

# National Textile University, Faisalabad



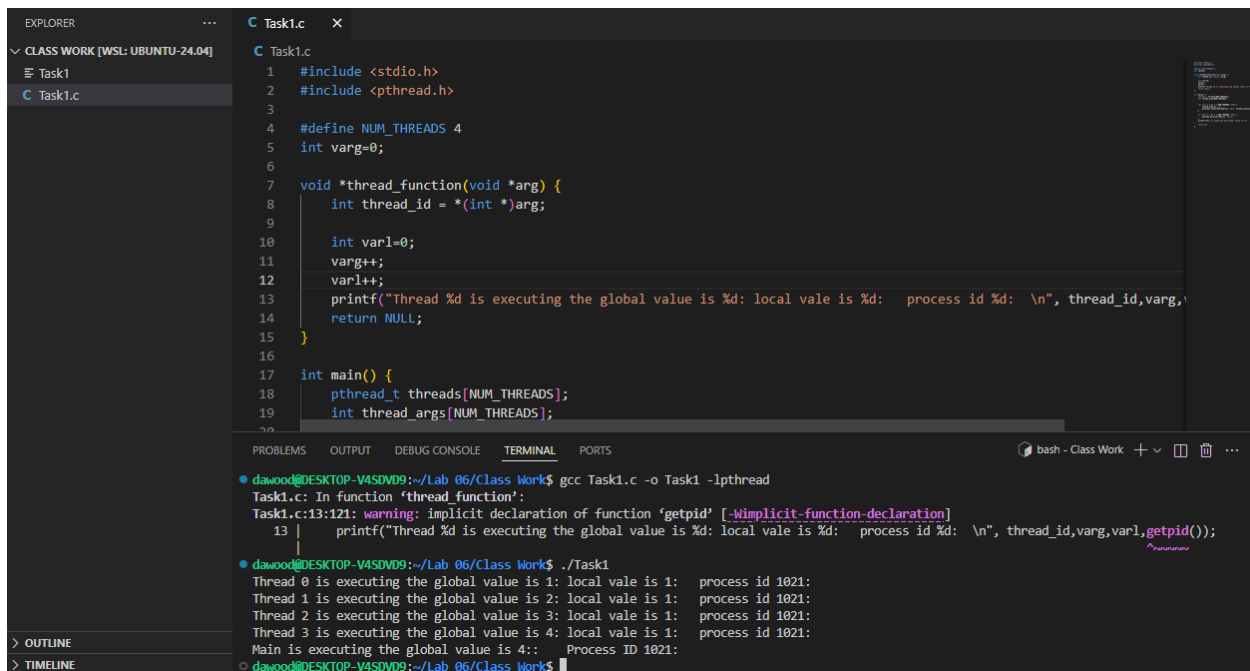
## Department of Computer Science

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

## Lab No. 6 Operating system:

### Program 1:

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



The screenshot displays a code editor with a file named `Task1.c` and a terminal window below it.

