National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Akasha Fatima

Reg. number:

23-NTU-CS-FL-1132

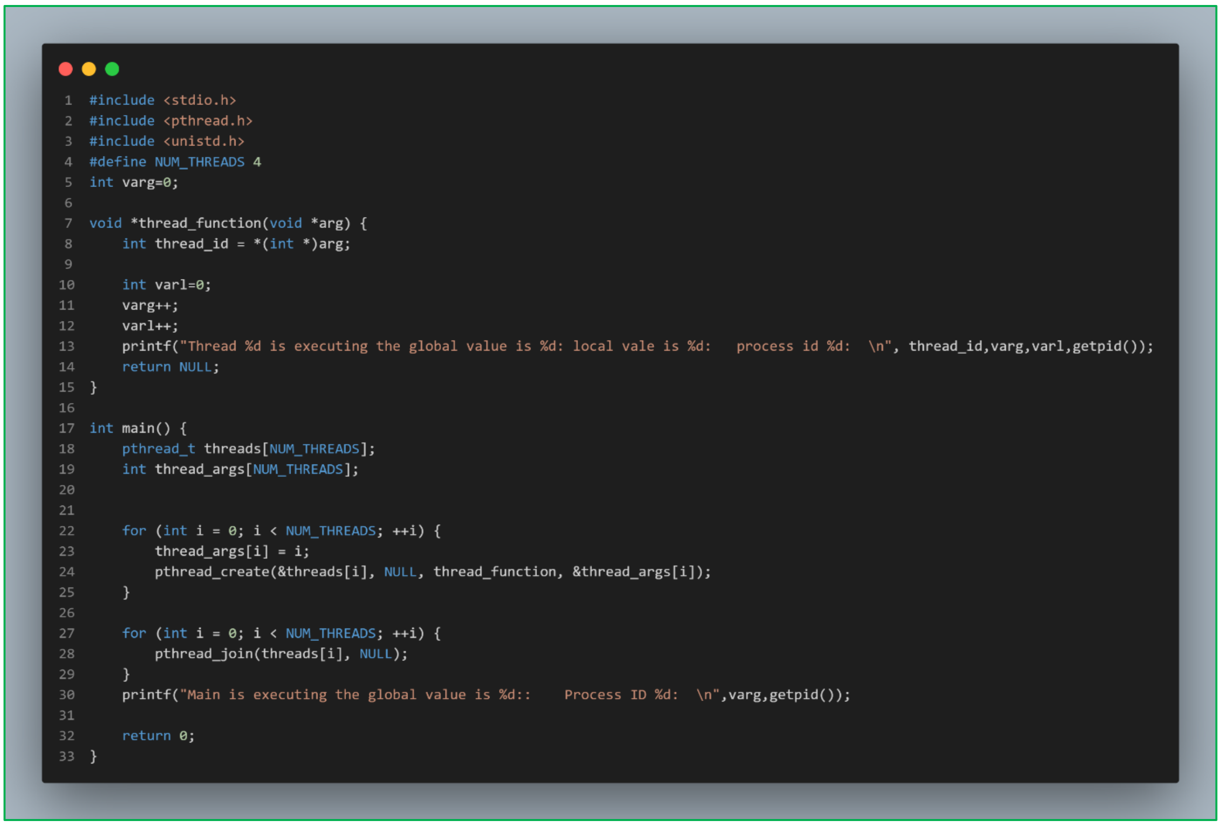
Semester:

5th - A

LAB-06

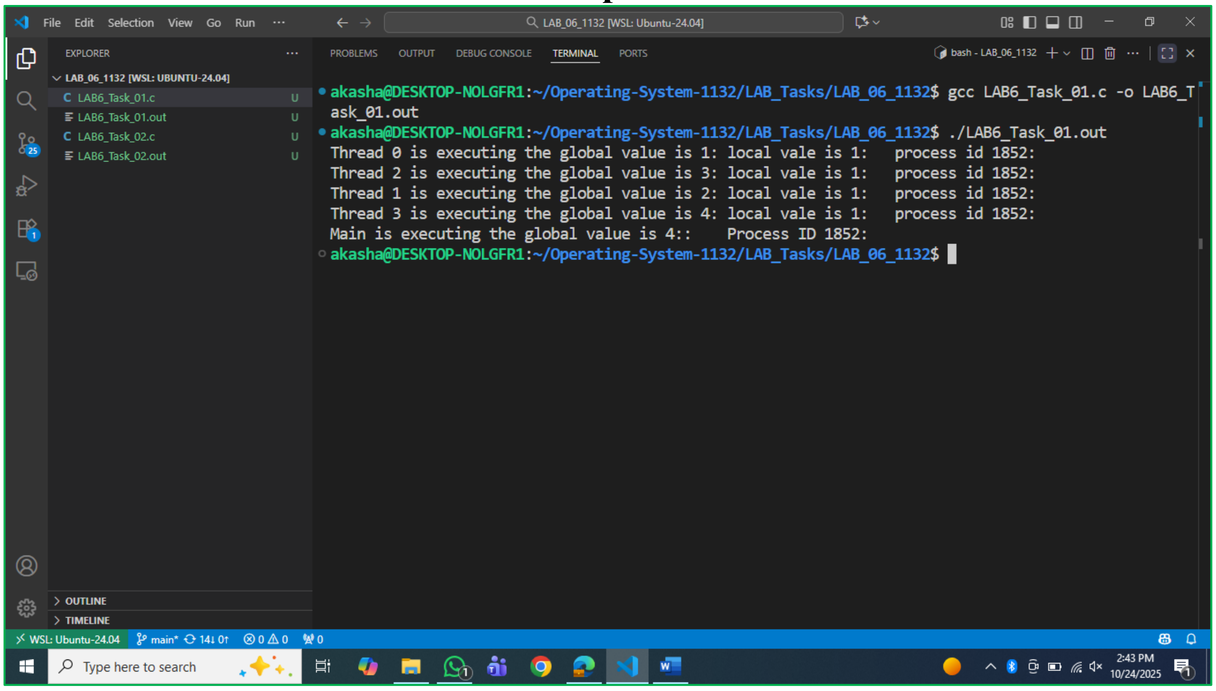
Task_01: Simple threads Execution:

CODE:



```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.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;
}
```

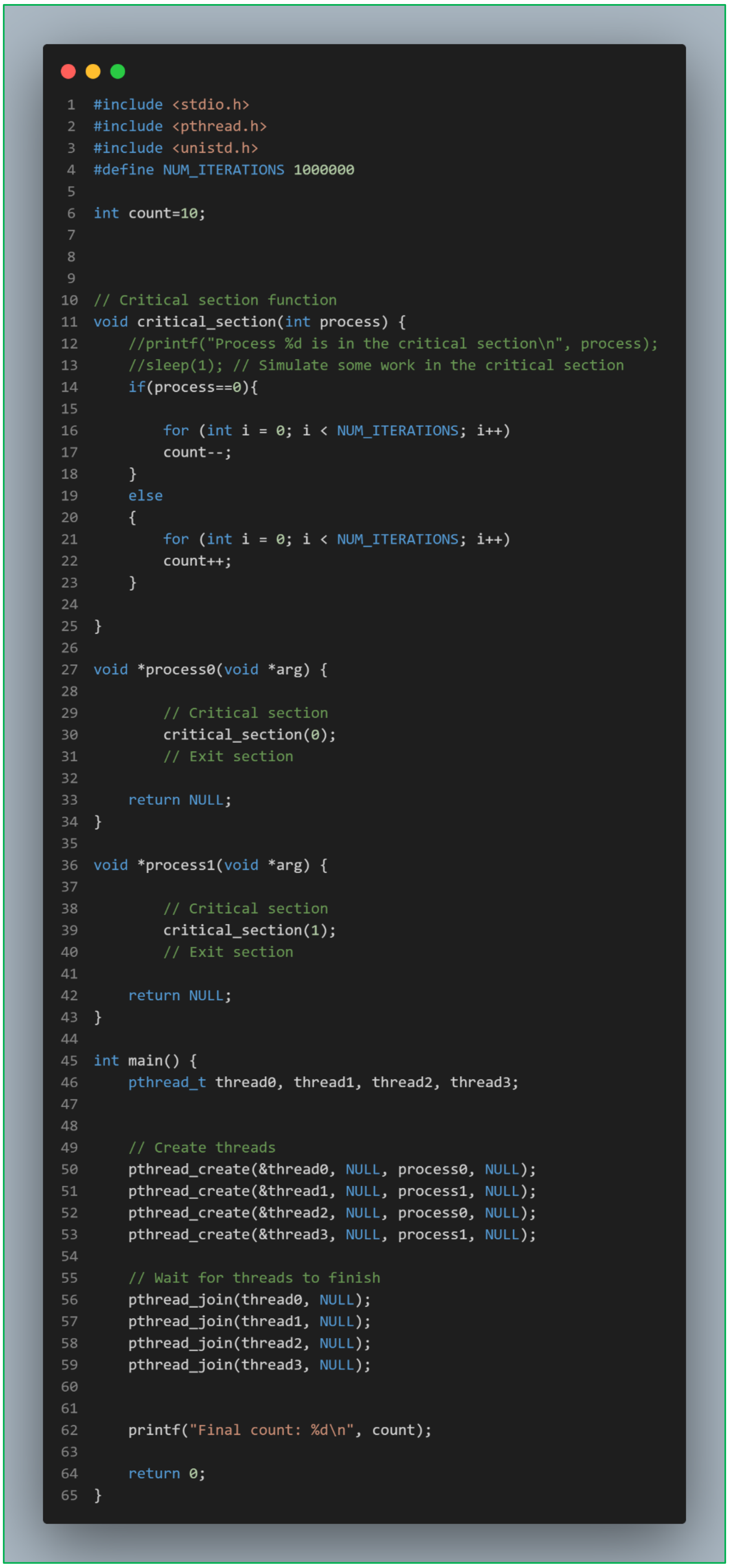
Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_01.c -o LAB6_T
ask_01.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_01.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 1852:
Thread 2 is executing the global value is 3: local vale is 1:   process id 1852:
Thread 1 is executing the global value is 2: local vale is 1:   process id 1852:
Thread 3 is executing the global value is 4: local vale is 1:   process id 1852:
Main is executing the global value is 4::    Process ID 1852:
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

