National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Nasir Mehmood

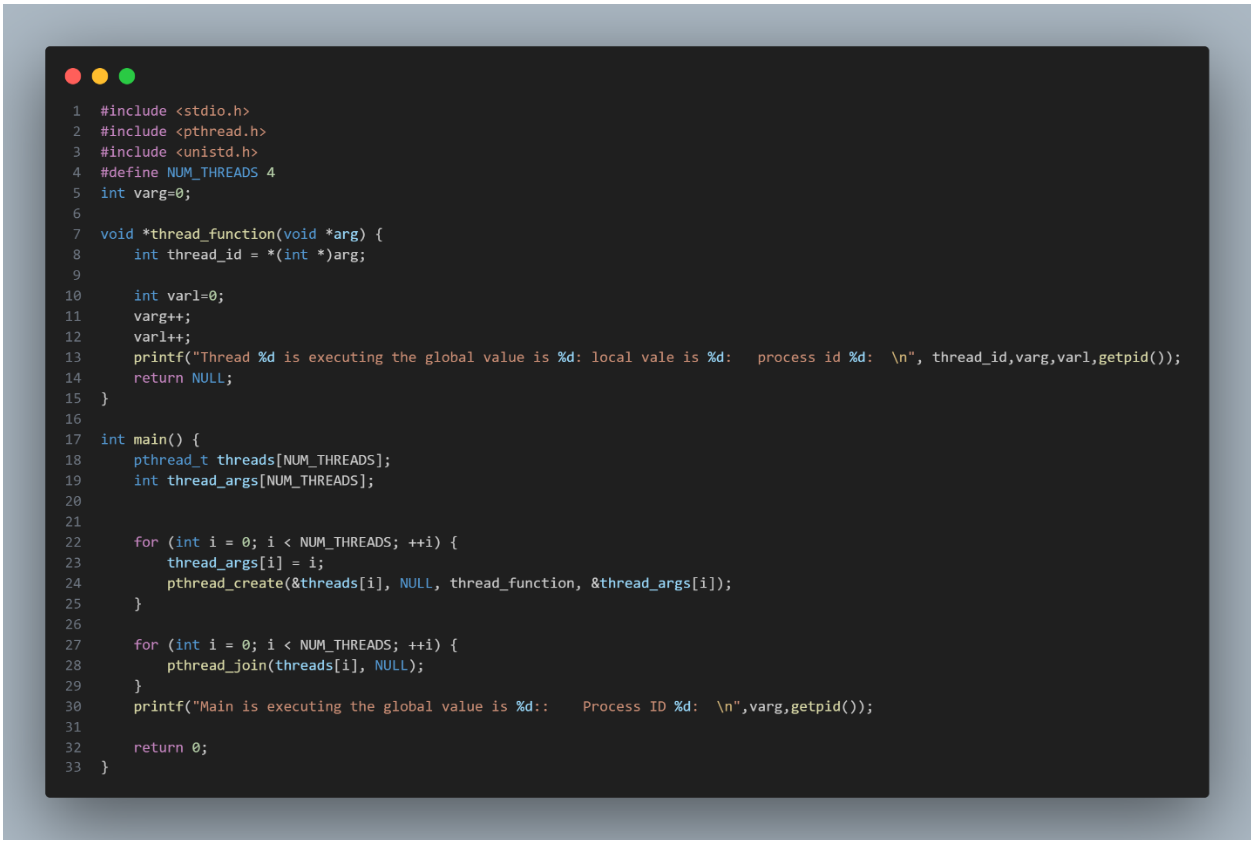
Submitted by: Nimra Tanveer

Reg number:23-NTU-CS-1201

