National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir

Submitted by:

Esha Mubashir Khan

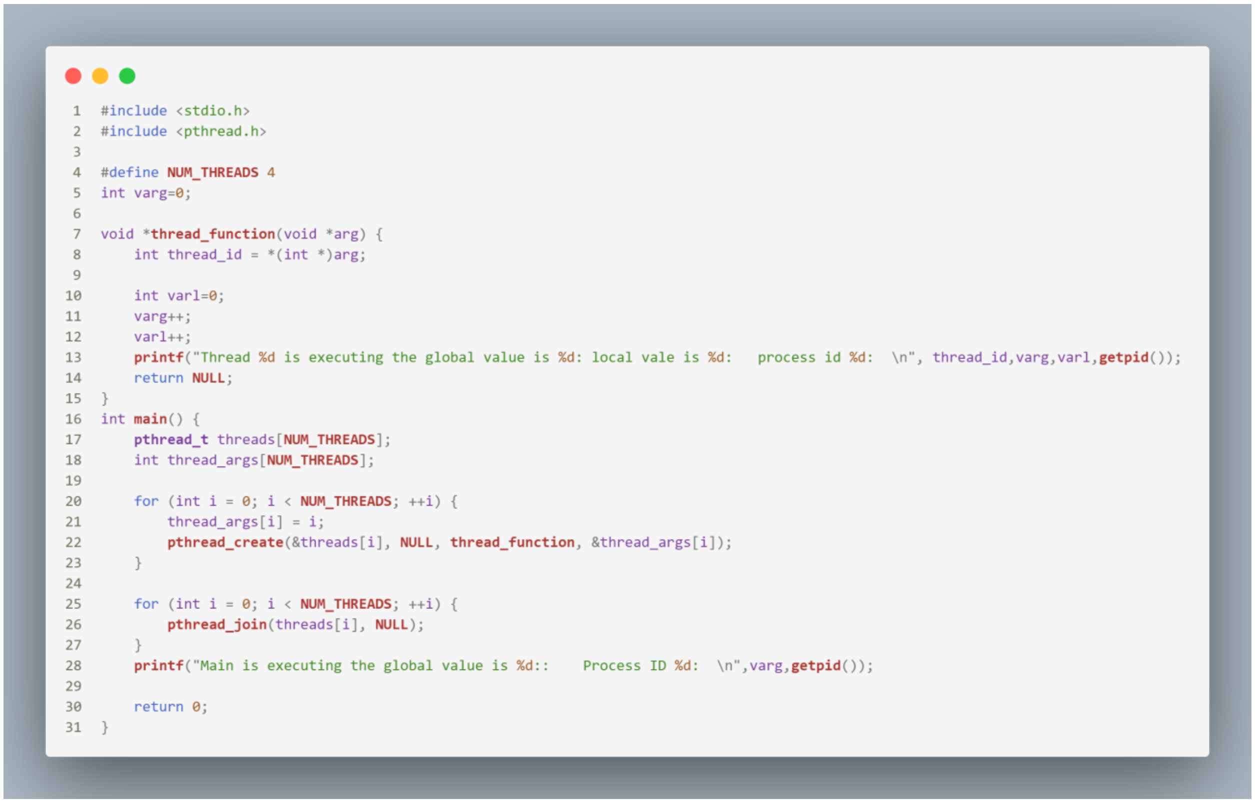
Reg number:

