National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Maha

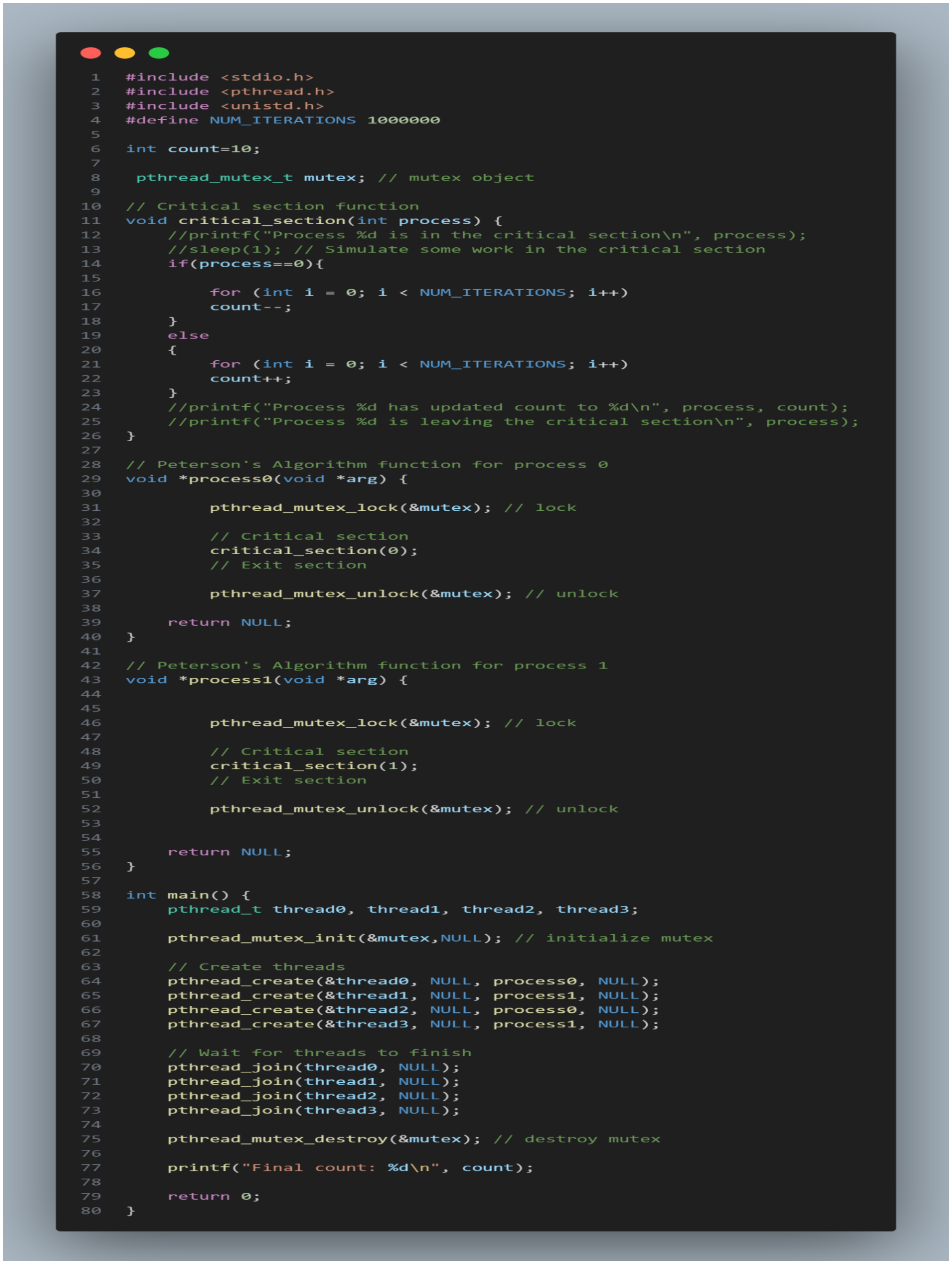
Reg. number: 23-NTU-CS-1170

Lab 6 (class task)