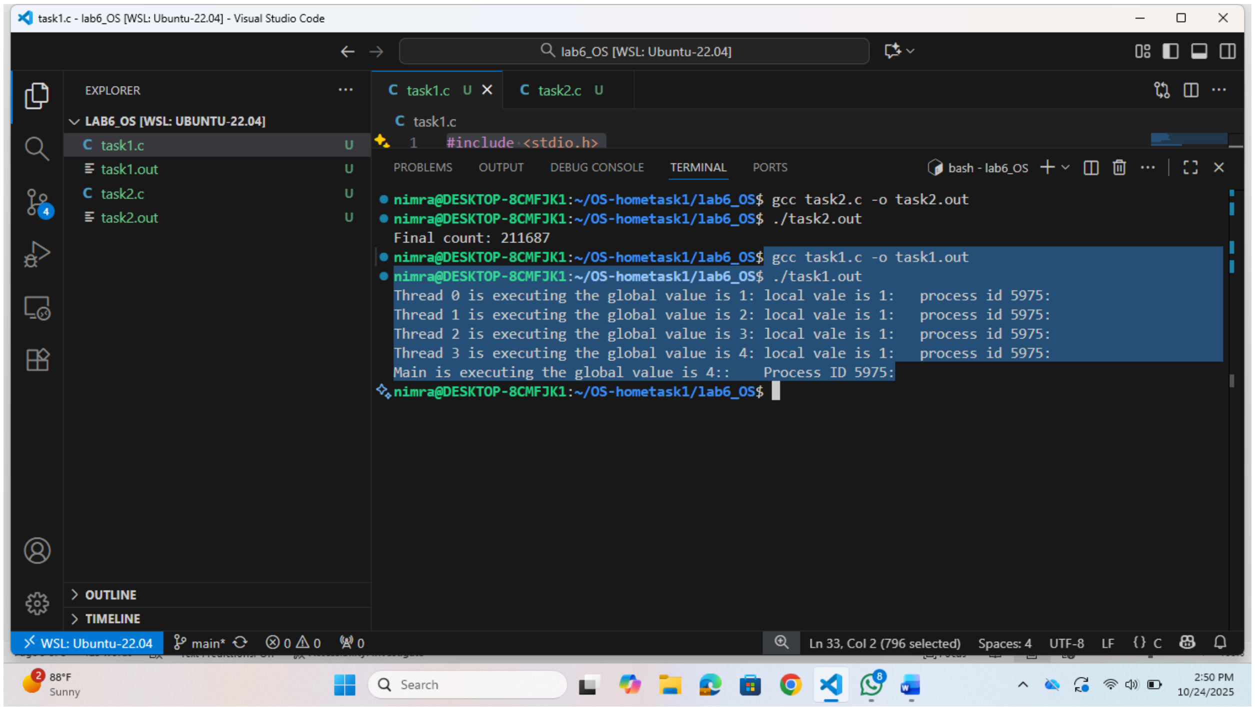
Semester:5th

Lab manual 06

Task 1:



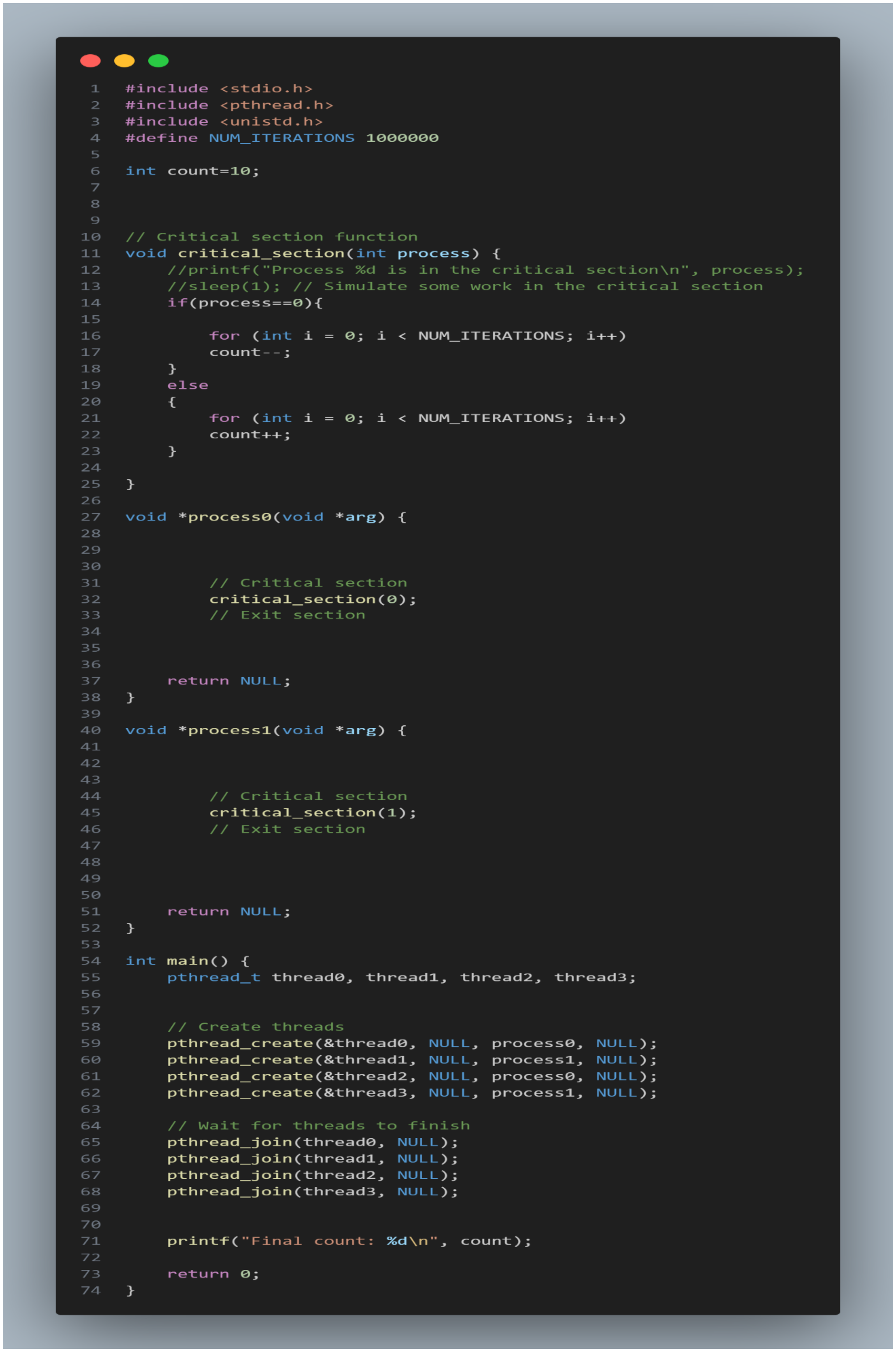
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_THREADS 4
int varg=0;

