National Textile University, Faisalabad

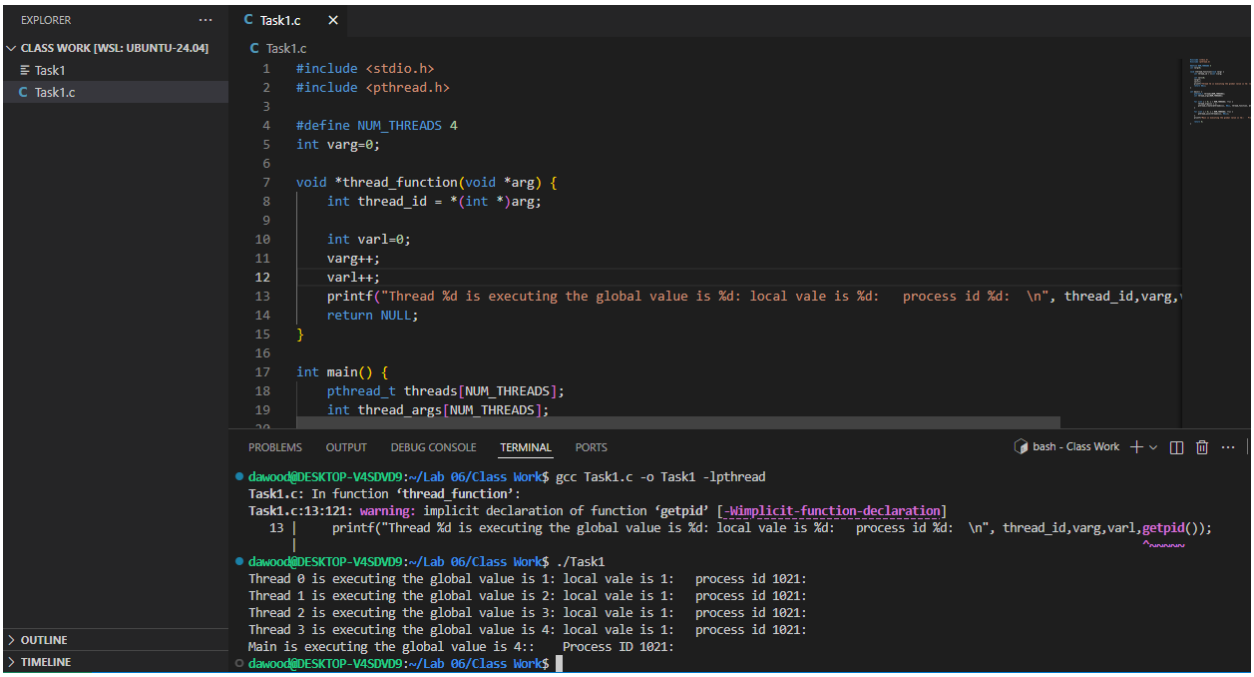
Department of Computer Science

Name	Dawood Saif
Class	SE-5th (A)
Reg. No.	23-NTU-CS-1145
Course	Operating system
Submitted To	Sir Nasir Mehmod
Submission Date	24/10/2025
Lab No.	6 (Class Work)

Lab No. 6 Operating system:

Program 1:

It is a basic program, like we did in the previous labs.



Program 2:

It gives different solutions on running code every time

Task1.c | Task2.c | Task2-Sol.c | Task3-mutx.c | Task3-mutex-modify.c

Task2.c

```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--;
    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task2.c -o Task2 -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task2
Final count: 1019168
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task2.c -o Task2 -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task2
Final count: -92786
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task2.c -o Task2 -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task2
Final count: -604789
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$
```

Program 2 (Solution using Petrison Algorithm)

1ST Method using while loop which is good for just 2 processes

Task1.c | Task2.c | Task2-Sol | Task2-Sol.c | Task3-mutx.c

Task2-Sol.c

```c
#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--;
    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task2-Sol.c -o Task2-Sol -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task2-Sol
Final count: 0
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task2-Sol.c -o Task2-Sol -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task2-Sol
Final count: 0
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$
```

Program 3 (2nd Method-mutex for multiple Processes)

This method contains 5 main steps

- First, we make object named mutex
- Then we initialize it in main function
- Thirdly, we lock it in process
- Then it unlocks it
- At Last, we destroy mutex in main function

Task3-mutx.c × Task3-mutex-modify.c

Task3-mutx.c

```c
void critical_section(int process) {
    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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task3-mutx.c -o Task3-mutx -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task3-mutx
Final count: 10
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$
```

Modification in program 3 (created 4 processes)

Task3-mutx.c Task3-mutex-modify.c

Task3-mutex-modify.c

```c
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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task3-mutex-modify.c -o Task3-mutex-modify -lpthread
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task3-mutex-modify
Final count: 10
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$
```

Differentiation between two

Peterson's Algorithm	Mutex Method
Works only for two processes (hard to extend to many).	Works for multiple processes or threads easily.
Uses busy waiting (keeps CPU working while waiting).	Usually blocks or sleeps the process until it can enter the critical section.
Programmer must manually control entry and exit logic.	The system automatically manages lock and unlock operations.
Helps to understand how mutual exclusion works conceptually.	Provides a practical, efficient, and safe way to ensure mutual exclusion.