutex_unlock(&mutex); // unlock

    return NULL;
}
```

```c
// Peterson's Algorithm function for process 1
void *process1(void *arg) {


        pthread_mutex_lock(&mutex); // lock

        // Critical section
        critical_section(1);
        // Exit section

        pthread_mutex_unlock(&mutex); // unlock


    return NULL;
}

void *process2(void *arg) {


        pthread_mutex_lock(&mutex); // lock

        // Critical section
        critical_section(2);
        // Exit section

        pthread_mutex_unlock(&mutex); // unlock


    return NULL;
}
void *process3(void *arg) {


        pthread_mutex_lock(&mutex); // lock

        // Critical section
        critical_section(3);
        // Exit section

        pthread_mutex_unlock(&mutex); // unlock
```

```c
    return NULL;
}
int main() {
    pthread_t thread0, thread1, thread2, thread3, thread4, thread5,
thread6, thread7;

    pthread_mutex_init(&mutex,NULL); // initialize mutex

    // Create threads
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process2, NULL);
    pthread_create(&thread3, NULL, process3, NULL);
    pthread_create(&thread4, NULL, process0, NULL);
    pthread_create(&thread5, NULL, process1, NULL);
    pthread_create(&thread6, NULL, process2, NULL);
    pthread_create(&thread7, NULL, process3, NULL);



    // Wait for threads to finish
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);
    pthread_join(thread5, NULL);
    pthread_join(thread6, NULL);
    pthread_join(thread7, NULL);

    pthread_mutex_destroy(&mutex); // destroy mutex

    printf("Final count: %d\n", count);

    return 0;
}
```

Output:



Comparison Between Peterson Alogrithm and one with Mutex:

Feature	Peterson's Algorithm	Mutex Version
Critical Section Protection	Software-only: flag[] + turn	OS-level: pthread_mutex_lock/unlock
Number of Threads Supported	2 only	Any number (e.g., 8 threads)
Waiting Mechanism	Busy wait: while(flag[1]==1 && turn==1)	Blocking: pthread_mutex_lock() suspends the thread
Code Complexity	Requires flag[], turn, and busy wait logic	Simple: lock → critical section → unlock
Scalability	Hard to extend beyond 2 threads	Easy to extend to many threads/processes
CPU Usage While Waiting	High (spins continuously)	Low (thread sleeps until lock is free)
Ease of Maintenance	Harder to read and extend	Easy to read, maintain, and reuse
Example Thread Creation	Only 2 threads (thread0, thread1)	Multiple threads (thread0–thread7)

Use Case	Educational, software mutual exclusion demo	Practical, real-world multithreading
Final Count Control	Works for 2 threads if increments/decrements are balanced	Works for any number of threads if mutex protects critical section