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;
}
```



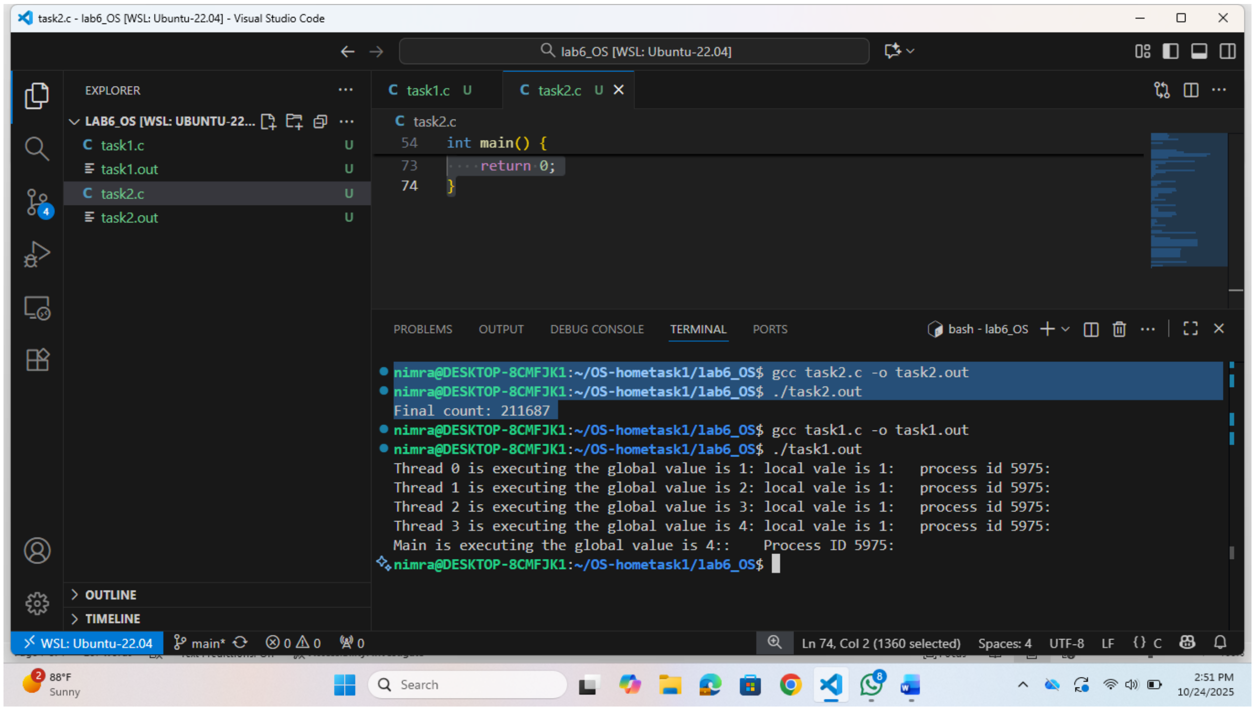
```
#include <stdio.h>
```

```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task2.c -o task2.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task2.out
Final count: 211687
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task1.c -o task1.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task1.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 5975:
Thread 1 is executing the global value is 2: local vale is 1:   process id 5975:
Thread 2 is executing the global value is 3: local vale is 1:   process id 5975:
Thread 3 is executing the global value is 4: local vale is 1:   process id 5975:
Main is executing the global value is 4::    Process ID 5975:
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ 
```

Task 02:



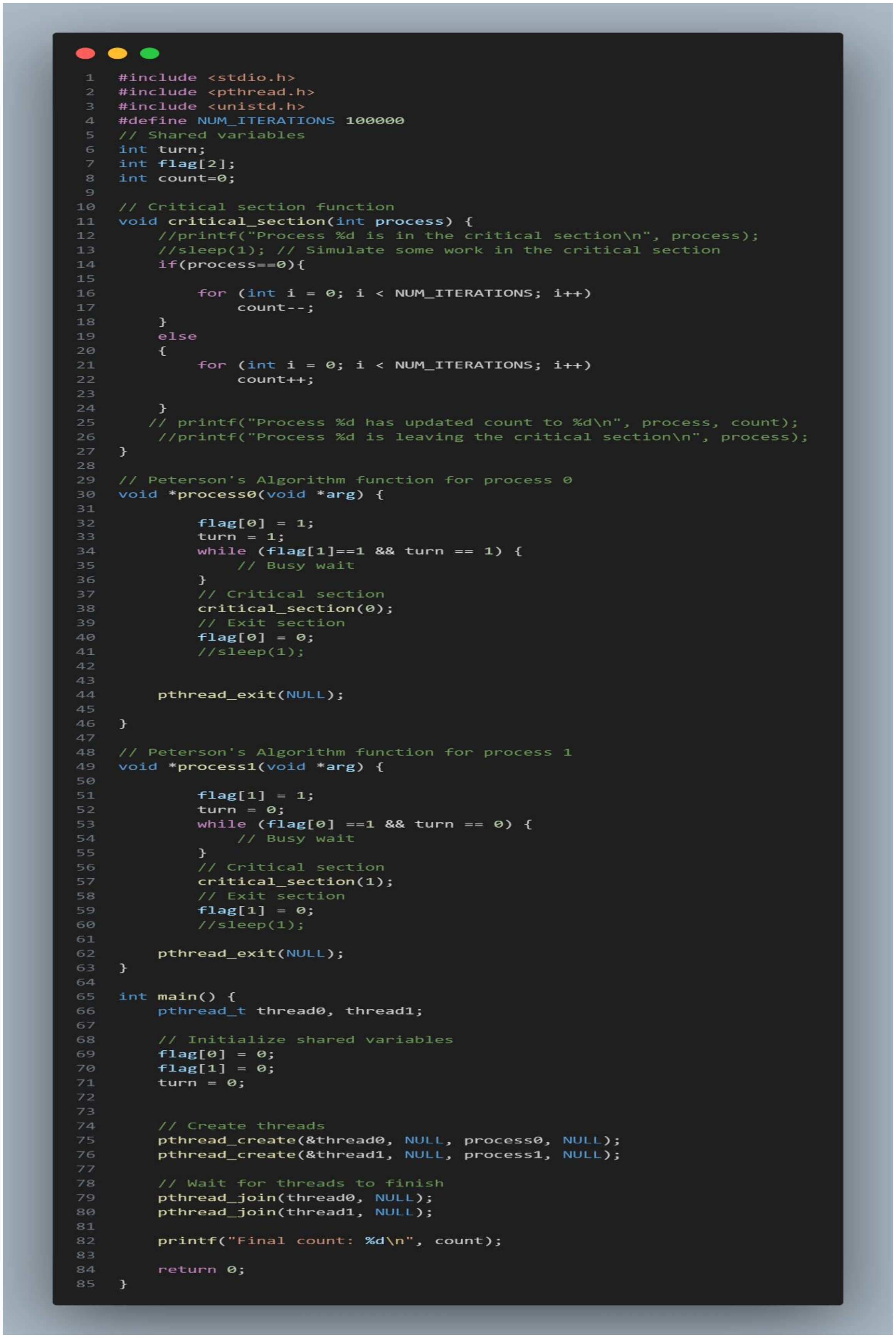
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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;
}
```



```c
int main() {
    return 0;
}
```