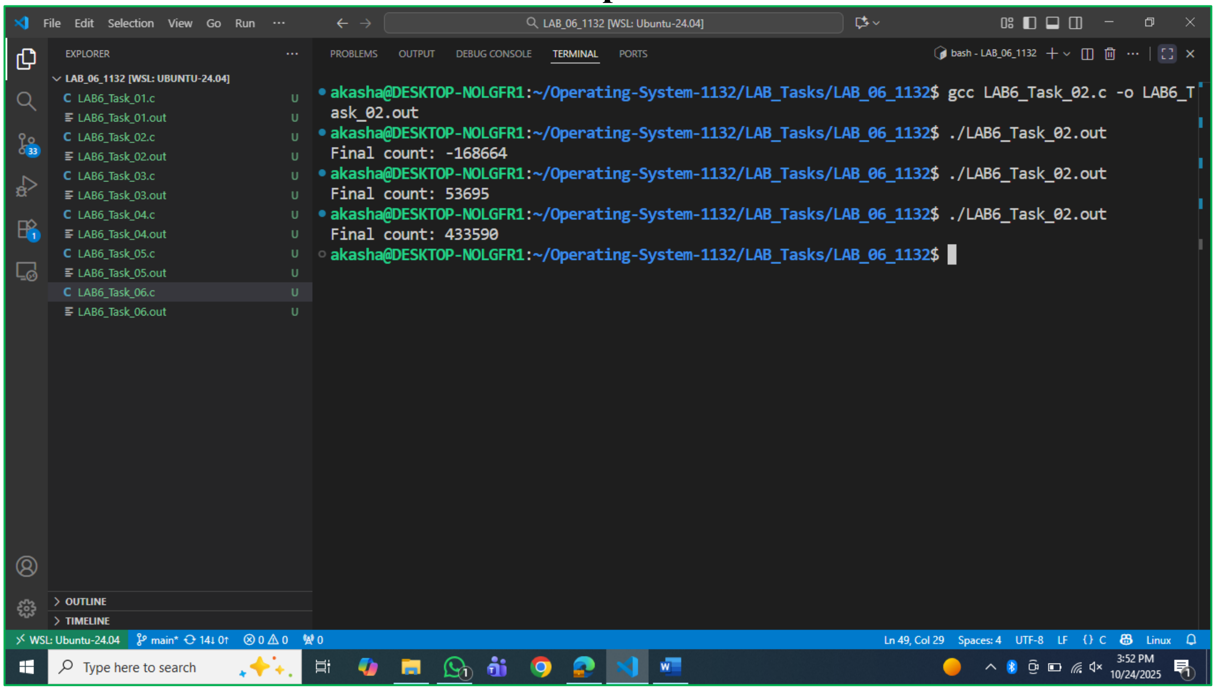
Task_02: Synchronization Problem

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;