Semester:5th

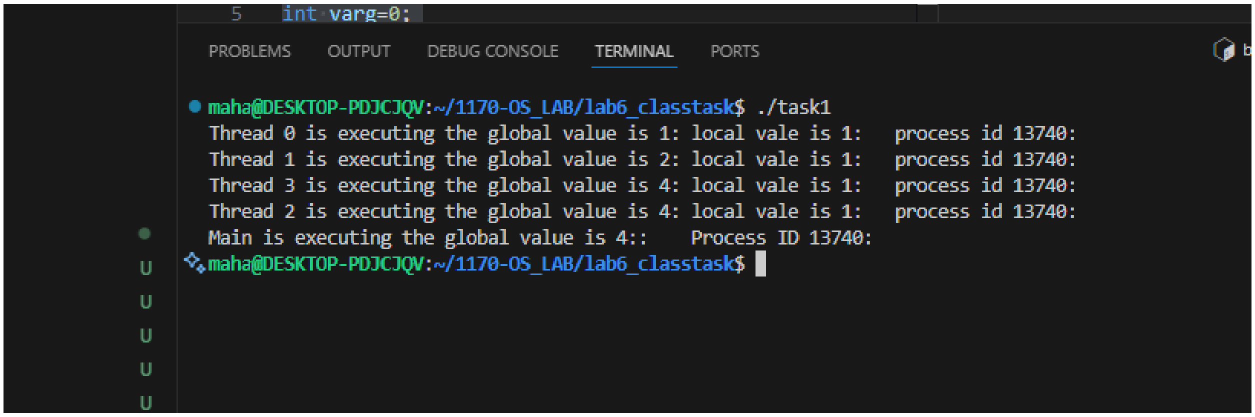
Task 1:

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;
}
```

Terminal:

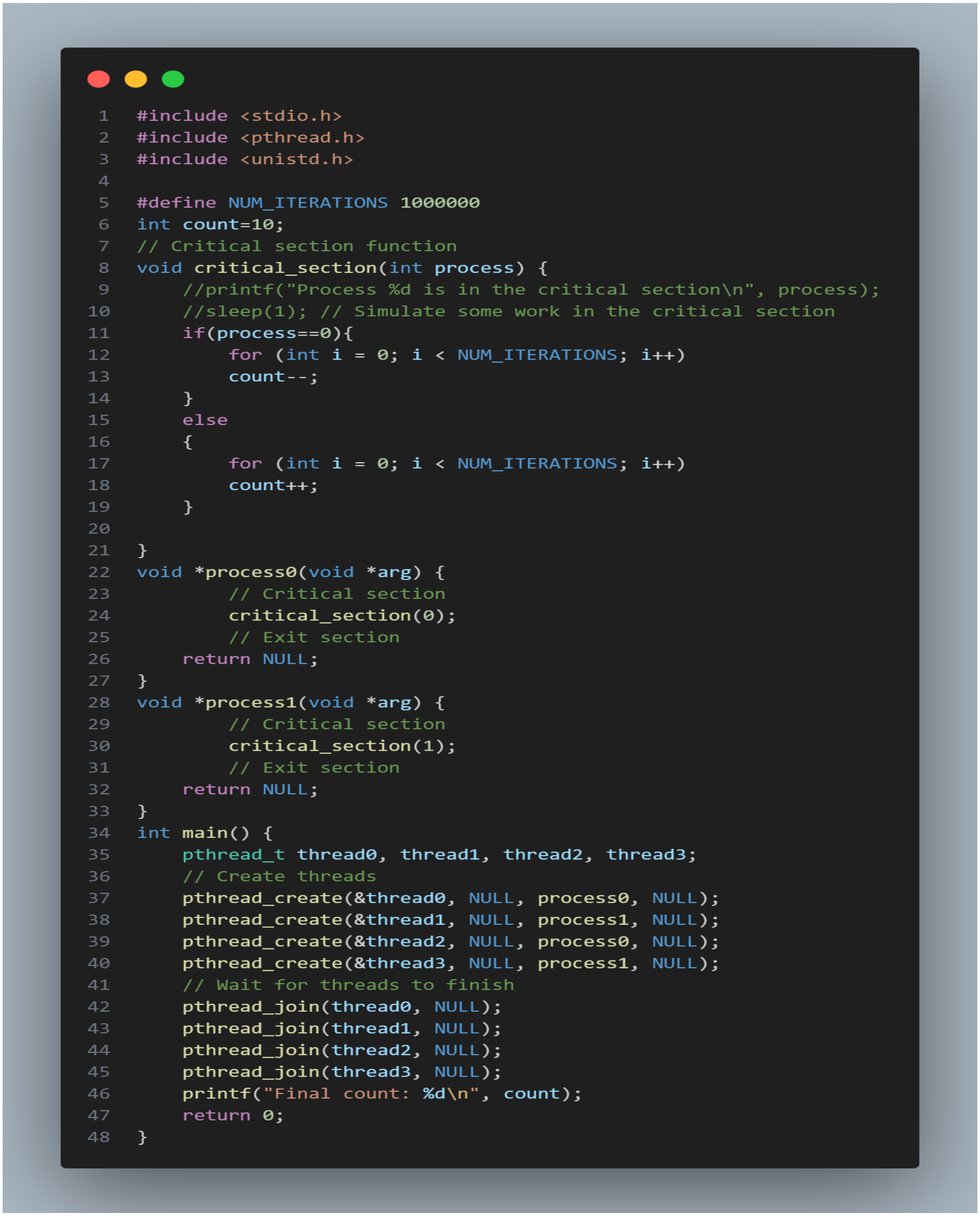


