NATIONAL TEXTILE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENC

SUBMITTED BY:

Eman Faisal 23-NTU-CS-1149

SECTION SE: 5th (A)

Operating Systems- LAB6 activity

SUBMITTED TO:

Sir Nasir Mahmood

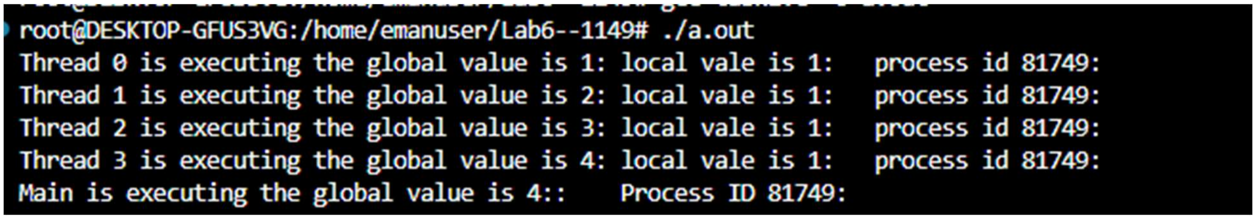
SUBMISSION DATE: 10/24/25

Task1:

Code:

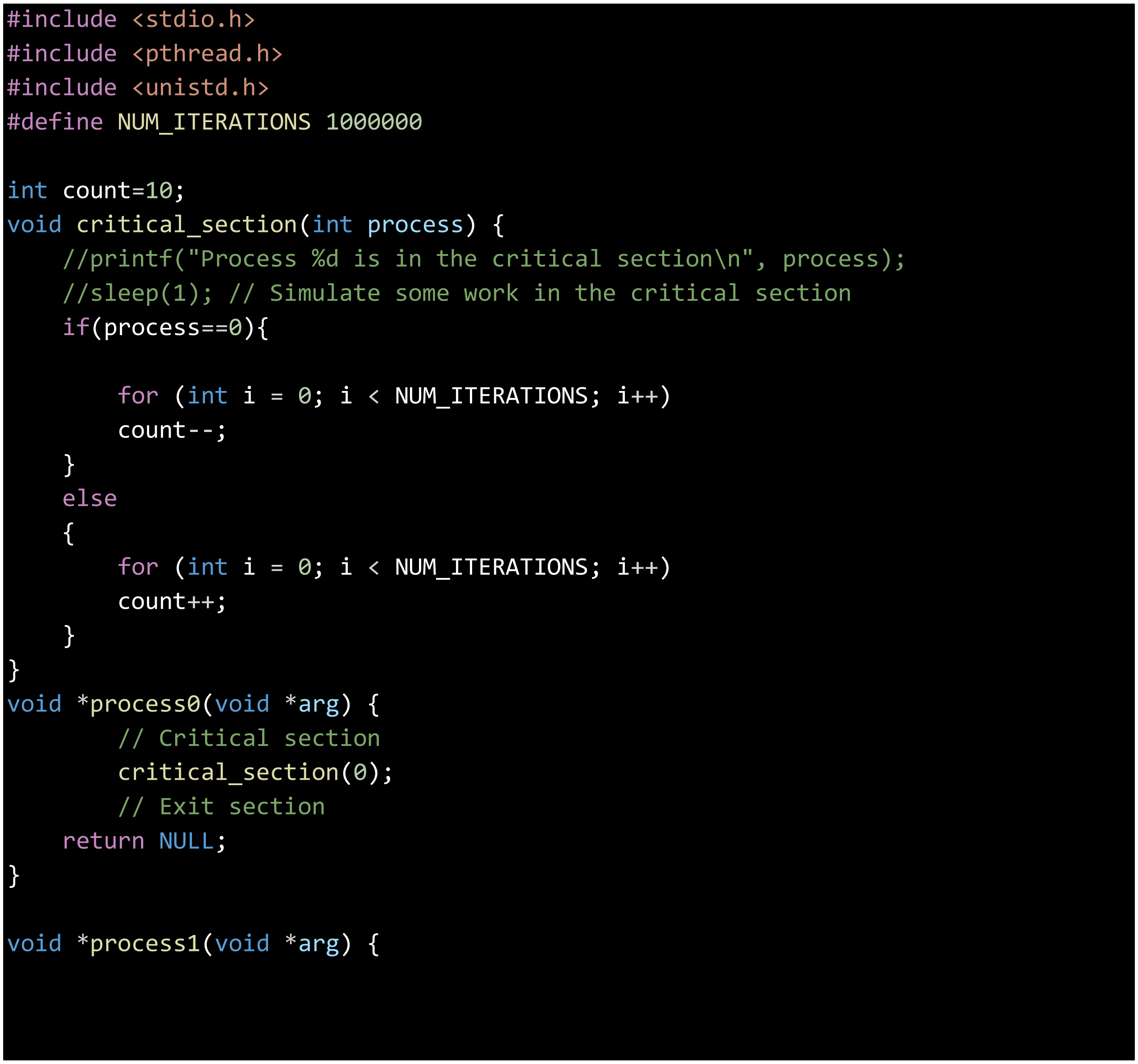
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#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:

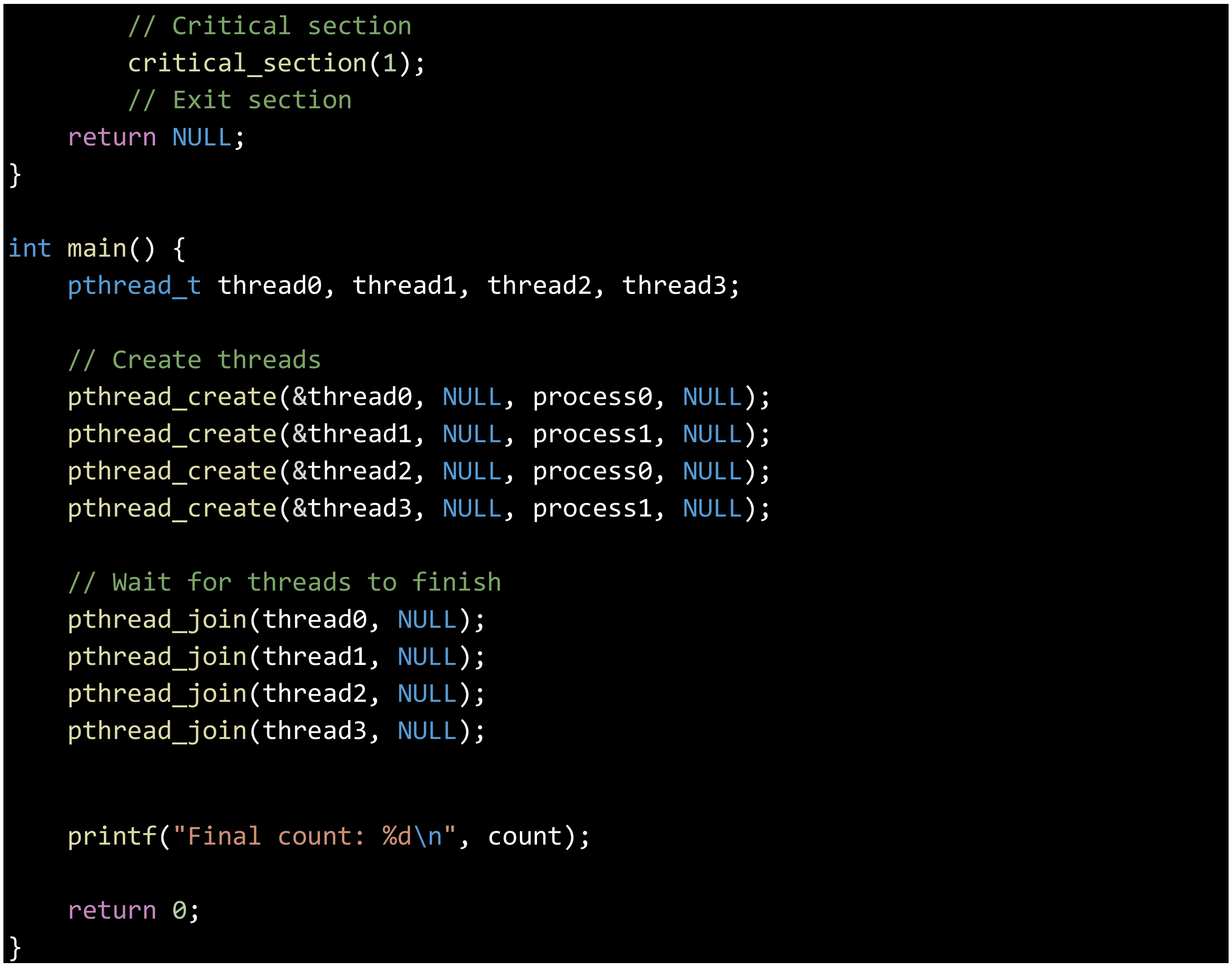


```
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 81749:
Thread 1 is executing the global value is 2: local vale is 1:   process id 81749:
Thread 2 is executing the global value is 3: local vale is 1:   process id 81749:
Thread 3 is executing the global value is 4: local vale is 1:   process id 81749:
Main is executing the global value is 4::    Process ID 81749:
```

