National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Naseer

Submitted by:

Ayema

Reg number:

23-NTU-CS-1142

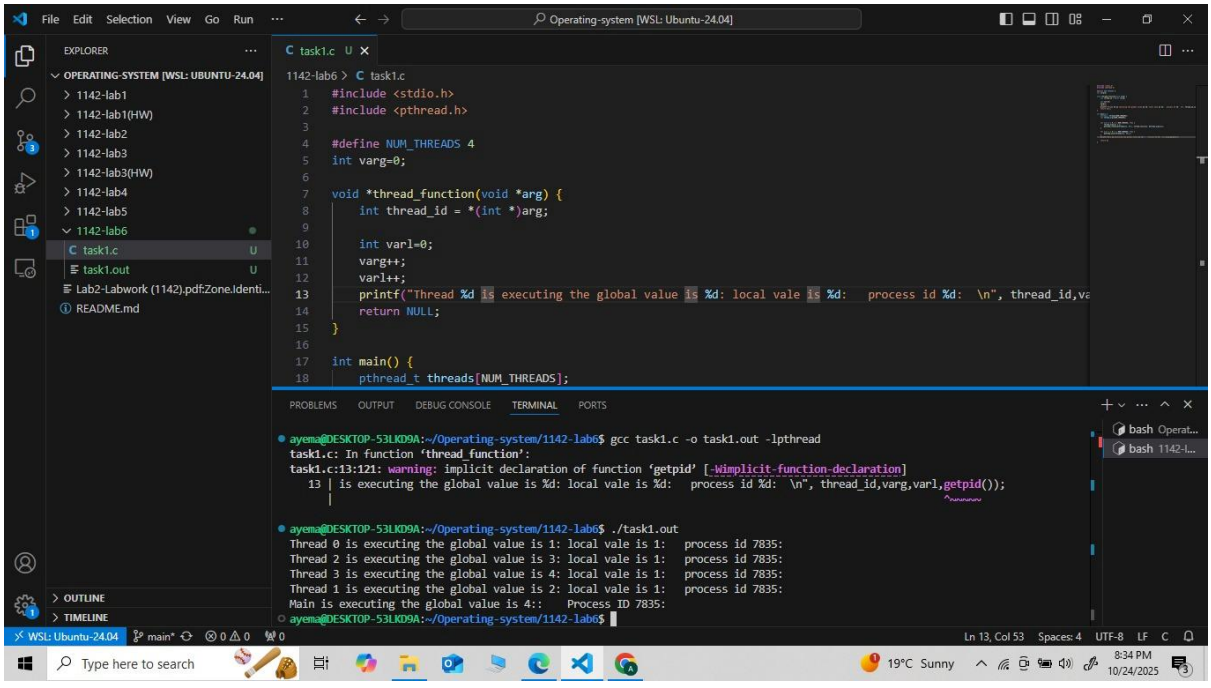
Lab no. :