```
int varg=0;
```

```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task1
Thread 0 is executing the global value is 1: local vale is 1:   process id 13740:
Thread 1 is executing the global value is 2: local vale is 1:   process id 13740:
Thread 3 is executing the global value is 4: local vale is 1:   process id 13740:
Thread 2 is executing the global value is 4: local vale is 1:   process id 13740:
Main is executing the global value is 4::    Process ID 13740:
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ 
```

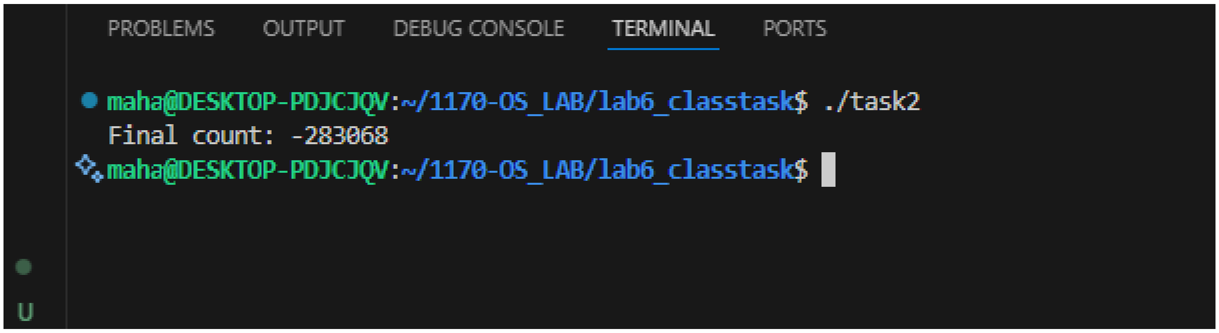
Question 2:

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;
}
```

Terminal:

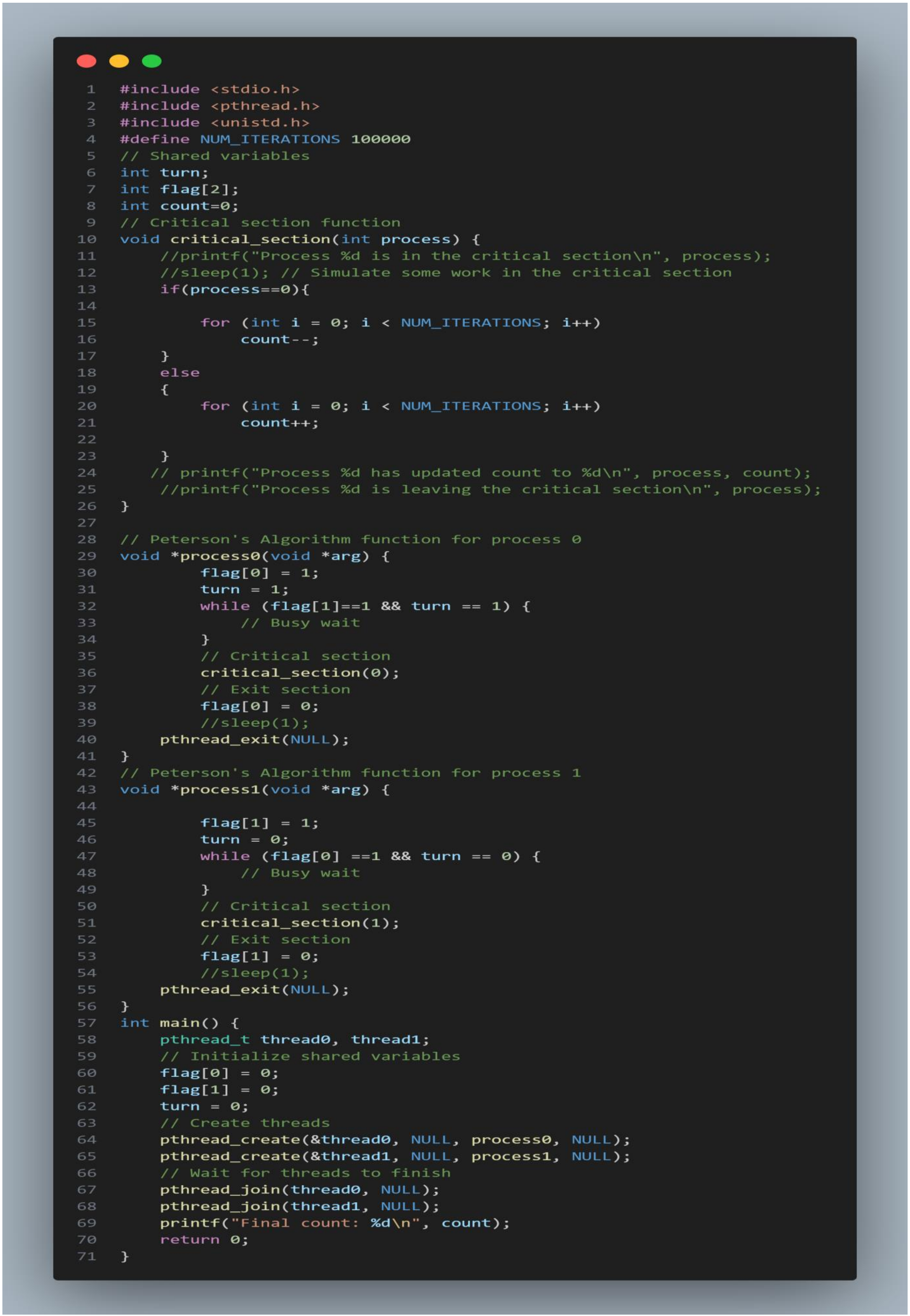


```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task2
Final count: -283068
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ 
```

Question 3:

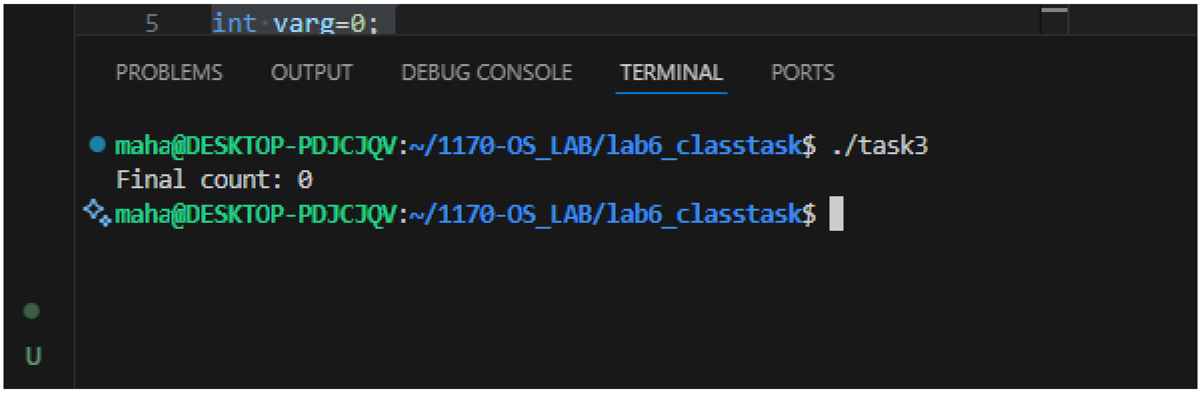
Peterson algorithm:

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;
}
```

Terminal:



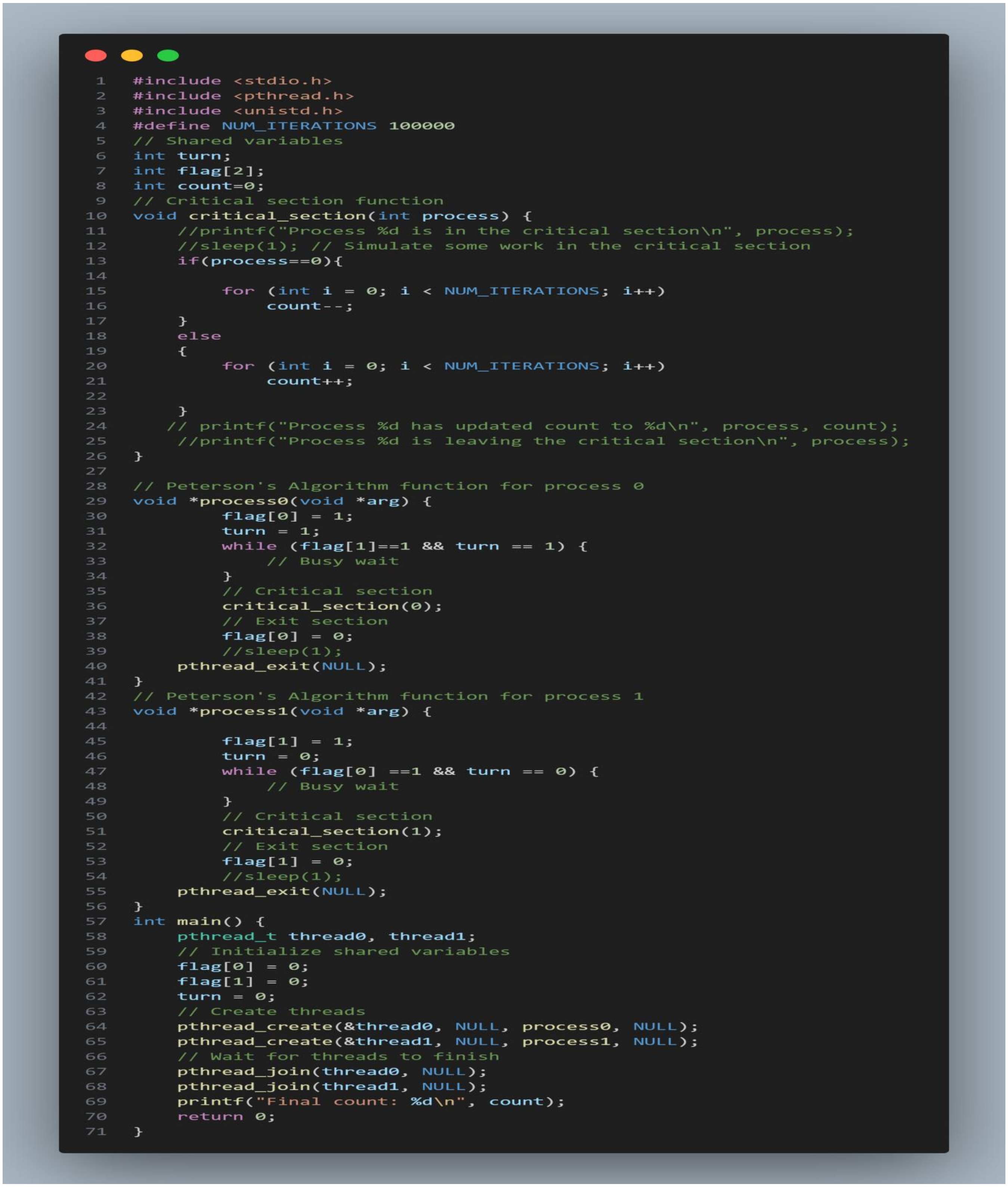
```
5    int varg=0;
```