Task2:
Code:

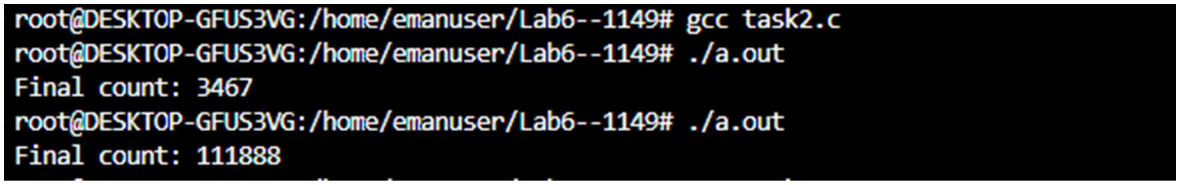


```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

int count=10;
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) {
```



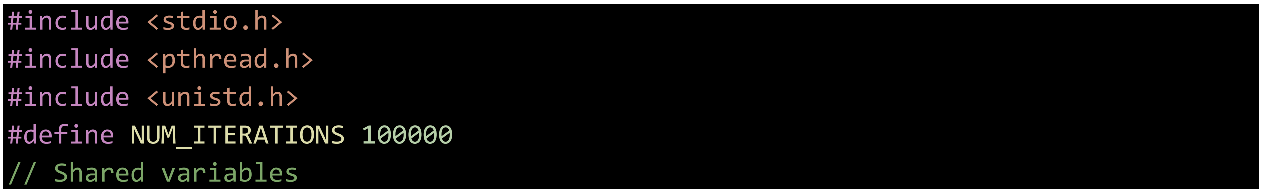
```c
        // Critical section
        critical_section(1);
        // Exit section
    return NULL;
}

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:

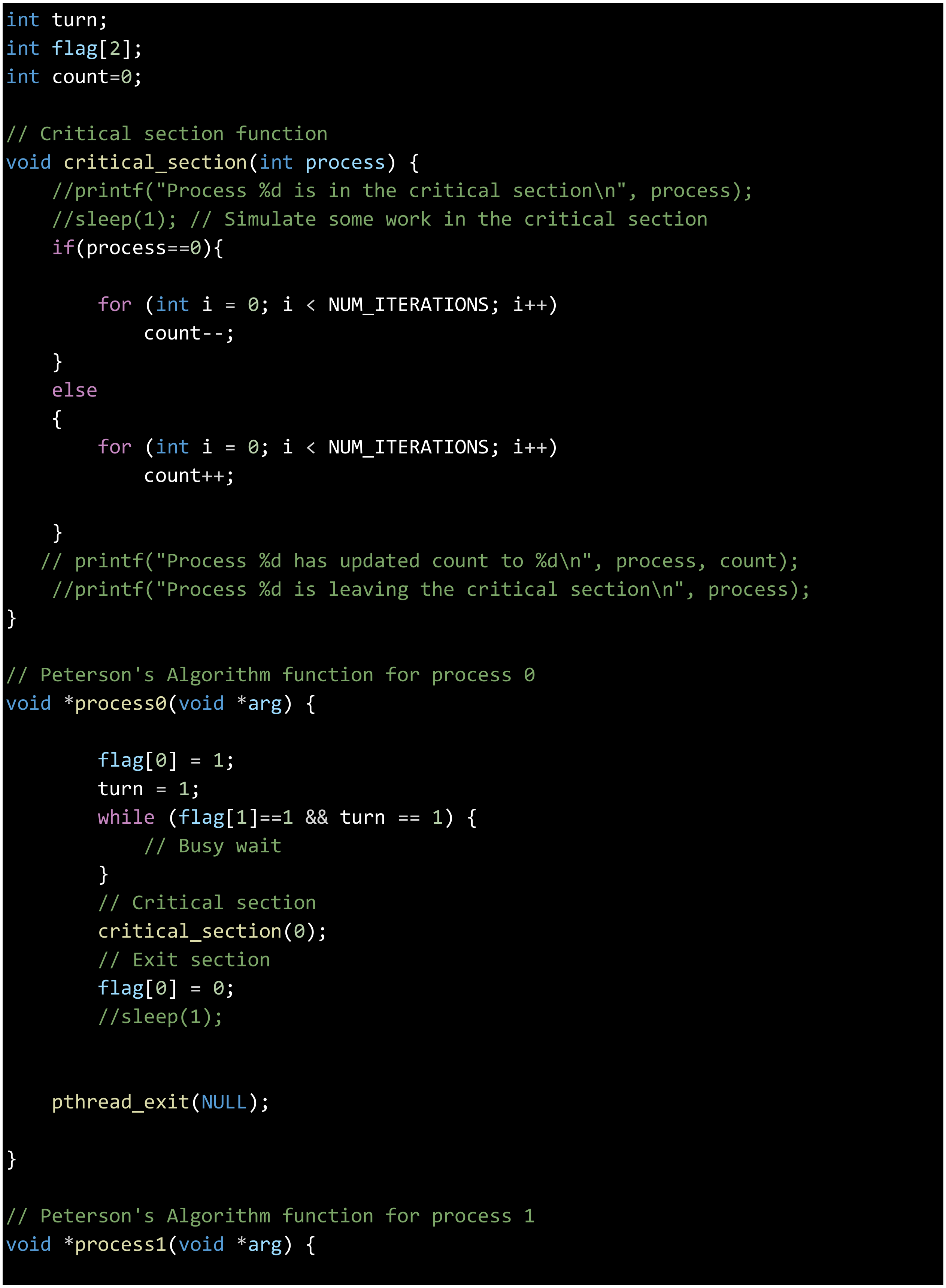


