National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Eman Marium Tariq Rao

Reg number:

23-NTU-CS-1150

Semester:

05

Task1:

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>   // ✅ Needed for getpid()

#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 | Global value: %d | Local value: %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 | Global value: %d | Process ID: %d\n", varg,
getpid());
    return 0;
}
```

Terminal:

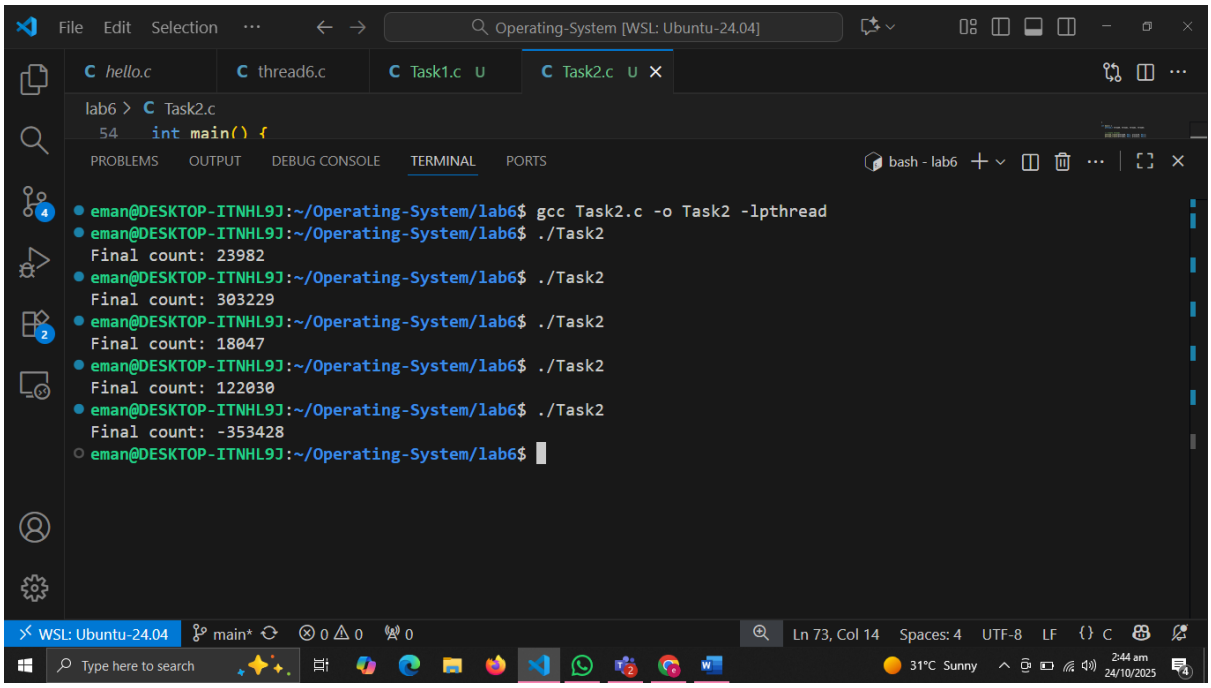
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        bash - lab6

eman@DESKTOP-ITNHL9J:~/Operating-System/lab6$ gcc Task1.c -o Task1 -lpthread
eman@DESKTOP-ITNHL9J:~/Operating-System/lab6$ ./Task1
Thread 0 is executing | Global value: 1 | Local value: 1 | Process ID: 9858
Thread 1 is executing | Global value: 2 | Local value: 1 | Process ID: 9858
Thread 2 is executing | Global value: 3 | Local value: 1 | Process ID: 9858
Thread 3 is executing | Global value: 4 | Local value: 1 | Process ID: 9858
Main is executing | Global value: 4 | Process ID: 9858
eman@DESKTOP-ITNHL9J:~/Operating-System/lab6$
```

Task2:

```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:



Task3:

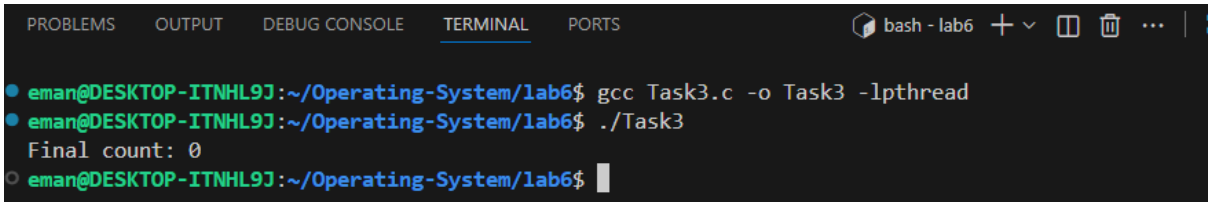
With PeterSon Algorithm

```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:



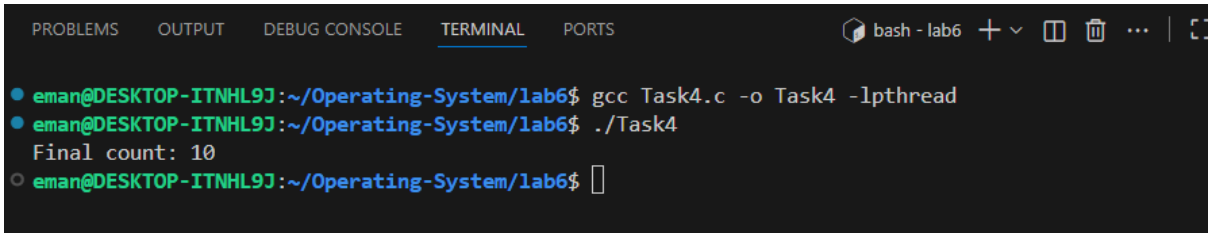
Task4:

```
#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:



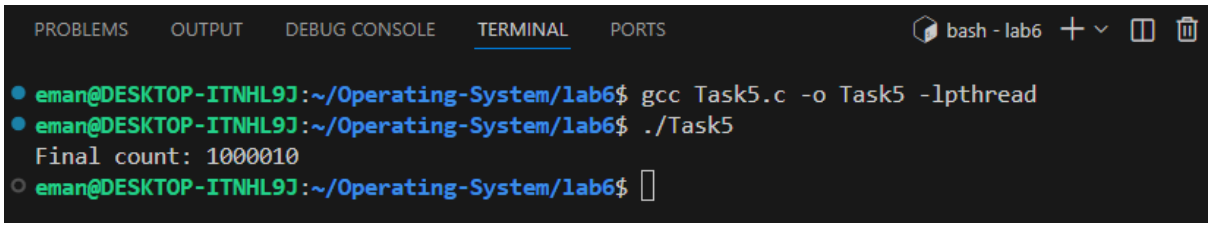
Task5:

```
#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("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(1);
        // Exit section

        pthread_mutex_unlock(&mutex); // unlock
```

```c
    return NULL;
}


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

    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, process2, NULL);


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


    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```

Terminal:



Comparison:

Mutex(Lock)	Peterson Algorithm
• Mutex uses OS lock object. • pthread_mutex_t mutex;	• Peterson uses shared flag[] and turn variables.
• Mutex initialized via API.	• Peterson manually sets shared variables.
• Handled by OS	• Manual “busy waiting” until safe to enter.
• Mutex can handle 3+ threads.	• Peterson works only for 2 threads.
• Threads sleep when locked.	• Continuous while-loop consumes CPU.