23-NTU-CS-1151

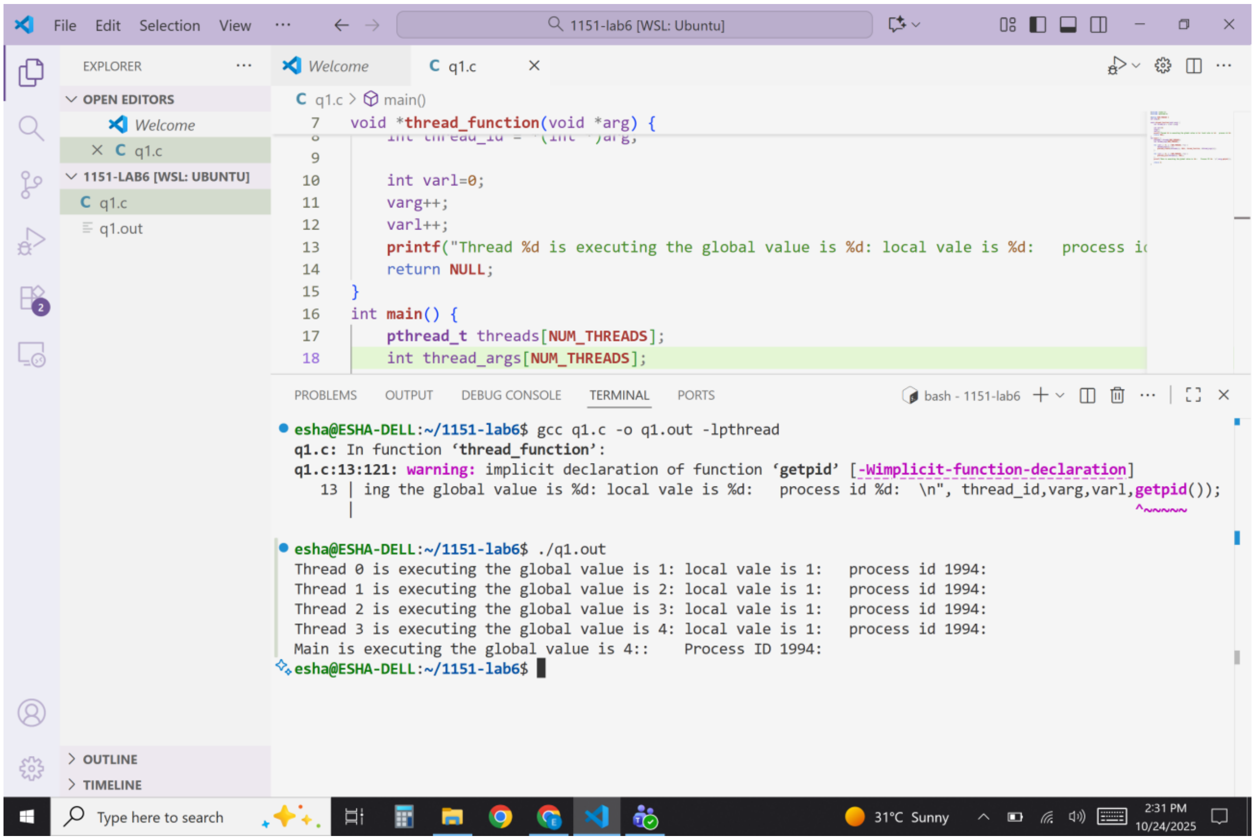
Lab no: **6**

Semester: **5th**

TASK 1:



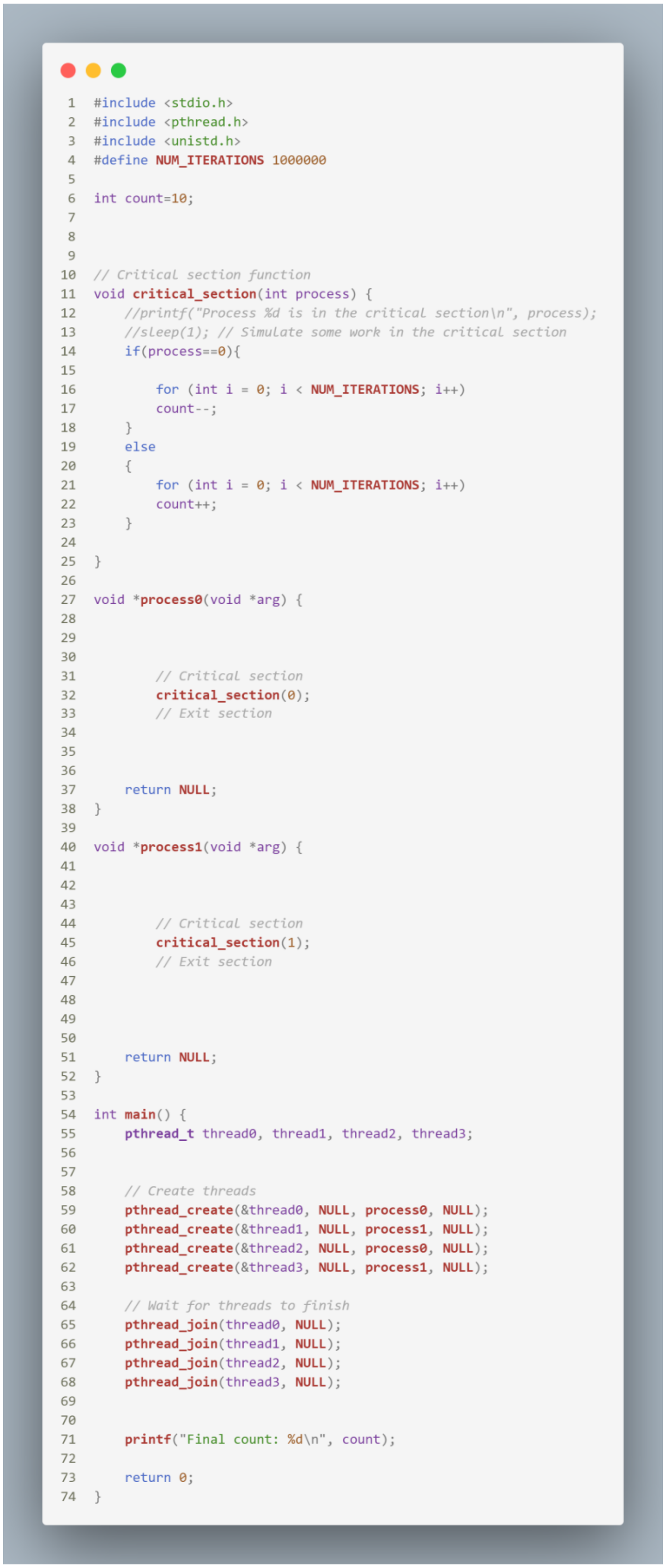
```c
#include <stdio.h>
#include <pthread.h>

#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;
}
```