    // 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;
}
```

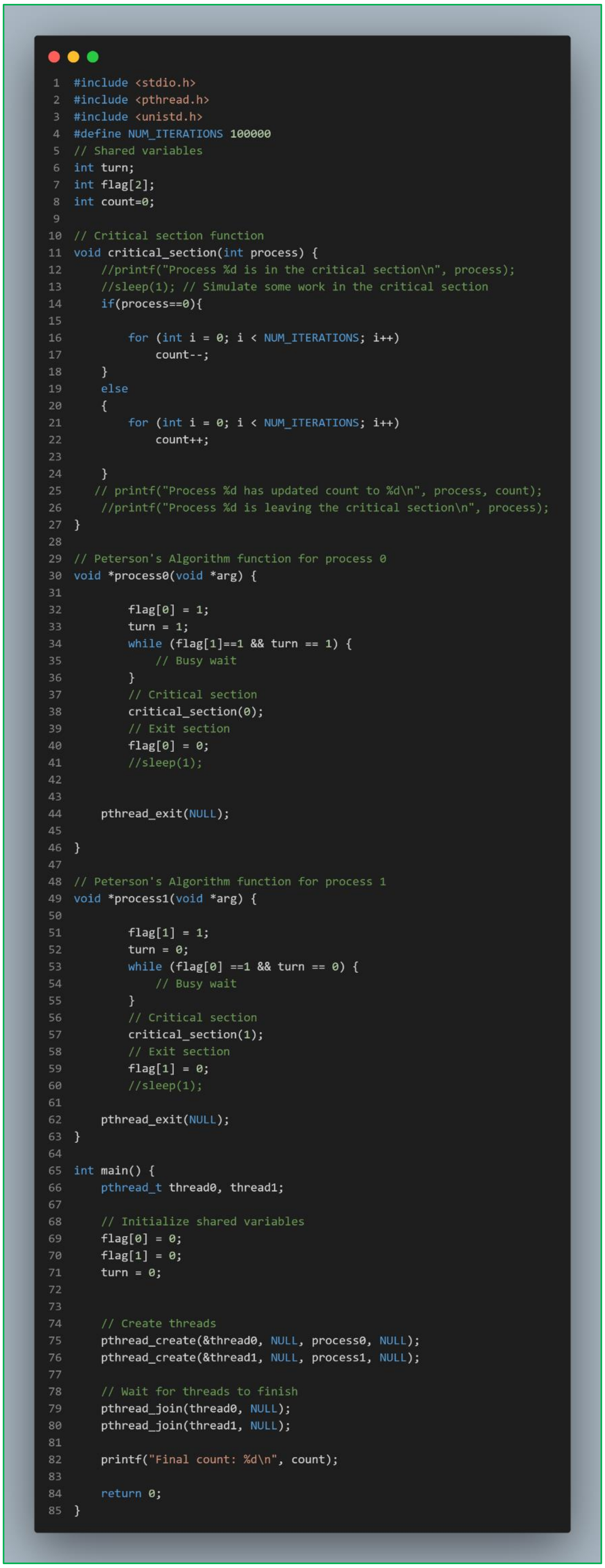
Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_02.c -o LAB6_Task_02.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_02.out
Final count: -168664
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_02.out
Final count: 53695
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_02.out
Final count: 433590
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

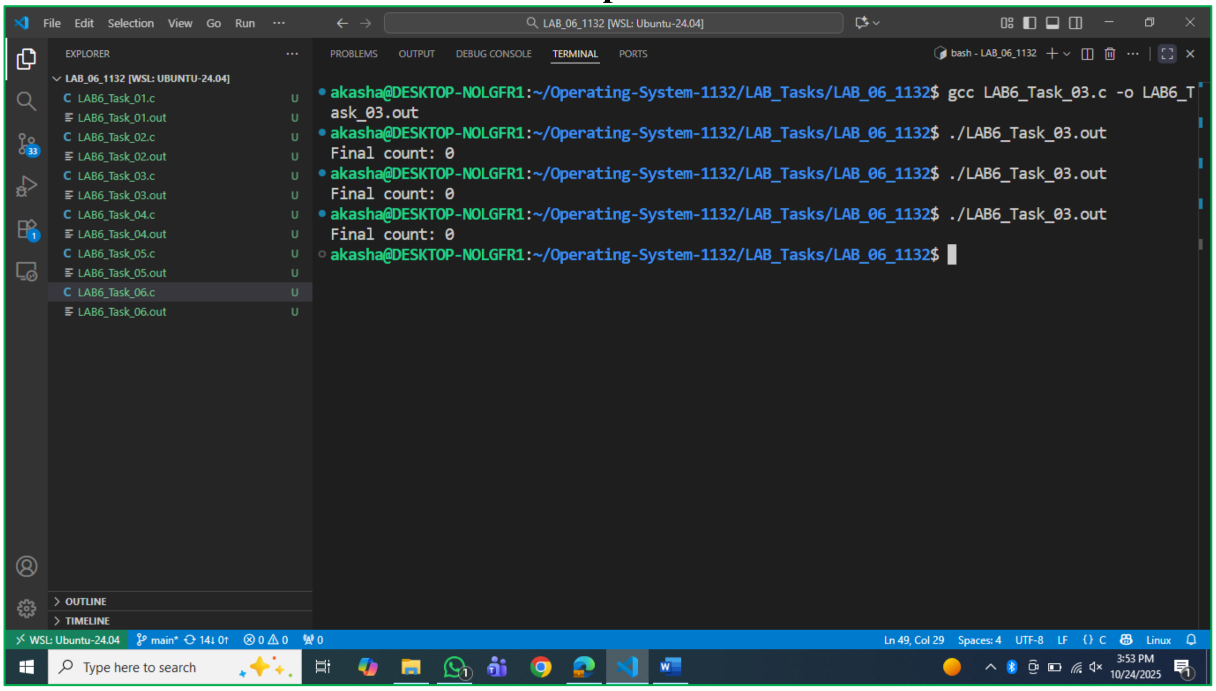
Task_03: Peterson Solution for Synchronization Issue of Task 02:

CODE:



```c
#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) {
    //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++;

    }
   // 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) {

        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;
}
```

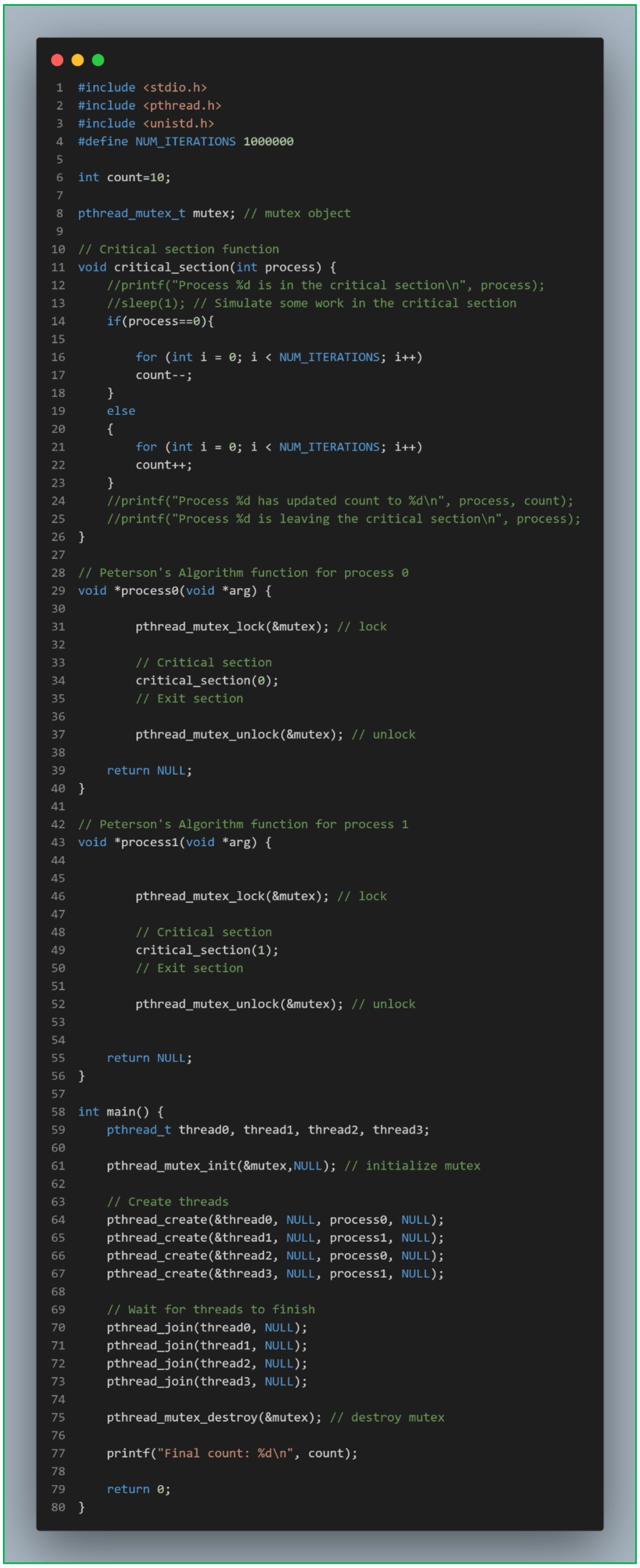
Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_03.c -o LAB6_Task_03.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_03.out
Final count: 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_03.out
Final count: 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_03.out
Final count: 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

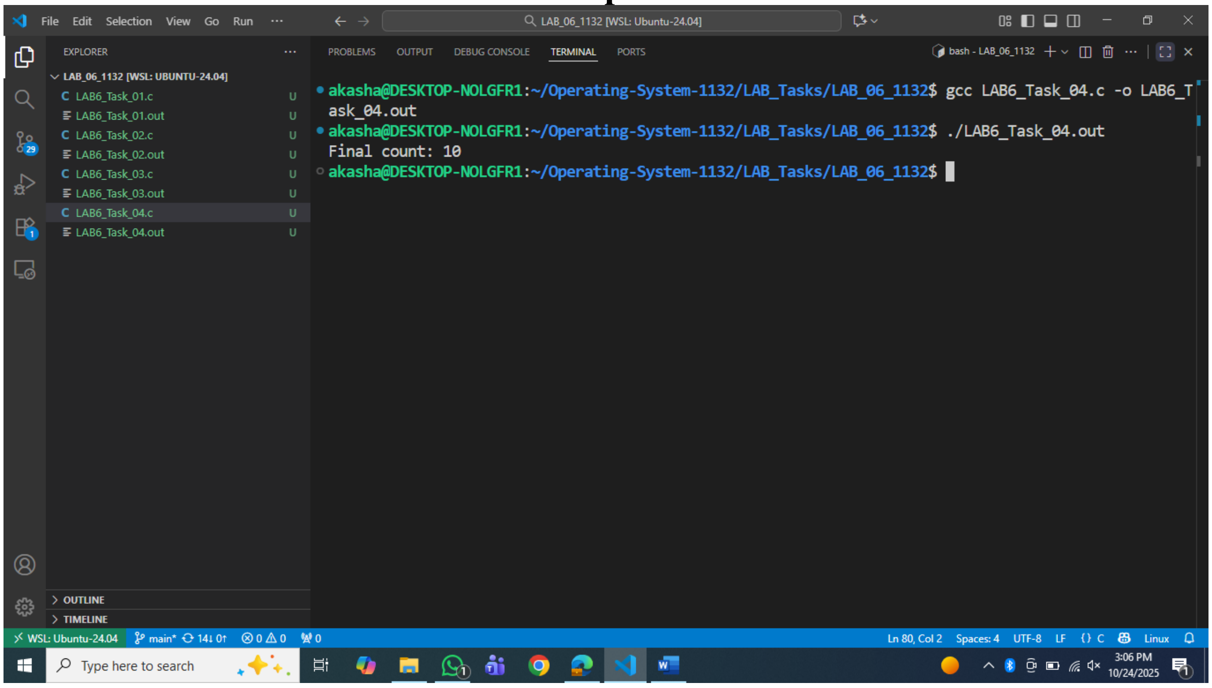
Task_04: Mutex Solution for Synchronization Issue of Task 02 with two processes only:

CODE:



```c
#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
    {
        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;
}

// 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;
}

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

    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, 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);

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```

Output:



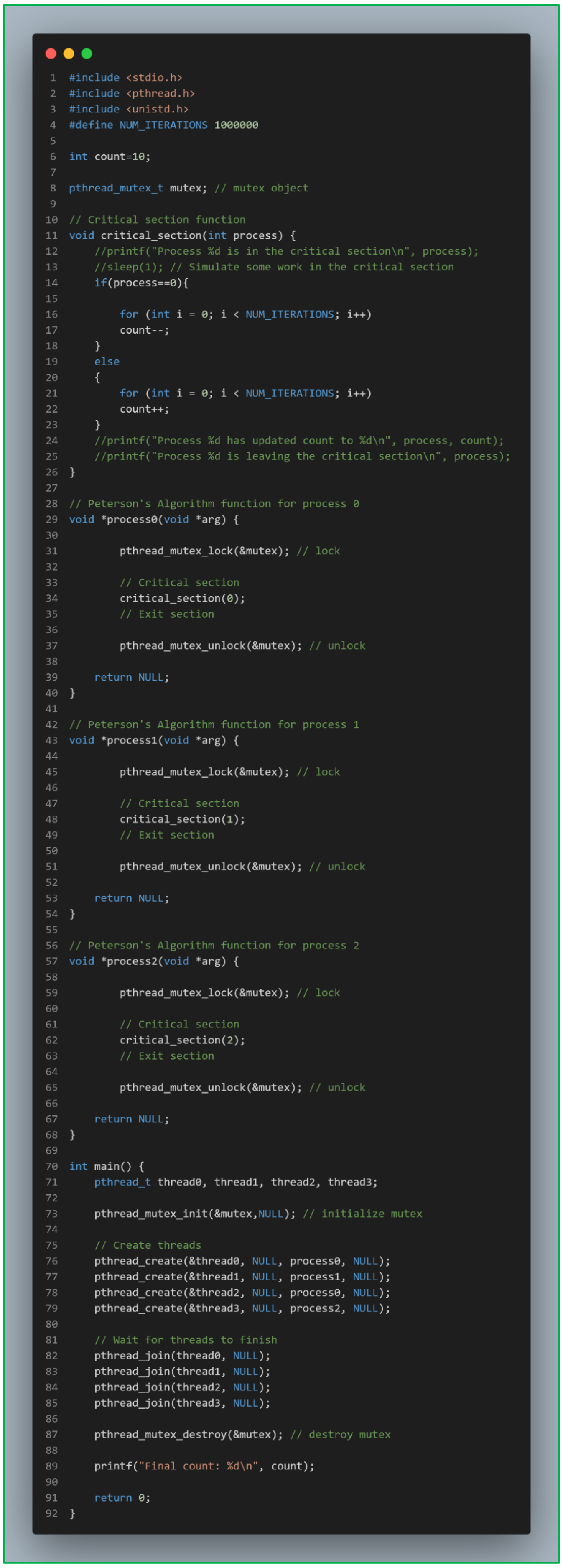
```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_04.c -o LAB6_T
ask_04.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_04.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

Difference between Mutex and Peterson Methods:

Peterson	Mutex
It uses the while loops.	It uses the build-in **Lock** and **Unlock** commands
The code is totally written by the user at the program written time.	The code is the build in for the Lock and Unlock commands.
It is used for only two processes.	It can be used for any numbers of the process

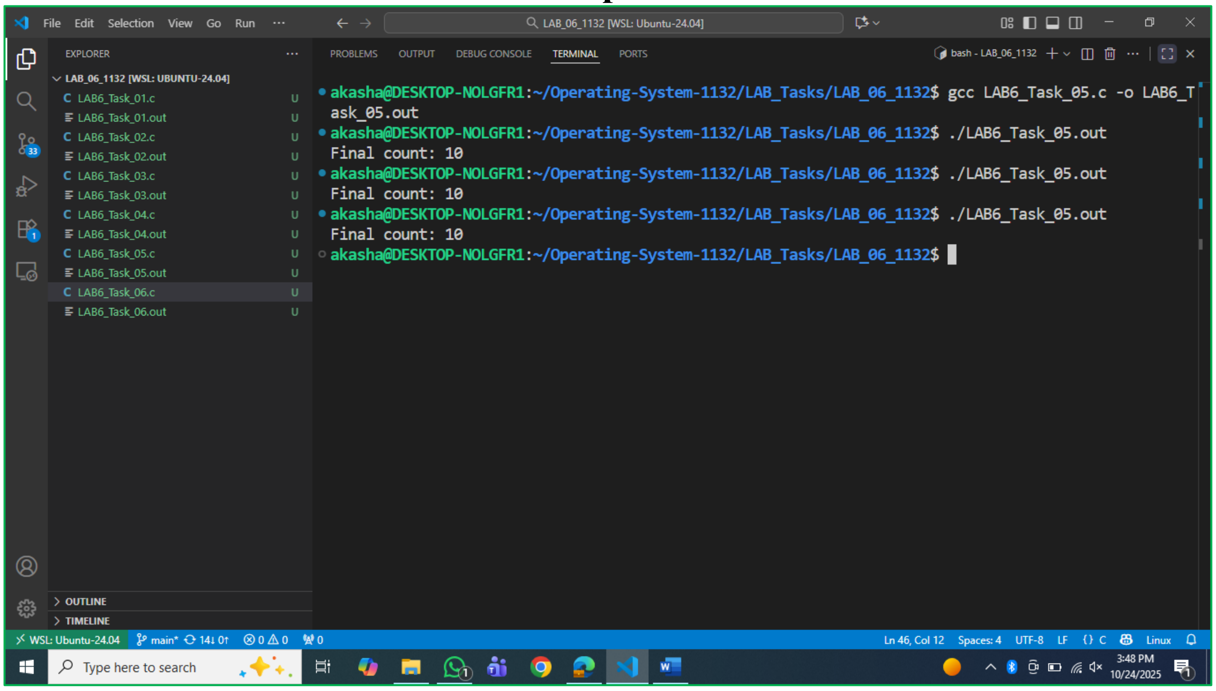
Task_05: Mutex Solution for Synchronization Issue of Task 02 with three processes:

CODE:



```c
#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
    {
        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;
}

// 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;
}

// Peterson's Algorithm function for process 2
void *process2(void *arg) {

        pthread_mutex_lock(&mutex); // lock

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

        pthread_mutex_unlock(&mutex); // unlock

    return NULL;
}

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

    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, process0, NULL);
    pthread_create(&thread3, NULL, process2, NULL);

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

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```

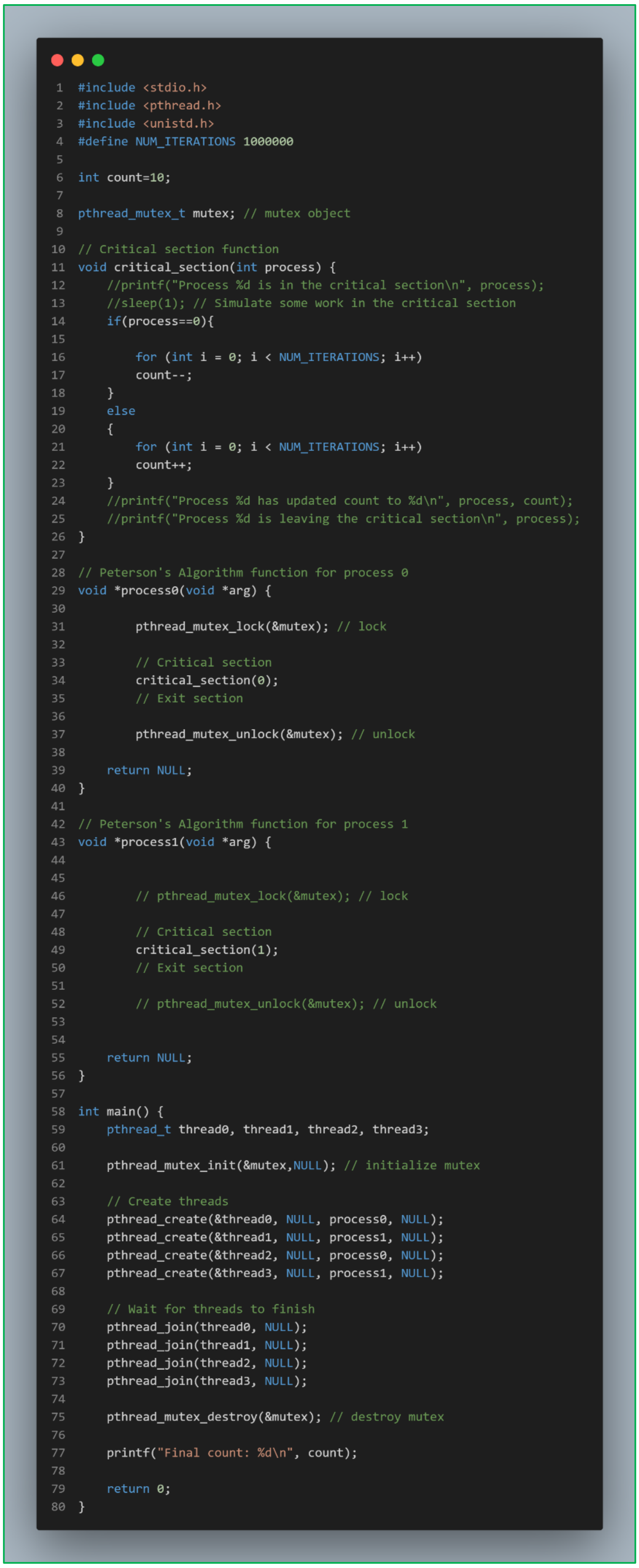
Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_05.c -o LAB6_Task_05.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_05.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_05.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_05.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

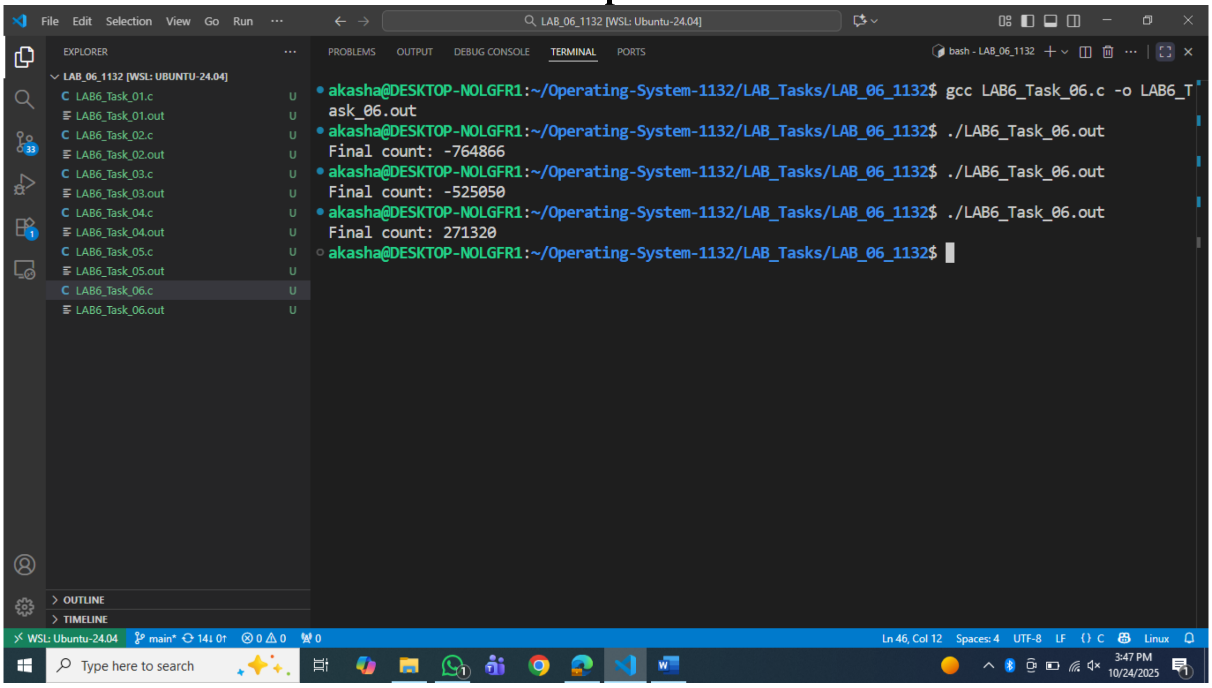
Task_06: Commenting the “Lock” and “Unlock” commands:

CODE:



```c
#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
    {
        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;
}

// 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;
}

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

    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, 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);

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_06.c -o LAB6_T
ask_06.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_06.out
Final count: -764866
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_06.out
Final count: -525050
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_06.out
Final count: 271320
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```