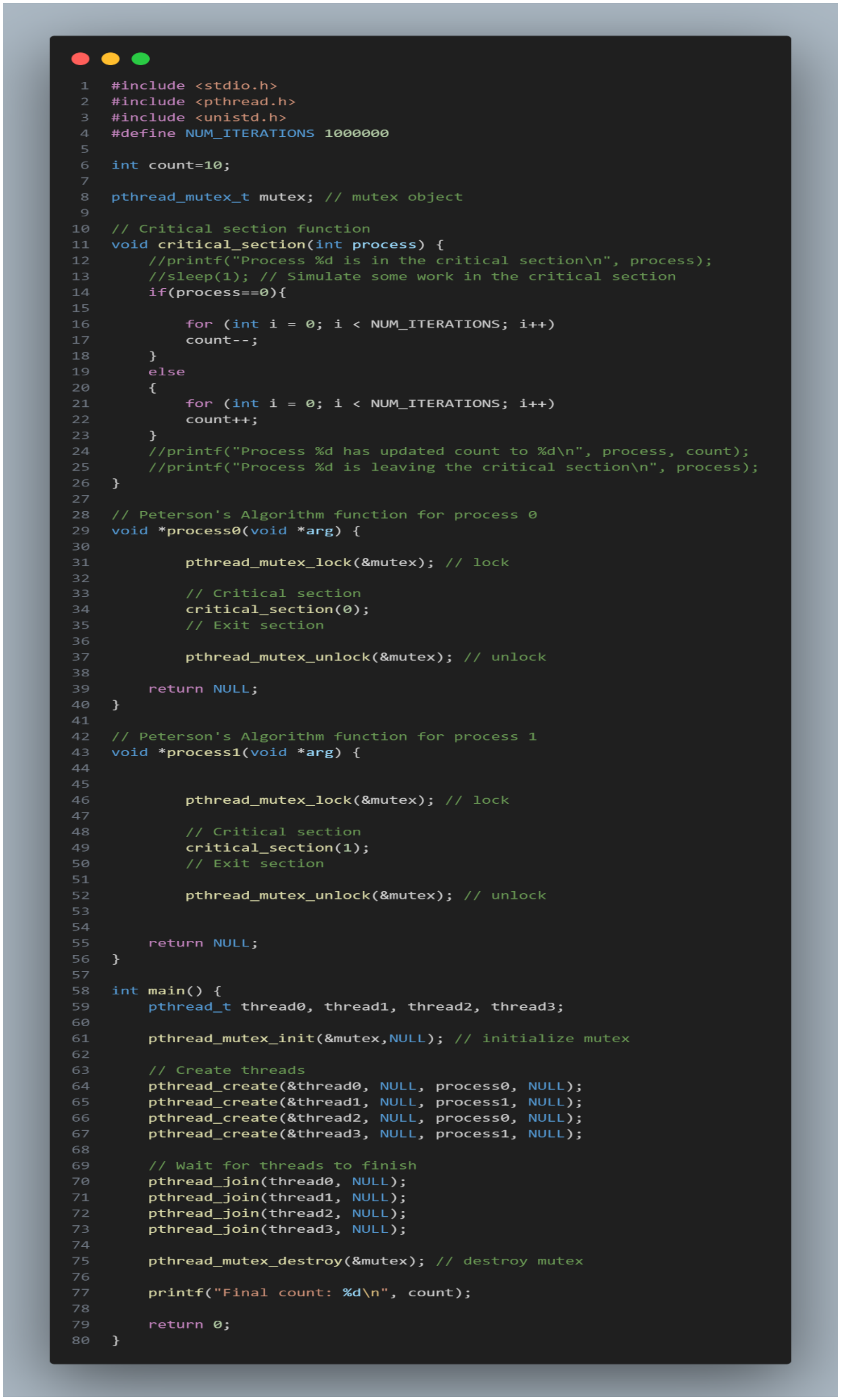
```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task2.c -o task2.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task2.out
Final count: 211687
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task1.c -o task1.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task1.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 5975:
Thread 1 is executing the global value is 2: local vale is 1:   process id 5975:
Thread 2 is executing the global value is 3: local vale is 1:   process id 5975:
Thread 3 is executing the global value is 4: local vale is 1:   process id 5975:
Main is executing the global value is 4::    Process ID 5975:
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
```