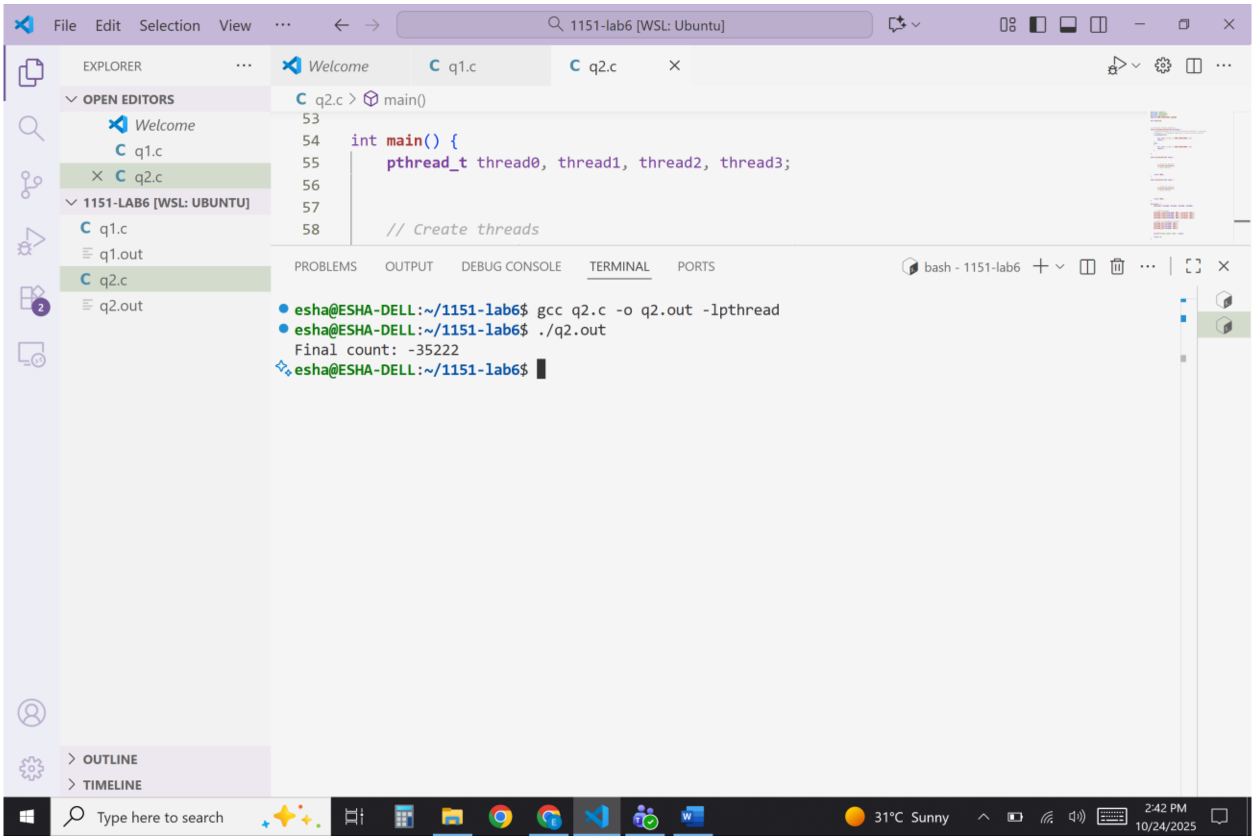
```c
void *thread_function(void *arg) {
    int thread_id = *(int *)arg;

    int varl=0;
    varg++;
    varl++;
    printf("Thread %d is executing the global value is %d: local vale is %d:   process i
    return NULL;
}
int main() {
    pthread_t threads[NUM_THREADS];
    int thread_args[NUM_THREADS];
```

```
esha@ESHA-DELL:~/1151-lab6$ gcc q1.c -o q1.out -lpthread
q1.c: In function ‘thread_function’:
q1.c:13:121: warning: implicit declaration of function ‘getpid’ [-Wimplicit-function-declaration]
   13 | ing the global value is %d: local vale is %d:   process id %d:  \n", thread_id,varg,varl,getpid());
      |                                                                                           ^~~~~~

esha@ESHA-DELL:~/1151-lab6$ ./q1.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 1994:
Thread 1 is executing the global value is 2: local vale is 1:   process id 1994:
Thread 2 is executing the global value is 3: local vale is 1:   process id 1994:
Thread 3 is executing the global value is 4: local vale is 1:   process id 1994:
Main is executing the global value is 4::    Process ID 1994:
esha@ESHA-DELL:~/1151-lab6$ 
```