05

Semester: 5Th

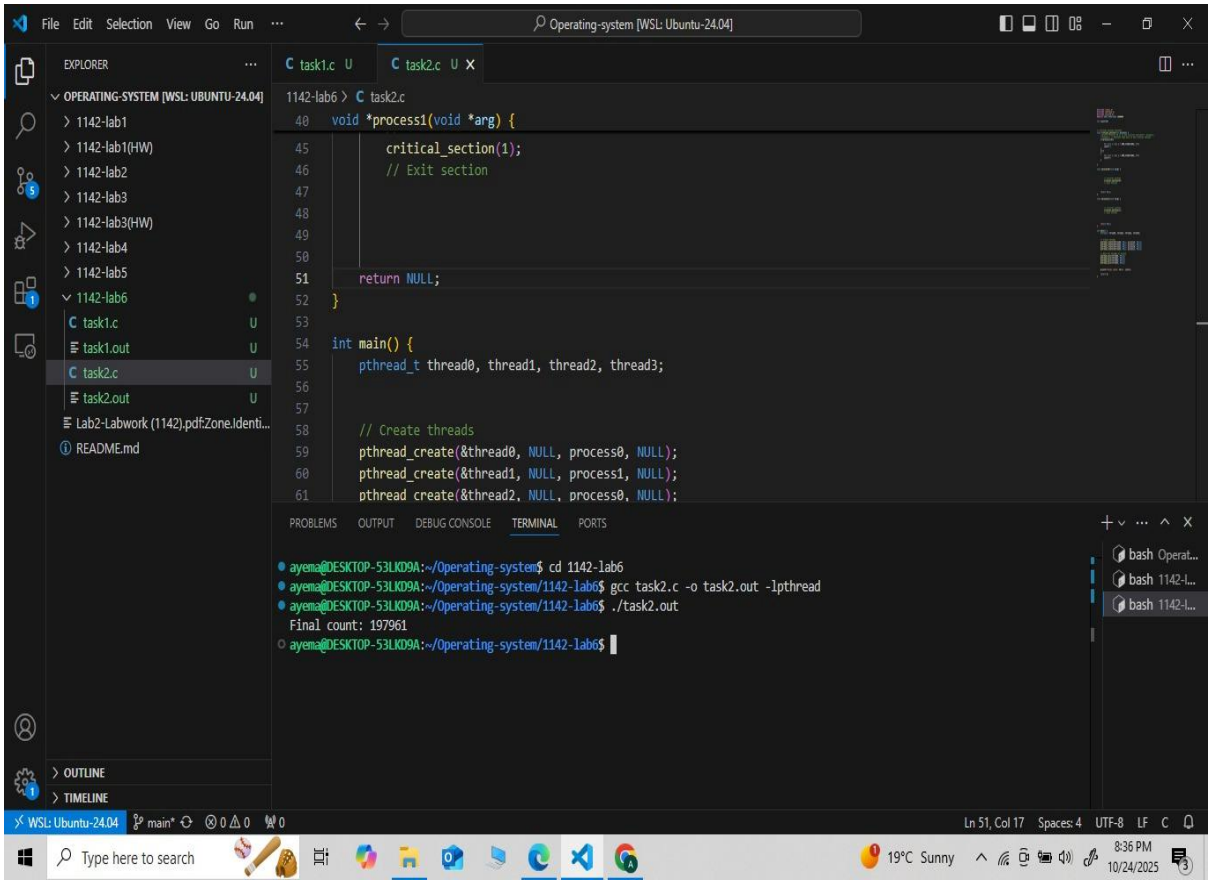
Task 1:

```c
#include <stdio.h>
#include <pthread.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;
}
```



Task 2:

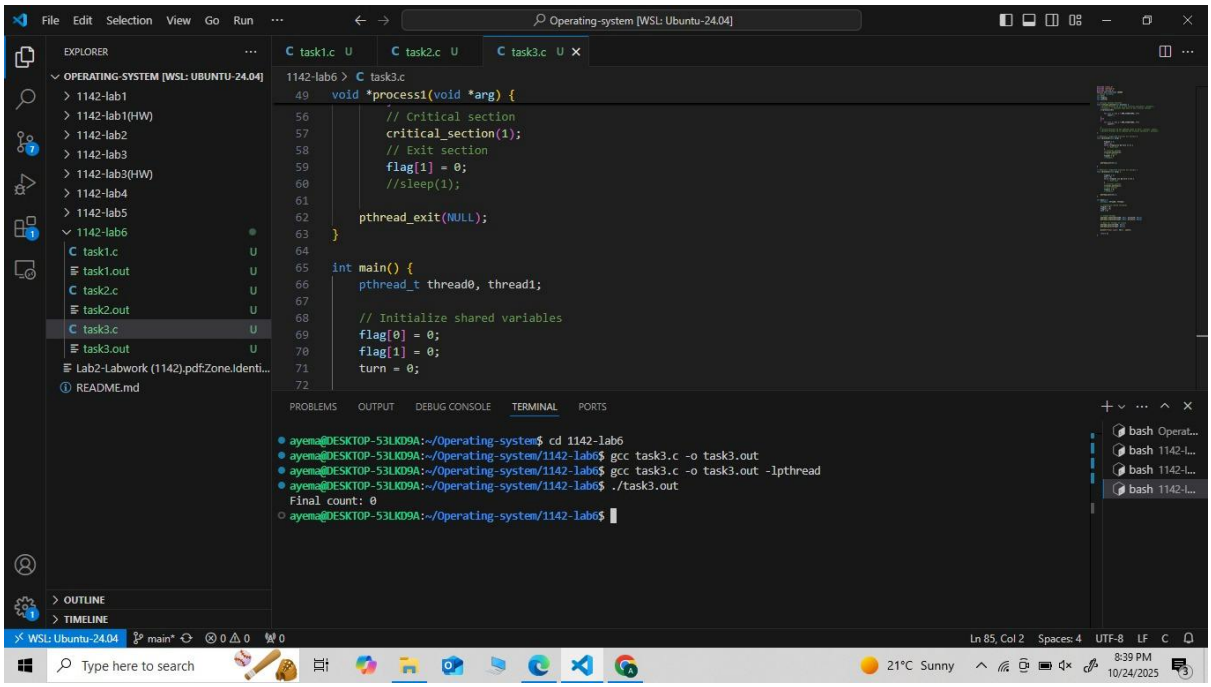
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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;
}
```


File Edit Selection View Go Run
Operating-system [WSL: Ubuntu-24.04]
EXPLORER
OPERATING-SYSTEM [WSL: UBUNTU-24.04]
1142-lab1
1142-lab1(HW)
1142-lab2
1142-lab3
1142-lab3(HW)
1142-lab4
1142-lab5
1142-lab6
task1.c
task1.out
task2.c
task2.out
Lab2-Labwork (1142).pdf:Zone.Identi...
README.md
task1.c U
task2.c U
1142-lab6 > task2.c
void *process1(void *arg) {
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);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ayema@DESKTOP-53LKD9A:~/Operating-system$ cd 1142-lab6
ayema@DESKTOP-53LKD9A:~/Operating-system/1142-lab6$ gcc task2.c -o task2.out -lpthread
ayema@DESKTOP-53LKD9A:~/Operating-system/1142-lab6$ ./task2.out
Final count: 197961
ayema@DESKTOP-53LKD9A:~/Operating-system/1142-lab6$
bash Operat...
bash 1142-l...
bash 1142-l...
OUTLINE
TIMELINE
WSL: Ubuntu-24.04
main*
Ln 51, Col 17
Spaces: 4
UTF-8
LF
C
Type here to search
19°C Sunny
8:36 PM
10/24/2025

TASK 3:

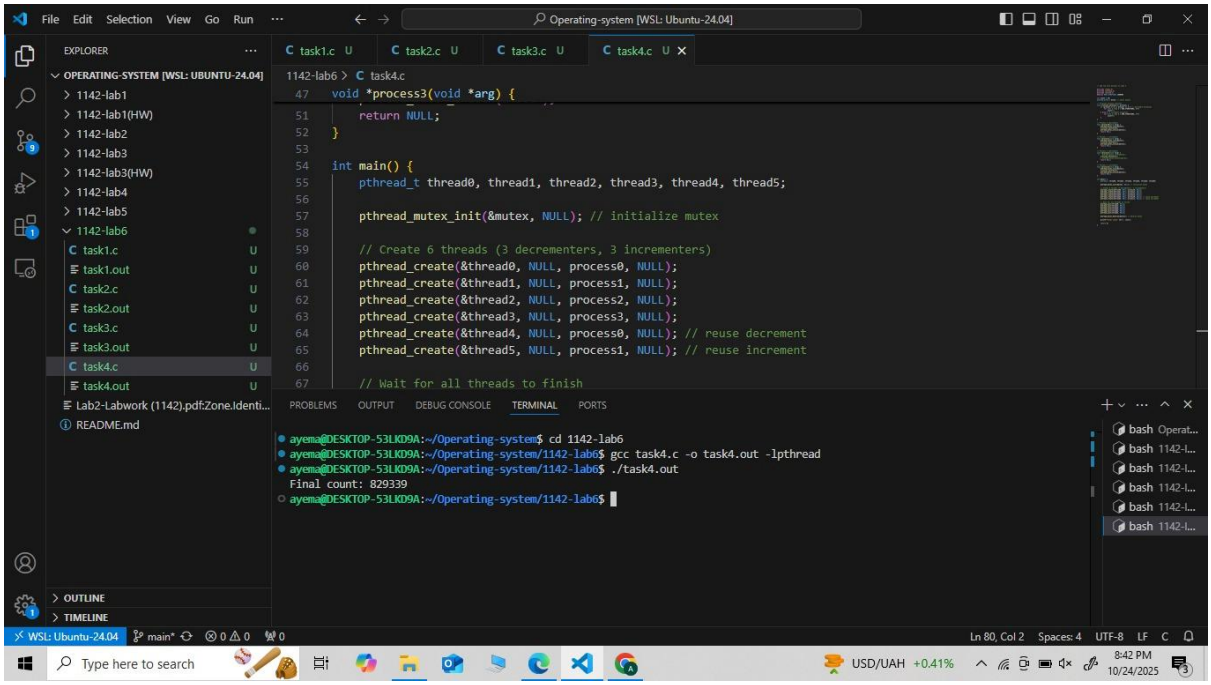
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 100000
// Shared variables
int turn;
int flag[2];
int count=0;

// 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;
}
```