**Task1.c Code:**

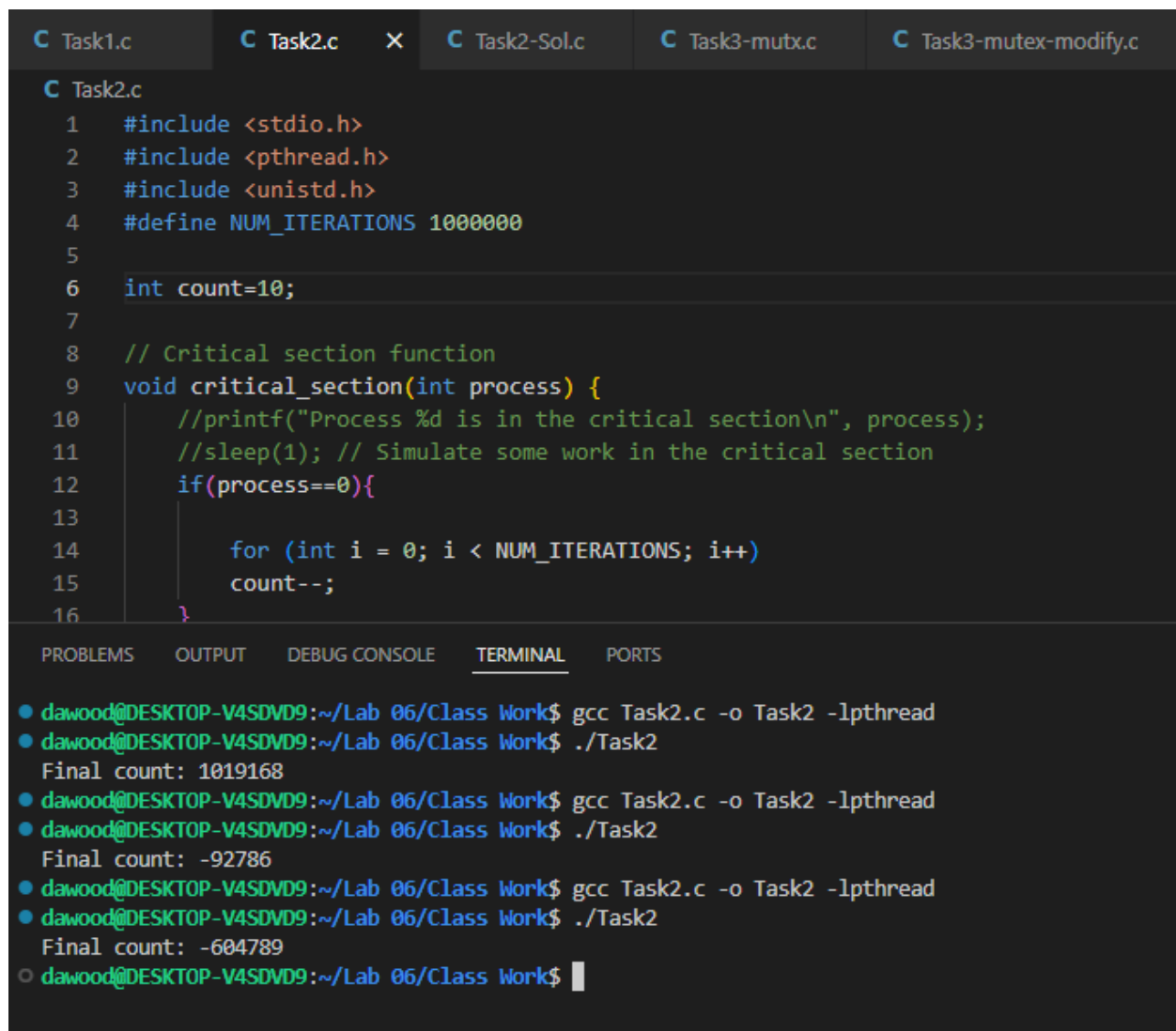
```
1 #include <stdio.h>
2 #include <pthread.h>
3
4 #define NUM_THREADS 4
5 int var=0;
6
7 void *thread_function(void *arg) {
8     int thread_id = *(int *)arg;
9
10    int var1=0;
11    var++;
12    var1++;
13    printf("Thread %d is executing the global value is %d: local vale is %d:  process id %d:  \n", thread_id,varg,
14    return NULL;
15 }
16
17 int main() {
18     pthread_t threads[NUM_THREADS];
19     int thread_args[NUM_THREADS];
20 }
```

**Terminal Output:**

```
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ gcc Task1.c -o Task1 -lpthread
Task1.c: In function 'thread_function':
Task1.c:13:121: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
13 |     printf("Thread %d is executing the global value is %d: local vale is %d:  process id %d:  \n", thread_id,varg,var1,getpid());
    |                                                                                                     ^
dawood@DESKTOP-V4SDVD9:~/Lab 06/Class Work$ ./Task1
Thread 0 is executing the global value is 1: local vale is 1:  process id 1021:
Thread 1 is executing the global value is 2: local vale is 1:  process id 1021:
Thread 2 is executing the global value is 3: local vale is 1:  process id 1021:
Thread 3 is executing the global value is 4: local vale is 1:  process id 1021:
Main is executing the global value is 4::  Process ID 1021:
```

## Program 2:

It gives different solutions on running code every time



The image shows a code editor with five tabs: Task1.c, Task2.c, Task2-Sol.c, Task3-mutx.c, and Task3-mutex-modify.c. The active tab is Task2.c, which contains the following C code:

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #define NUM_ITERATIONS 1000000
5
6  int count=10;
7
8  // Critical section function
9  void critical_section(int process) {
10     //printf("Process %d is in the critical section\n", process);
11     //sleep(1); // Simulate some work in the critical section
12     if(process==0){
13
14         for (int i = 0; i < NUM_ITERATIONS; i++)
15             count--;
16     }
```

Below the code editor is a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing the following commands and output:

```
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)**

1<sup>ST</sup> Method using while loop which is good for just 2 processes

C Task1.cC Task2.c≡ Task2-SolC Task2-Sol.c X C Task3-mutex.c

C Task2-Sol.c

```
3  #include <unistd.h>
4  #define NUM_ITERATIONS 100000
5  // Shared variables
6  int turn;
7  int flag[2];
8  int count=0;
9
10 // Critical section function
11 void critical_section(int process) {
12     //printf("Process %d is in the critical section\n", process);
13     //sleep(1); // Simulate some work in the critical section
14     if(process==0){
15
16         for (int i = 0; i < NUM_ITERATIONS; i++)
17             count--;
18     }
19 }
```