TASK 2:



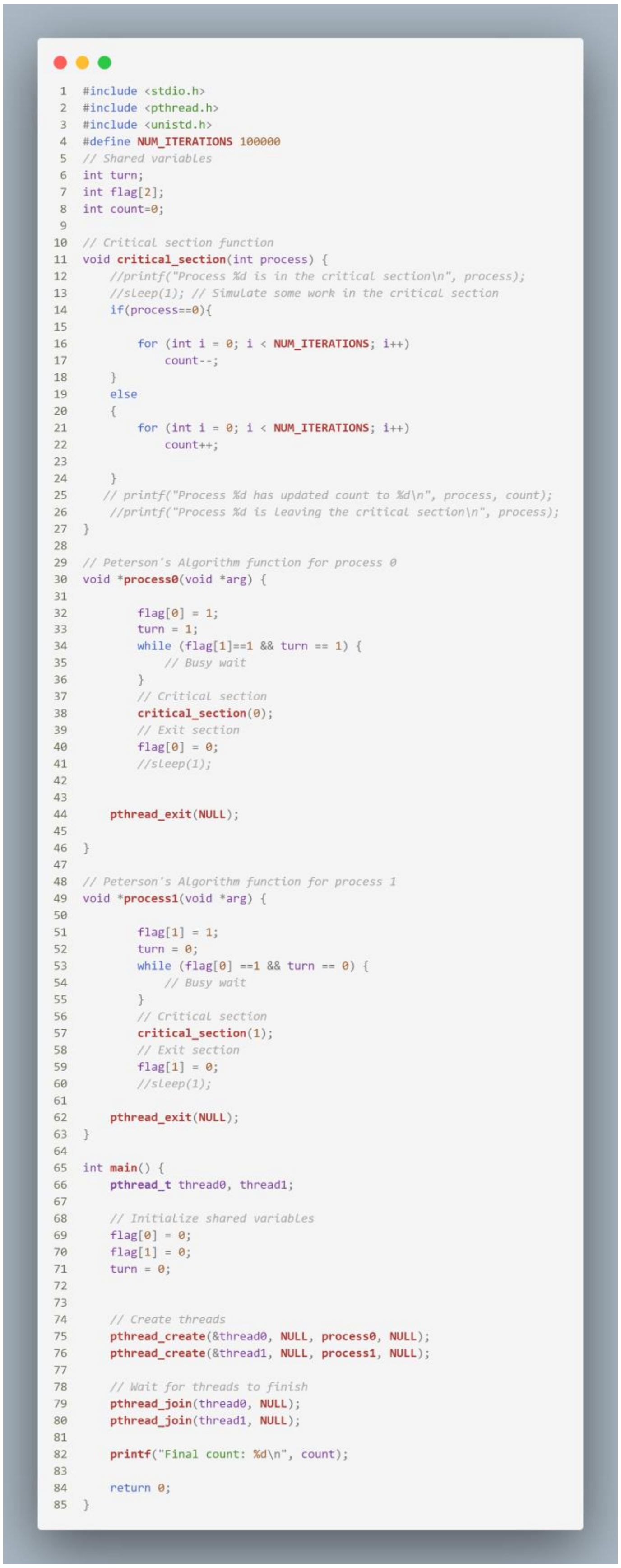
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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