5



TASK 4:

```c
// add two more process in code 4

#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 || process == 2) { // decrement processes
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count--;
    } else { // increment processes
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count++;
    }
}

// Process 0 (decrement)
void *process0(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(0);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 1 (increment)
void *process1(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(1);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 2 (decrement)
void *process2(void *arg) {
 //   pthread_mutex_lock(&mutex);
    critical_section(2);
   // pthread_mutex_unlock(&mutex);
    return NULL;
}

// Process 3 (increment)
void *process3(void *arg) {
    pthread_mutex_lock(&mutex);
    critical_section(3);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main() {
    pthread_t thread0, thread1, thread2, thread3, thread4, thread5;

    pthread_mutex_init(&mutex, NULL); // initialize mutex

    // Create 6 threads (3 decrementers, 3 incrementers)
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process2, NULL);
    pthread_create(&thread3, NULL, process3, NULL);
    pthread_create(&thread4, NULL, process0, NULL); // reuse decrement
    pthread_create(&thread5, NULL, process1, NULL); // reuse increment

    // Wait for all threads to finish
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);
    pthread_join(thread5, NULL);

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```



PETERSON VS MUTEX:

Feature	Peterson's	Mutex
Type	process synchronization	OS-level synchronization primitive
Processes	2 only	Multiple
Support	Theoretical	Practical
Busy waiting	Yes	Usually No
Hardware/OS support	No	Yes
Use case	Learning / research	Real-world systems

- **Both Peterson's Algorithm and Mutex are used for achieving mutual exclusion in operating systems.**
- **Peterson's Algorithm is a software-based method that uses shared variables like flag and turn to control access.**

- **It works only for two processes and relies on busy waiting, where the CPU keeps checking repeatedly.**
- **Mutex is an operating system-supported lock that can handle multiple processes or threads efficiently.**
- **It uses system calls such as lock() and unlock() to manage access to the critical section.**
- **Unlike Peterson's Algorithm, Mutex blocks a process when the resource is busy instead of wasting CPU time.**
- **Peterson's Algorithm is mainly theoretical and used for understanding synchronization concepts.**
- **Mutex is a practical synchronization tool commonly used in real-world operating systems and applications.**