Task 03:

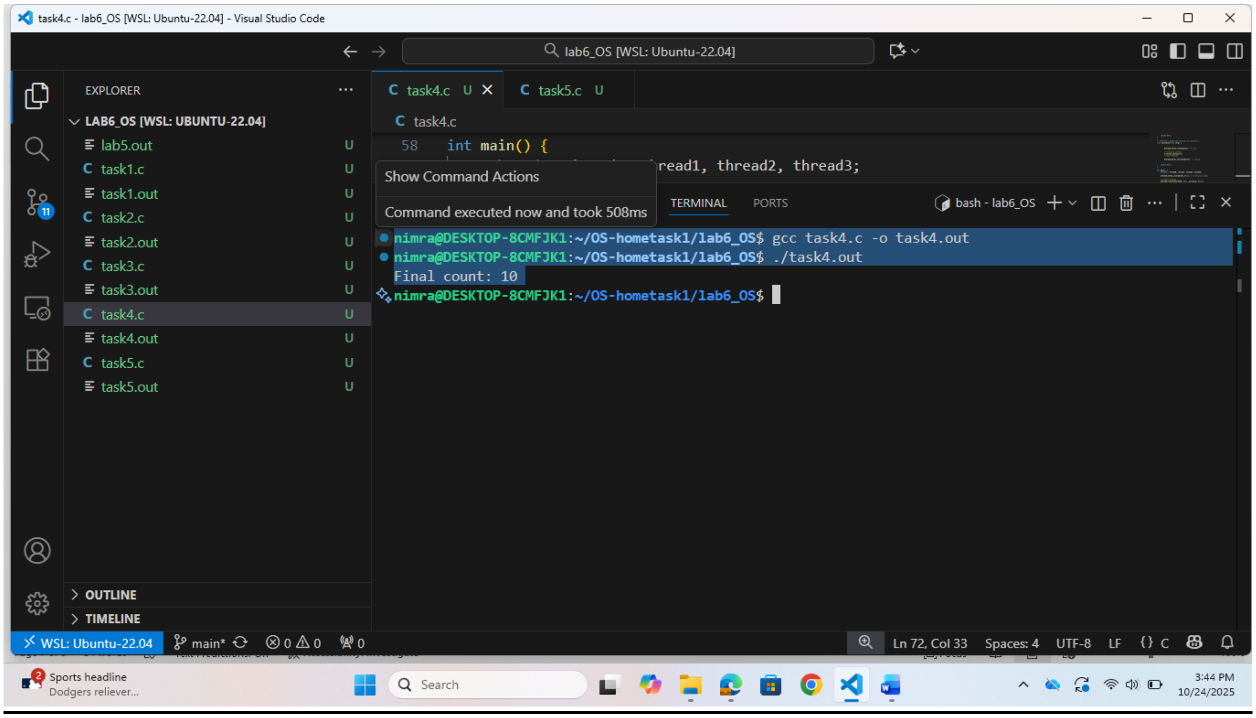


```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 100000
// Shared variables
int turn;
int flag[2];
int count=0;

// 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;
}
```

Task 04:



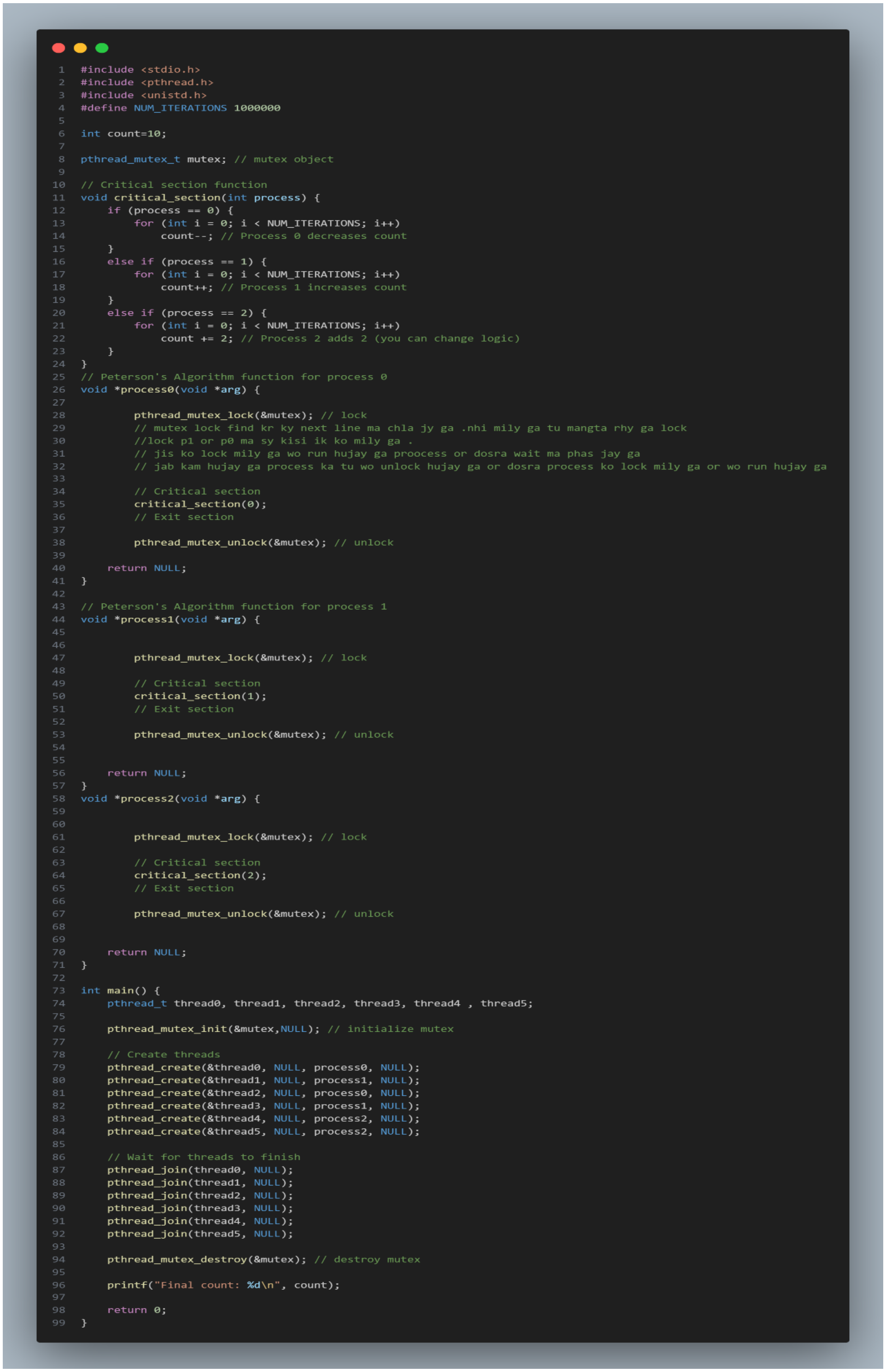
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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;
}
```



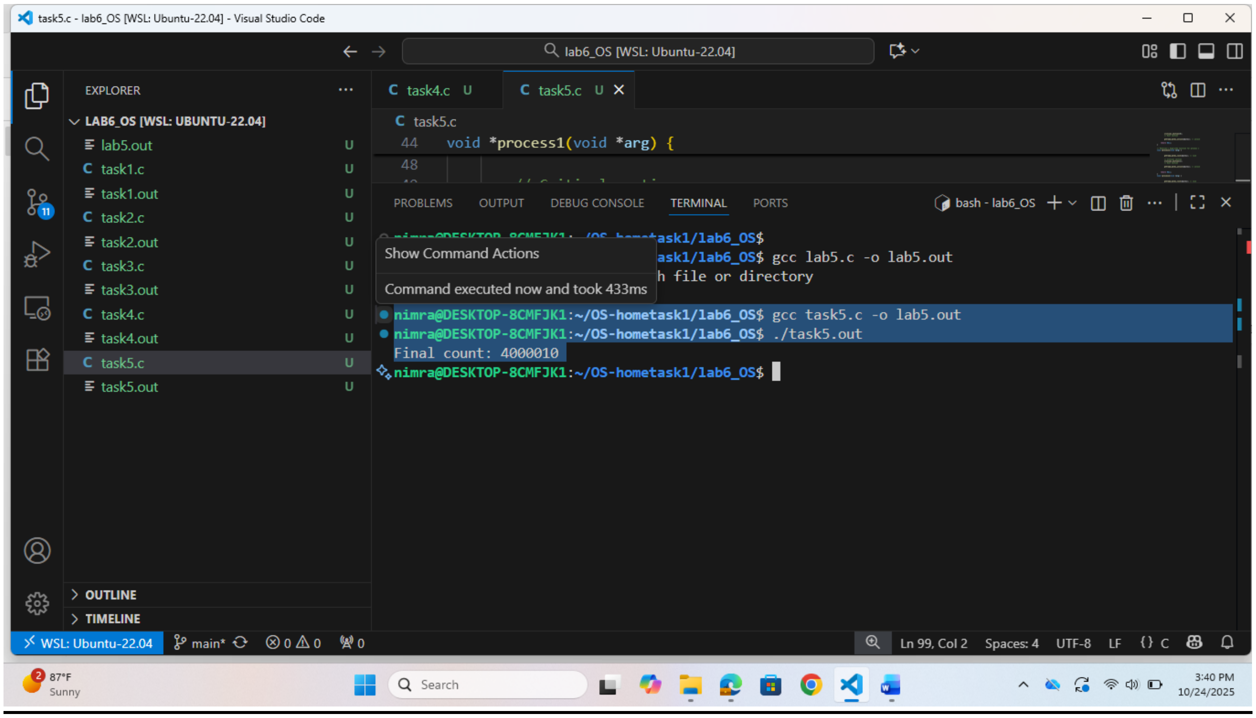
```
58    int main() {
                                  read1, thread2, thread3;
```

