#### 

National Textile University

**Department of Computer Science**

Subject: Operating System

Submitted to: Sir Nasir Mahmood

Submitted by: Eisha Muzaffar

Reg. number: 23-NTU-CS-1147

Lab no.: lab6

semester:5th

**Question no:01**

#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);

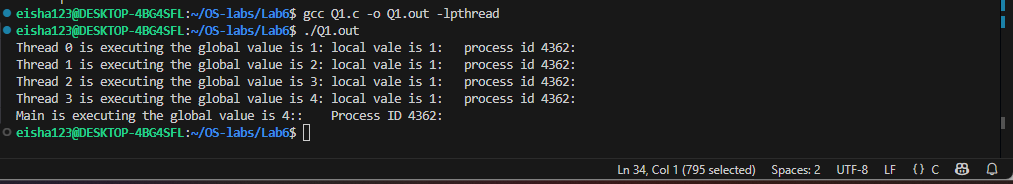
    }

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

return 0;

}

**Terminal:**

****

**Question no:02**

#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);

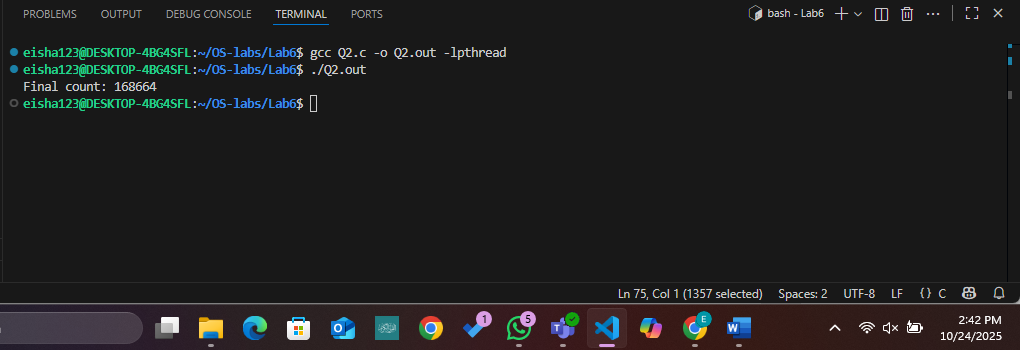
    pthread\_join(thread3, NULL);

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

return 0;

}

**Terminal:**

****

**Question no:03**

**Implementing Peterson:**

#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);

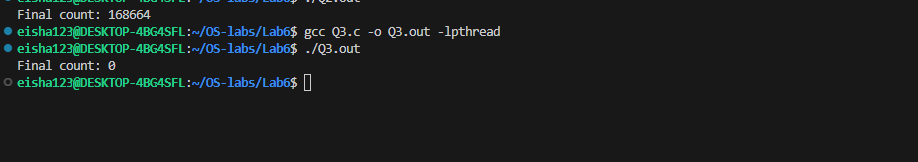
    pthread\_join(thread1, NULL);

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

return 0;

}

**Terminal:**

****

**Question 4:**

**Mutex:**

#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); // Finding 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);

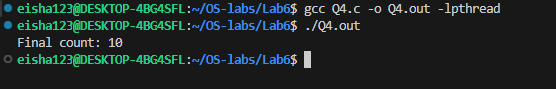
    pthread\_mutex\_destroy(&mutex); // destroy mutex

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

return 0;

}

**Terminal:**

****

**Question no:06**

#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) {

        for (int i = 0; i < NUM\_ITERATIONS; i++)

            count--;

    }

    else if (process == 1) {

        for (int i = 0; i < NUM\_ITERATIONS; i++)

            count++;

    }

    else if (process == 2) {

        for (int i = 0; i < NUM\_ITERATIONS; i++)

            count += 2;

    }

}

// Process 0

void \*process0(void \*arg) {

    pthread\_mutex\_lock(&mutex);

    critical\_section(0);

    pthread\_mutex\_unlock(&mutex);

    return NULL;

}

// Process 1

void \*process1(void \*arg) {

    pthread\_mutex\_lock(&mutex);

    critical\_section(1);

    pthread\_mutex\_unlock(&mutex);

    return NULL;

}

// New Process 2

void \*process2(void \*arg) {

    pthread\_mutex\_lock(&mutex);

    critical\_section(2);

    pthread\_mutex\_unlock(&mutex);

    return NULL;

}

int main() {

    pthread\_t thread0, thread1, thread2;

    pthread\_mutex\_init(&mutex, NULL); // initialize mutex

    // Create 3 threads (3 processes)

    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);

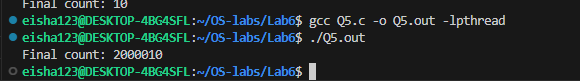
    pthread\_mutex\_destroy(&mutex); // destroy mutex

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

    return 0;

}

**Terminal:**

****

**Comparison:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Peterson’s Algorithm** | **Mutex** |
| Type | Software synchronization | Hardware/OS synchronization |
| Waiting | Busy wait | Blocking (no CPU waste) |
| Process limit | 2 only | Many |
| Speed | Slower | Faster |
| Simplicity | Complex | Simple to use |
| Portability | Works in pure C (no  OS functions) | Needs pthread library (OS-level) |

**Question 6:**

**Without lock and unlock:**

#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) {

        for (int i = 0; i < NUM\_ITERATIONS; i++)

            count--;

    }

    else if (process == 1) {

        for (int i = 0; i < NUM\_ITERATIONS; i++)

            count++;

    }

    else if (process == 2) {

        for (int i = 0; i < NUM\_ITERATIONS; i++)

            count += 2;

    }

}

// Process 0

void \*process0(void \*arg) {

    pthread\_mutex\_lock(&mutex);

    critical\_section(0);

    pthread\_mutex\_unlock(&mutex);

    return NULL;

}

// Process 1

void \*process1(void \*arg) {

    pthread\_mutex\_lock(&mutex);

    critical\_section(1);

    pthread\_mutex\_unlock(&mutex);

    return NULL;

}

// New Process 2

void \*process2(void \*arg) {

    critical\_section(2);

    return NULL;

}

int main() {

    pthread\_t thread0, thread1, thread2;

    pthread\_mutex\_init(&mutex, NULL); // initialize mutex

    // Create 3 threads (3 processes)

    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);

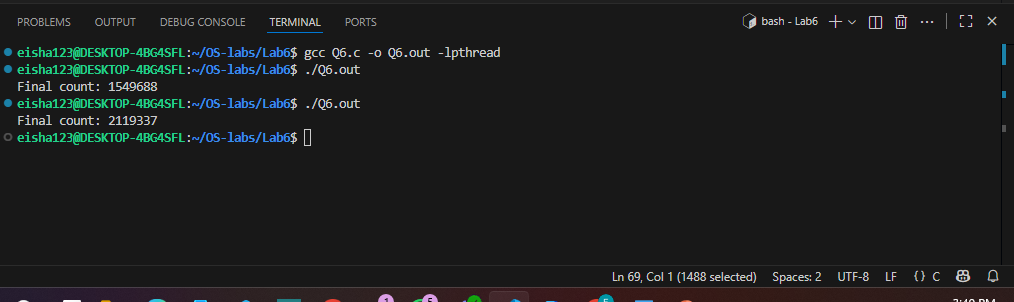
    pthread\_mutex\_destroy(&mutex); // destroy mutex

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

    return 0;

}

**Terminal:**

****