

**NATIONAL TEXTILE**

**UNIVERSITY**

DEPARTMENT OF COMPUTER SCIENCE

**SUBMITTED BY:**

Farah Naz 23-NTU-CS-1152

**SECTION SE: 5th(A)**

**LAB MANUAL**

**SUBMITTED TO:**

Sir Nasir Mehmood

**SUBMISSION DATE:**

10-24-2025

**Task 1:**

**Code:**

//EXECUTE MULTIPLE THREADS WITH RACE CONDITION

#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);

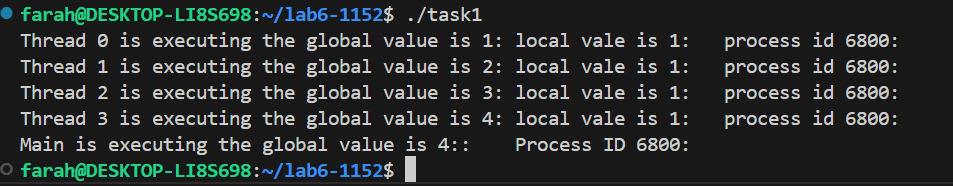
    }

    printf("Main is executing the global value is %d::    Process ID %d:  \n",varg,getpid());

    return 0;

}

**Output:**



**Task 2:**

**Code:**

#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);

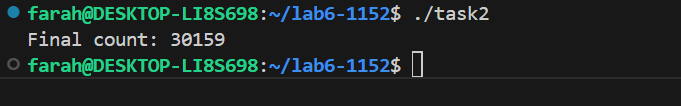
    pthread\_join(thread3, NULL);

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

    return 0;

}

**Output:**



**Task 3:**

**Code:**

//PETERSON ALGORITHM

#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);

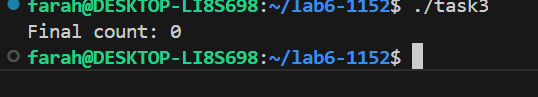
    pthread\_join(thread1, NULL);

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

    return 0;

}

**Output:**



**Task 4:**

**Code:**

//MUTEX ALGORITHM

#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(2);

        // Exit section

        pthread\_mutex\_unlock(&mutex); // unlock

    return NULL;

}

int main() {

    pthread\_t thread0, thread1, thread2, thread3,thread4;

    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);

    // 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);

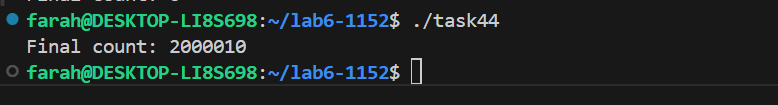
    pthread\_mutex\_destroy(&mutex); // destroy mutex

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

    return 0;

}

**Output:**



**Compare and contrast Peterson and Mutex:**

|  |  |
| --- | --- |
| **Peterson** | **Mutex** |
| 1. Peterson works only for two processes/threads | 1. Mutex can work for 1000s of threads. |
| 1. In Peterson we have to manually define the turn, flag value of the processes and also have to define while condition to apply mutual exclusion. | 2. In Mutex, Mutex lock automatically do all of this for us. |
| 1. Use shared variables and busy waiting. | 3. Use CPU instructions and OS \_\_\_\_\_kernel Support. |
| 1. Exit- we have to set flag to 0 to implement exit. | 4. Mutex-unlock do this automatically |
| 1. Programmer manually enforce mutual exclusion. | 5. The OS kernel and **pthread** library \_\_\_\_\_handles it automatically. |
| 1. Low efficiency – Wastes CPU cycles. | 6. High efficiency- OS kernel handles |