```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task4.c -o task4.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task4.out
Final count: 10
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
```

Task 05:



```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) {
    if (process == 0) {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count--; // Process 0 decreases count
    }
    else if (process == 1) {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count++; // Process 1 increases count
    }
    else if (process == 2) {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count += 2; // Process 2 adds 2 (you can change logic)
    }
}
// Peterson's Algorithm function for process 0
void *process0(void *arg) {

        pthread_mutex_lock(&mutex); // lock
        // mutex lock find kr ky next line ma chla jy ga .nhi mily ga tu mangta rhy ga lock
        //lock p1 or p0 ma sy kisi ik ko mily ga .
        // jis ko lock mily ga wo run hujay ga proocess or dosra wait ma phas jay ga
        // jab kam hujay ga process ka tu wo unlock hujay ga or dosra process ko lock mily ga or wo run hujay ga

        // 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;
}
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, thread4 , thread5;

    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);
    pthread_create(&thread4, NULL, process2, NULL);
    pthread_create(&thread5, 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_join(thread4, NULL);
    pthread_join(thread5, NULL);

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```

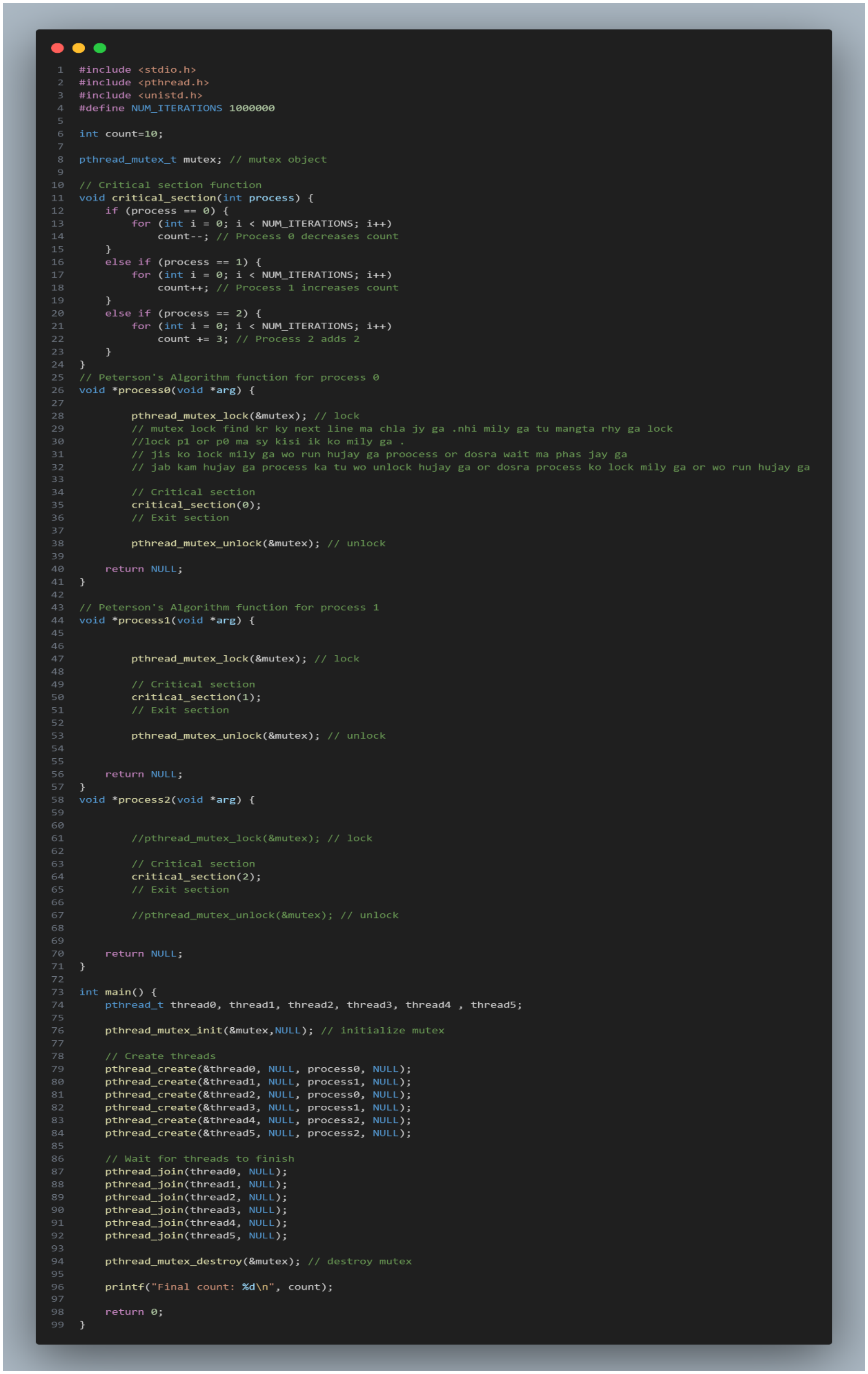


```c
void *process1(void *arg) {
```

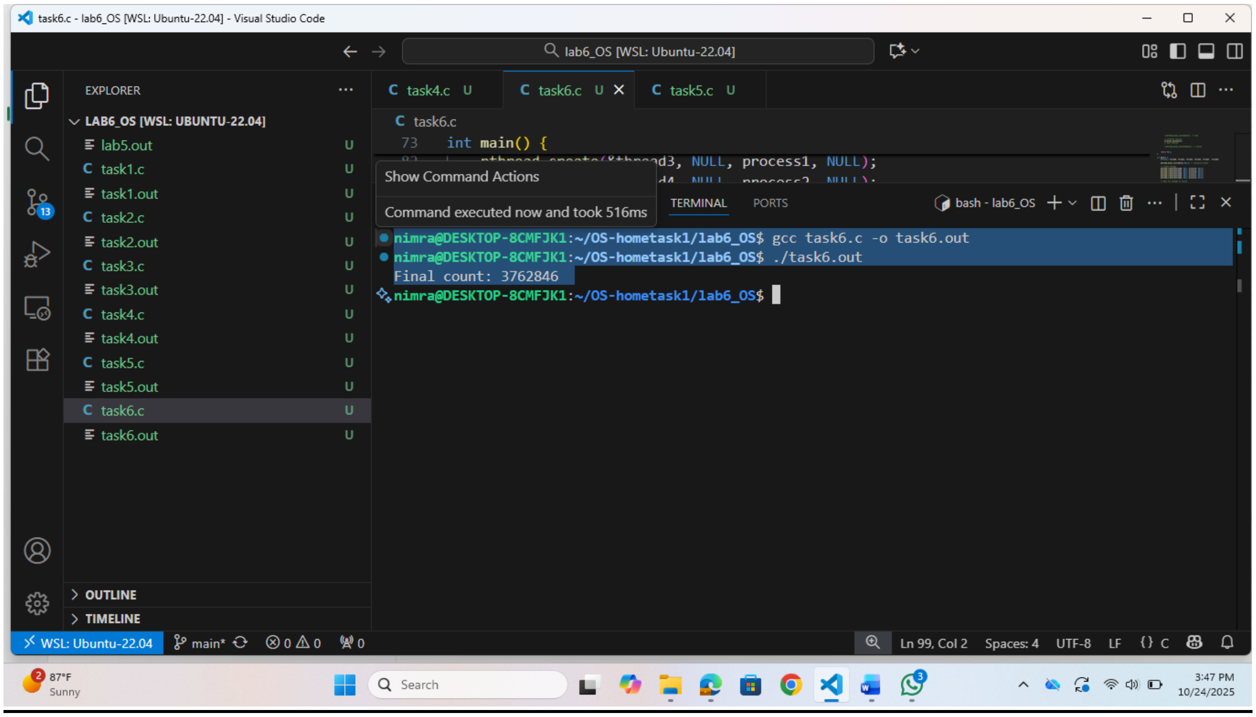
Command executed now and took 433ms

```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
ask1/lab6_OS$ gcc lab5.c -o lab5.out
h file or directory
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task5.c -o lab5.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task5.out
Final count: 4000010
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
```

Task 06:



```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) {
    if (process == 0) {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count--; // Process 0 decreases count
    }
    else if (process == 1) {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count++; // Process 1 increases count
    }
    else if (process == 2) {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count += 3; // Process 2 adds 2
    }
}
// Peterson's Algorithm function for process 0
void *process0(void *arg) {

        pthread_mutex_lock(&mutex); // lock
        // mutex lock find kr ky next line ma chla jy ga .nhi mily ga tu mangta rhy ga lock
        //lock p1 or p0 ma sy kisi ik ko mily ga .
        // jis ko lock mily ga wo run hujay ga proocess or dosra wait ma phas jay ga
        // jab kam hujay ga process ka tu wo unlock hujay ga or dosra process ko lock mily ga or wo run hujay ga

        // 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;
}
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, thread4 , thread5;

    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);
    pthread_create(&thread4, NULL, process2, NULL);
    pthread_create(&thread5, 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_join(thread4, NULL);
    pthread_join(thread5, NULL);

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```



```
int main() {
    pthread_create(&thread3, NULL, process1, NULL);
```

```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task6.c -o task6.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task6.out
Final count: 3762846
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
```

Difference between Peterson’s Algorithm and Mutex Algorithm?

Feature	Peterson’s Algorithm	Mutex Algorithm
Type	Software-based algorithm	Hardware/OS-supported mechanism
Implementation	Uses shared variables (flag, turn)	Uses OS/system calls (pthread_mutex_*)
Performance	Inefficient (CPU wasting)	Efficient (uses blocking)
Portability	Theoretical / Educational	Real-world use
Complexity	Simple, but limited	Abstracted by OS (complex)
Use Case	Teaching	Real-world concurrent

	synchronization concepts	programming