    // Create threads
```

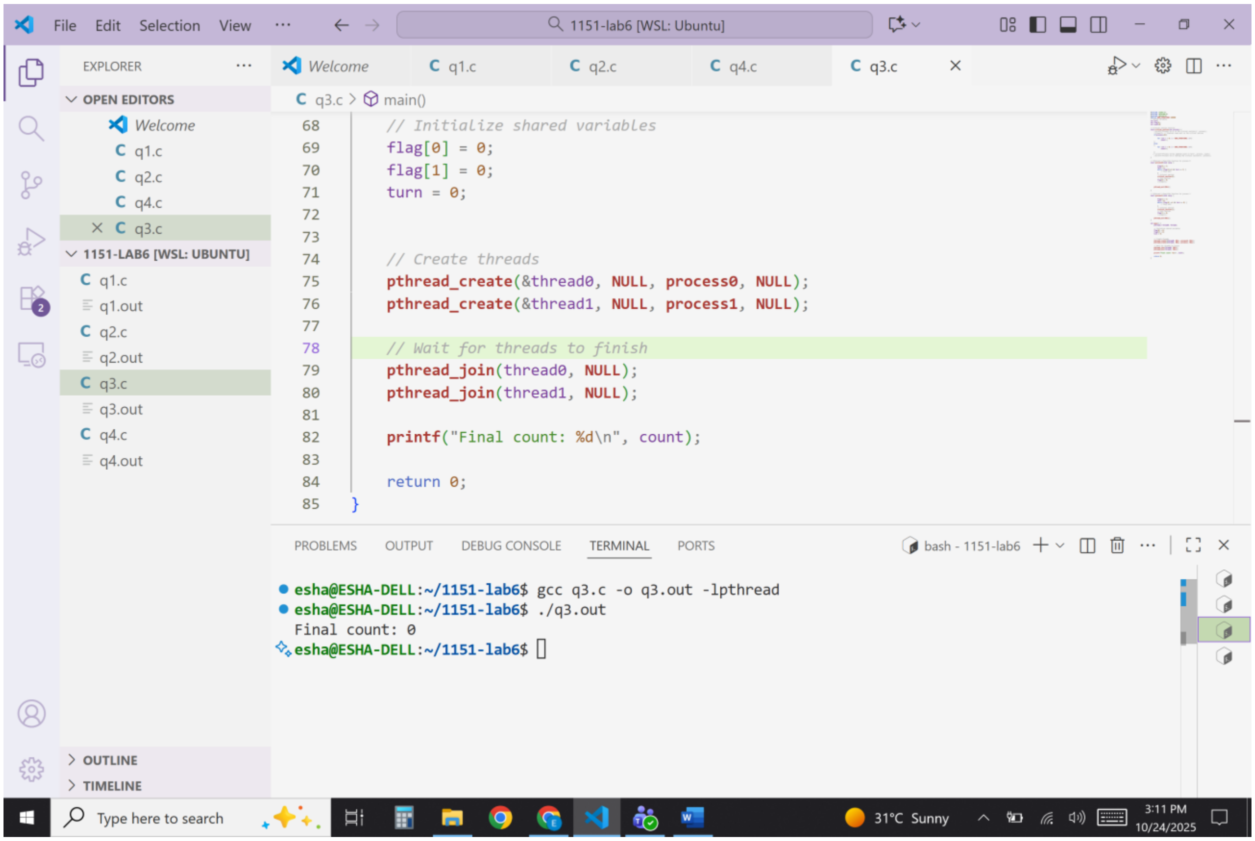
```
esha@ESHA-DELL:~/1151-lab6$ gcc q2.c -o q2.out -lpthread
esha@ESHA-DELL:~/1151-lab6$ ./q2.out
Final count: -35222
esha@ESHA-DELL:~/1151-lab6$ 
```

TASK 3:



```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;
}
```

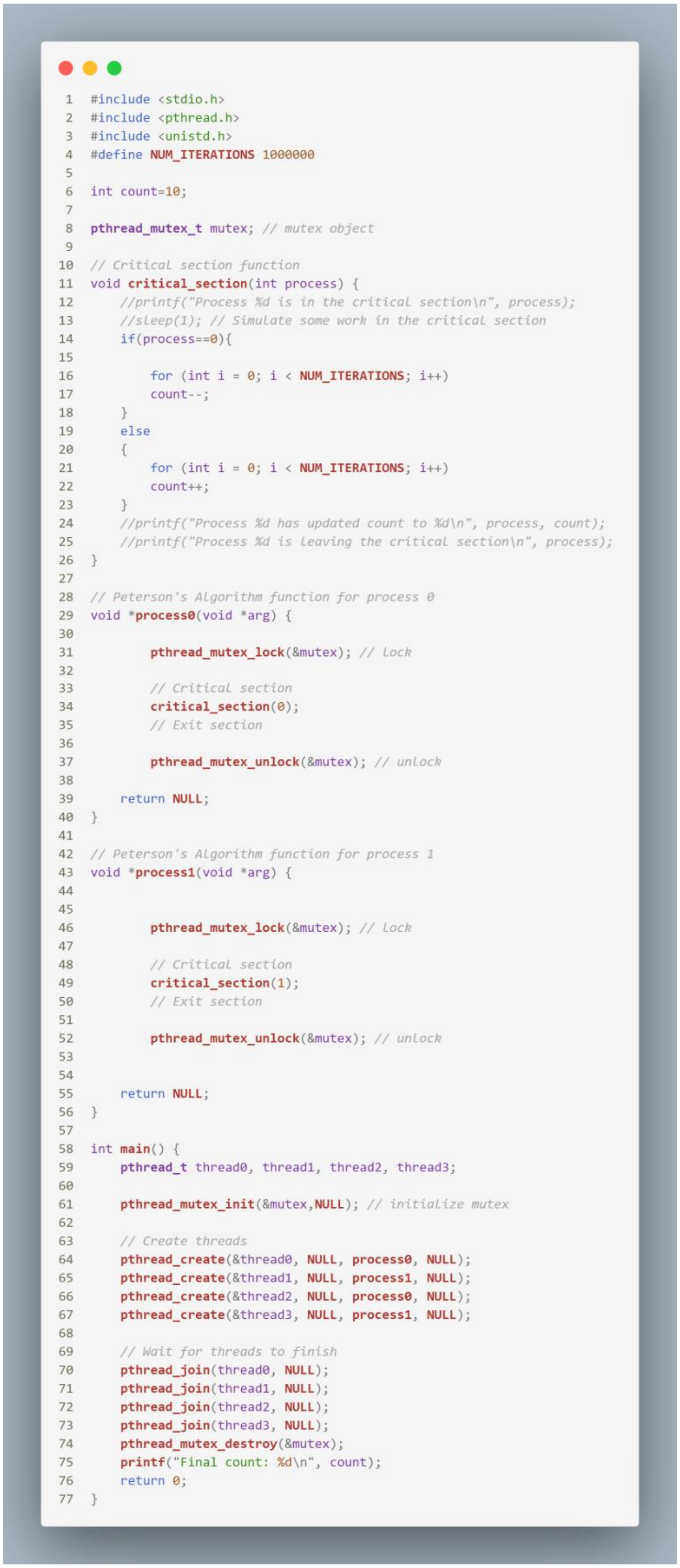
TASK 4:



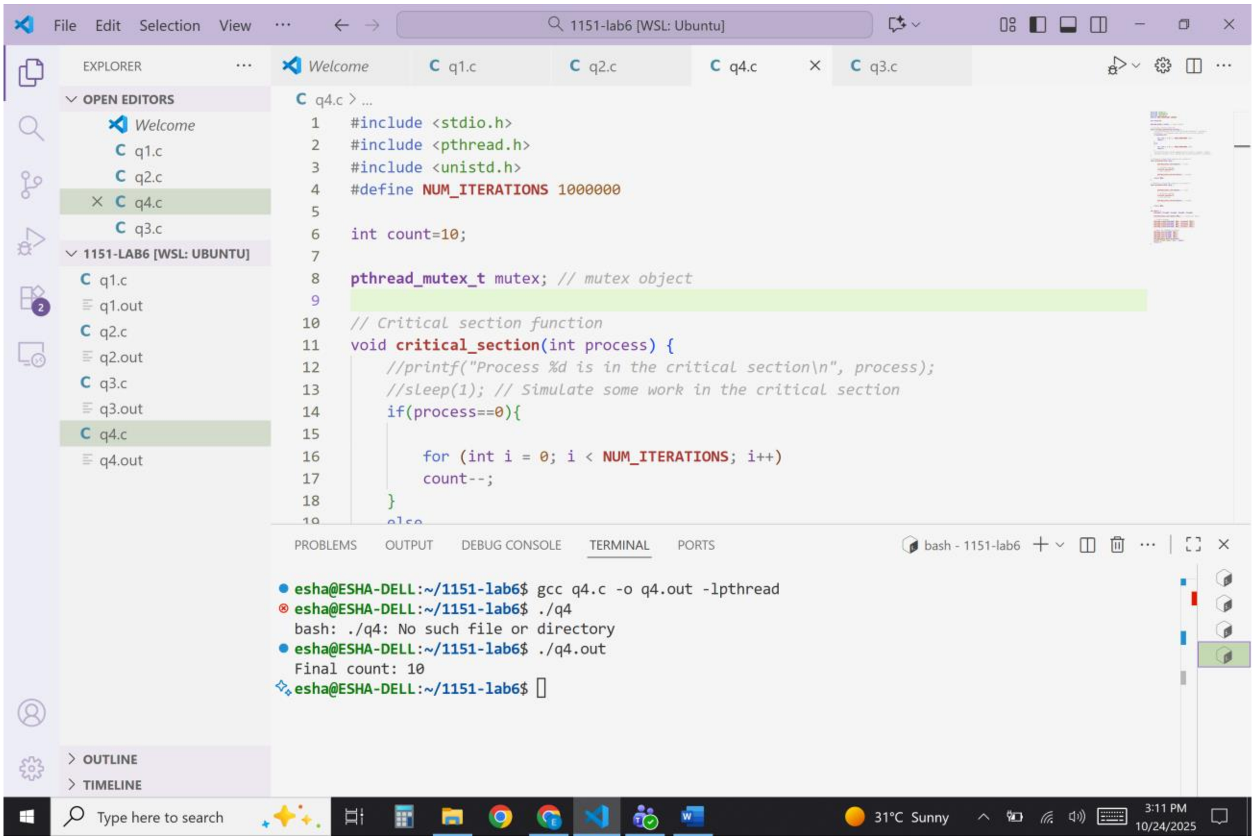
```c
    // Initialize shared variables
    flag[0] = 0;
    flag[1] = 0;
    turn = 0;