```
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# gcc task2.c
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
Final count: 3467
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
Final count: 111888
```

Task3:
Code:

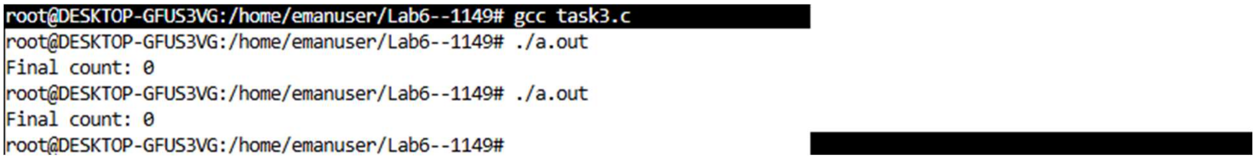


```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 100000
// Shared variables
```



```
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:

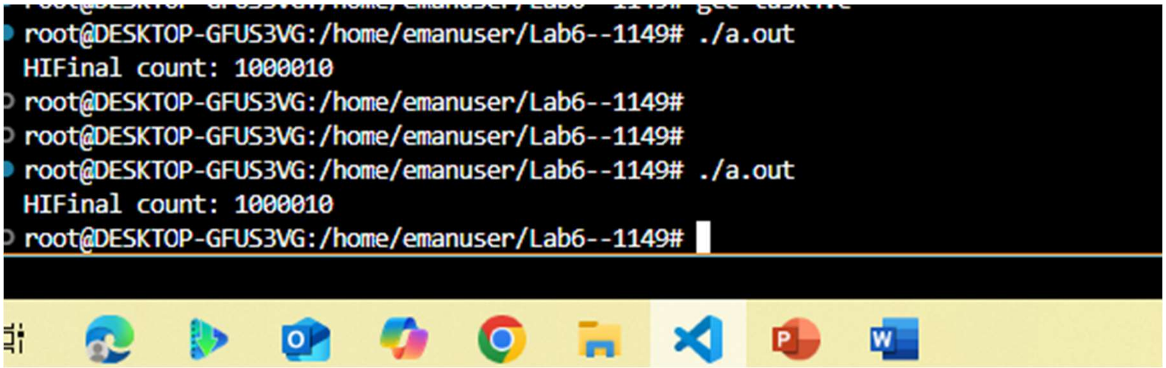


```
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# gcc task3.c
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
Final count: 0
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
Final count: 0
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149#
```

Task4:
Code:

```
#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 if(process==1)
    {
        for (int i = 0; i < NUM_ITERATIONS; i++)
        count++;
    }
    else{
        for (int i = 0; i < NUM_ITERATIONS; i++)
        count+=2;
        printf("HI");
    }
    //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;
}

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:



```
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# gcc task1.c
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
HIFinal count: 1000010
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149#
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149#
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149# ./a.out
HIFinal count: 1000010
root@DESKTOP-GFUS3VG:/home/emanuser/Lab6--1149#
```

Compare and Contrast Peterson and Mutex:

Both mutex and Peterson do the same task but work very differently.

Peterson	Mutex
Peterson's algorithm work for only 2 processes	mutex can work for any number of threads.
Peterson use shared variables like flag and turn	mutex uses kernel support and CPU instructions
we apply complete logic ourself.	Mutex uses built in function.
In Peterson the process turns its own flag to 0 after critical section.	In mutex,we unlock to give our turn to next process
In Peterson,the loop continues until the condition is met	in mutex it just blocks thread until lock is free.