```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task3
Final count: 0
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ 
```

Question 4:

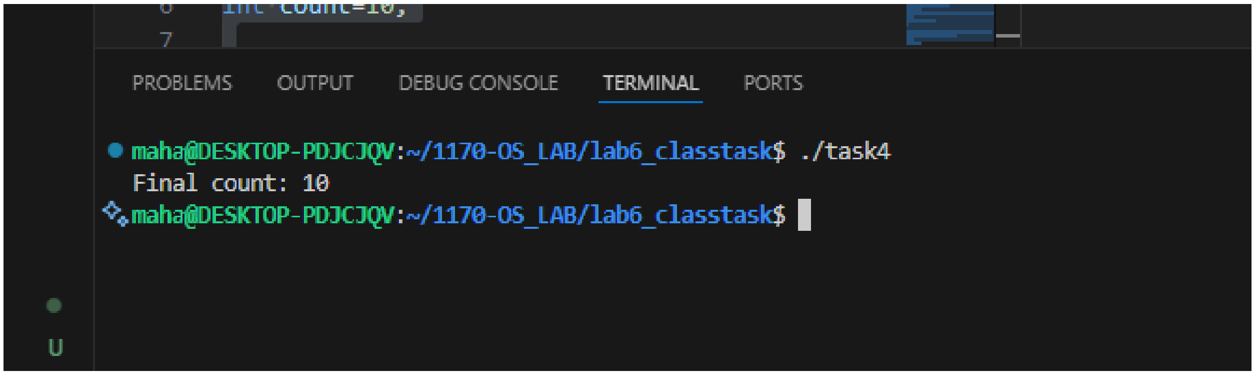
mutex locks

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;
}
```

Terminal:



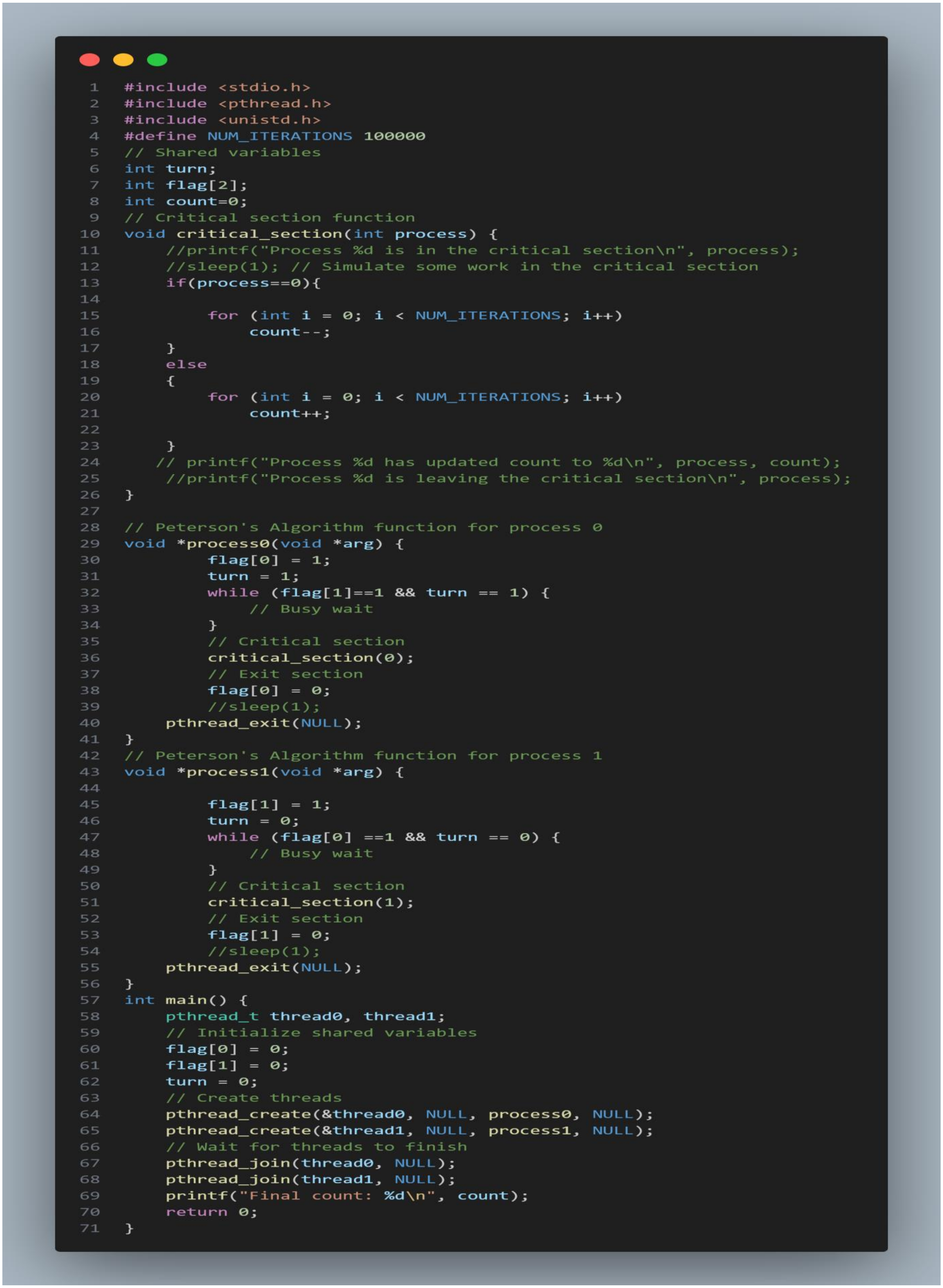
```
int count=10;
```

```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task4
Final count: 10
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ 
```

Question 5:

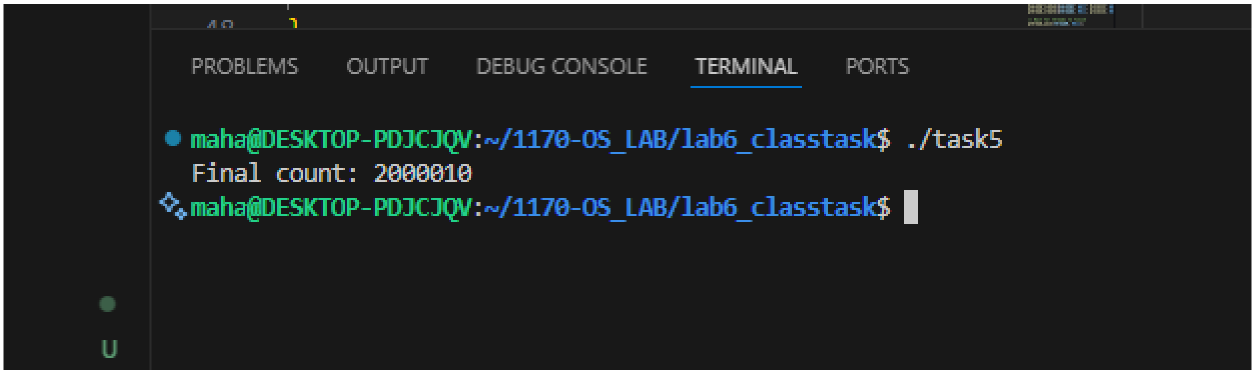
Do task4 with three process

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;
}
```

Terminal:



```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task5
Final count: 2000010
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ 
```

Compare Peterson and mutex locks:

Peterson’s Algorithm and mutex locks both prevent race conditions, but Peterson’s works only for two threads and wastes CPU time by busy waiting. Mutex locks are faster, more efficient, and support many threads, so they’re better for real-world use.