    // 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;
}
```

```
esha@ESHA-DELL:~/1151-lab6$ gcc q3.c -o q3.out -lpthread
esha@ESHA-DELL:~/1151-lab6$ ./q3.out
Final count: 0
esha@ESHA-DELL:~/1151-lab6$
```

TASK 5:



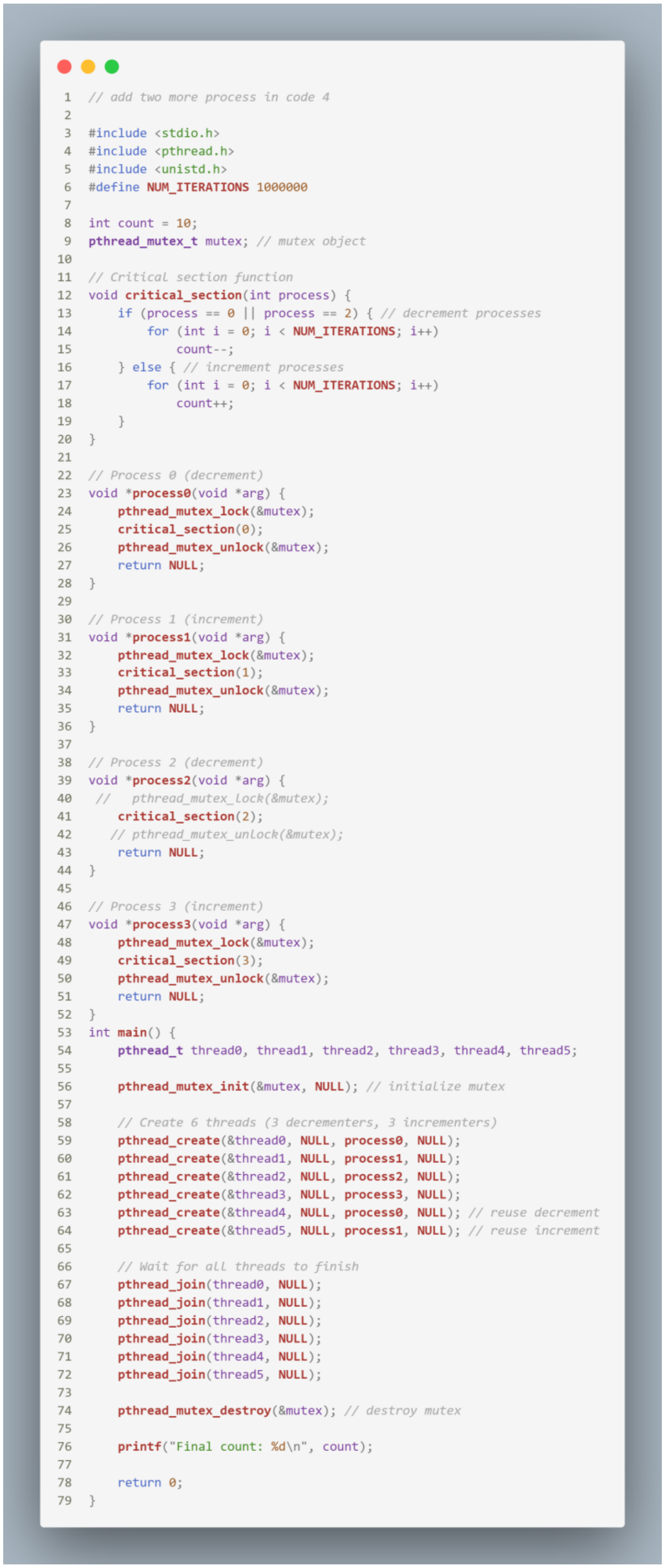
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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);
    printf("Final count: %d\n", count);
    return 0;
}
```



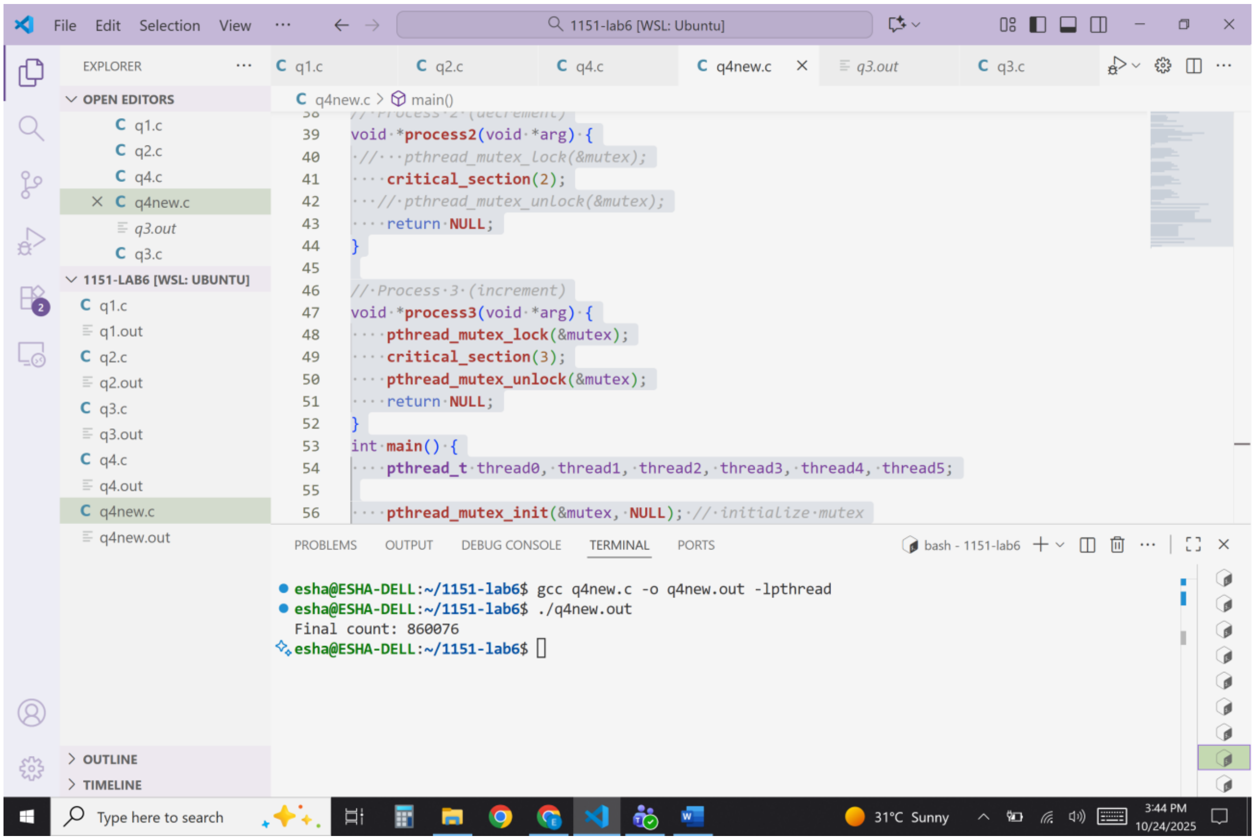
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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
```

```
esha@ESHA-DELL:~/1151-lab6$ gcc q4.c -o q4.out -lpthread
esha@ESHA-DELL:~/1151-lab6$ ./q4
bash: ./q4: No such file or directory
esha@ESHA-DELL:~/1151-lab6$ ./q4.out
Final count: 10
esha@ESHA-DELL:~/1151-lab6$ 
```

TASK 6:



```c
// add two more process in code 4

#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 || process == 2) { // decrement processes
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count--;
    } else { // increment processes
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count++;
    }
}

// Process 0 (decrement)
void *process0(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(0);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 1 (increment)
void *process1(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(1);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 2 (decrement)
void *process2(void *arg) {
 //   pthread_mutex_lock(&mutex);
    critical_section(2);
   // pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 3 (increment)
void *process3(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(3);
    pthread_mutex_unlock(&mutex);
    return NULL;
}
int main() {
    pthread_t thread0, thread1, thread2, thread3, thread4, thread5;

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

    // Create 6 threads (3 decrementers, 3 incrementers)
    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); // reuse decrement
    pthread_create(&thread5, NULL, process1, NULL); // reuse increment

    // Wait for all 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
// Process 2 (decrement)
void *process2(void *arg) {
 //   pthread_mutex_lock(&mutex);
    critical_section(2);
  // pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 3 (increment)
void *process3(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(3);
    pthread_mutex_unlock(&mutex);
    return NULL;
}
int main() {
    pthread_t thread0, thread1, thread2, thread3, thread4, thread5;

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

```
esha@ESHA-DELL:~/1151-lab6$ gcc q4new.c -o q4new.out -lpthread
esha@ESHA-DELL:~/1151-lab6$ ./q4new.out
Final count: 860076
esha@ESHA-DELL:~/1151-lab6$
```

Difference between Peterson's Algorithm & Mutex

Feature	Peterson's Algorithm	Mutex
Type	Software-based algorithm	Hardware/OS-supported mechanism
Number of Processes	Only 2 (basic form)	Any number of threads/processes
Implementation	Uses shared variables (flag, turn)	Uses OS/system calls (pthread_mutex_*)
Busy Waiting	Yes (spinlock)	No (threads sleep if lock unavailable)
Performance	Inefficient (CPU wasting)	Efficient (uses blocking)
Portability	Theoretical / Educational	Real-world use

Feature	**Peterson’s Algorithm**	**Mutex**
Guarantees	Mutual exclusion, progress, bounded waiting	Mutual exclusion (progress depends on scheduler)
Complexity	Simple, but limited	Abstracted by OS (complex internally)
Use Case	Teaching synchronization concepts	Real-world concurrent programming