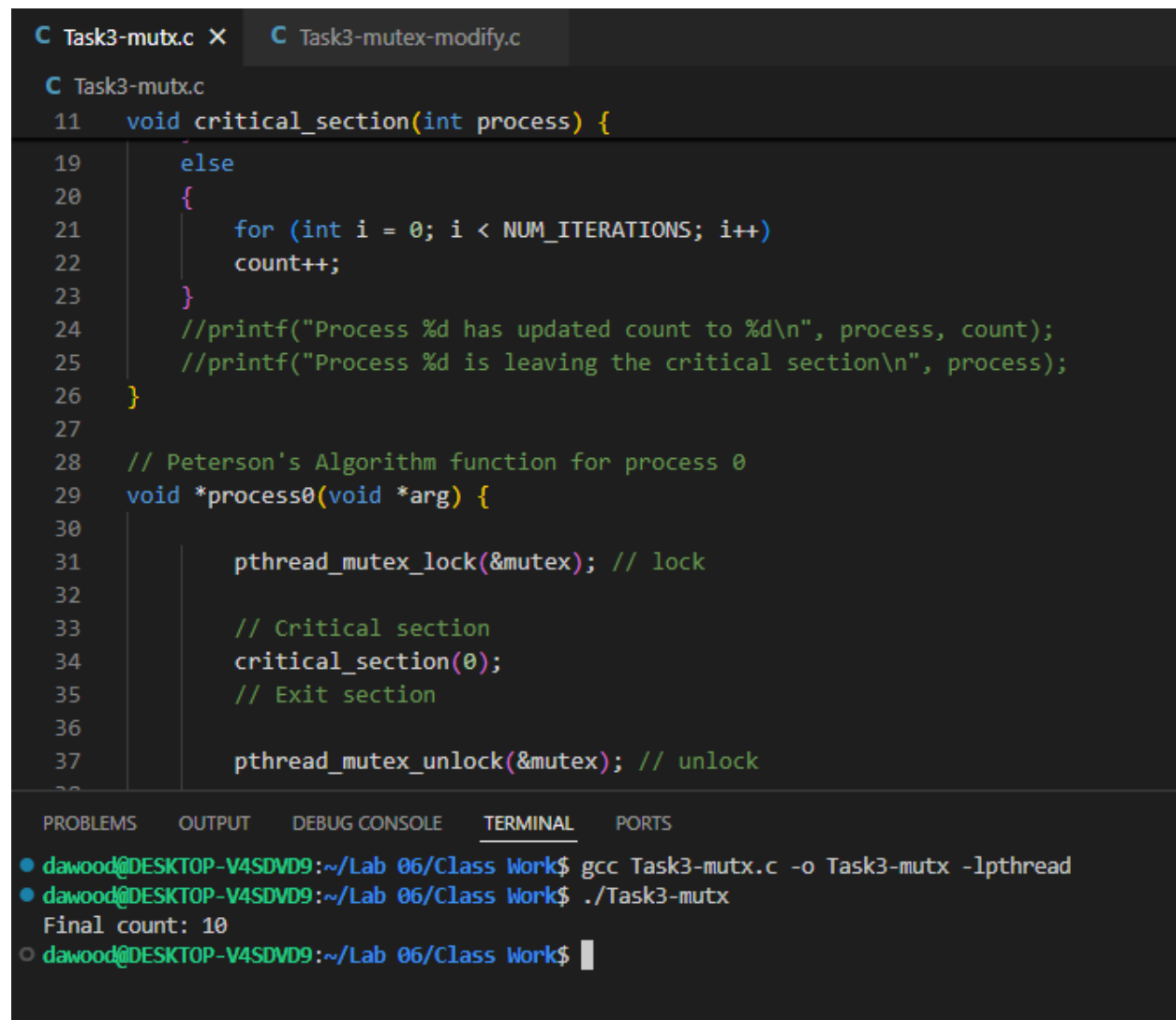
PROBLEMSOUTPUTDEBUG CONSOLETERMINALPORTS

```
• 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 (2<sup>nd</sup> 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

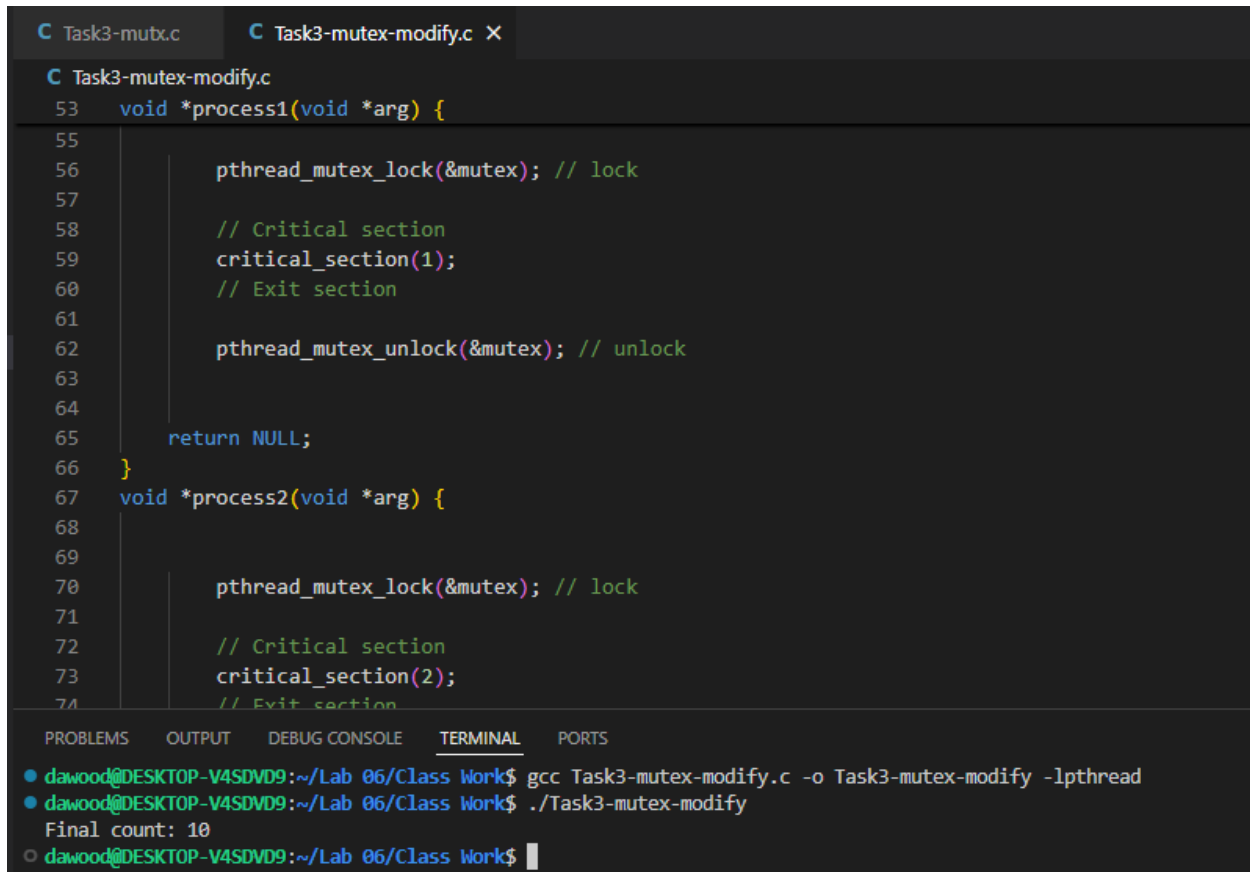


```
C Task3-mutex.c X C Task3-mutex-modify.c
C Task3-mutex.c
11 void critical_section(int process) {
19     else
20     {
21         for (int i = 0; i < NUM_ITERATIONS; i++)
22             count++;
23     }
24     //printf("Process %d has updated count to %d\n", process, count);
25     //printf("Process %d is leaving the critical section\n", process);
26 }
27
28 // Peterson's Algorithm function for process 0
29 void *process0(void *arg) {
30
31     pthread_mutex_lock(&mutex); // lock
32
33     // Critical section
34     critical_section(0);
35     // Exit section
36
37     pthread_mutex_unlock(&mutex); // unlock
38 }
39
40 int main() {
41     pthread_t t1;
42     pthread_create(&t1, NULL, process0, NULL);
43     pthread_join(t1, NULL);
44     printf("Final count: %d\n", count);
45     return 0;
46 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

## Modification in program 3 (created 4 processes)



```
C Task3-mutex.c  C Task3-mutex-modify.c X
C Task3-mutex-modify.c
53 void *process1(void *arg) {
55     pthread_mutex_lock(&mutex); // lock
56
57     // Critical section
58     critical_section(1);
59     // Exit section
60
61     pthread_mutex_unlock(&mutex); // unlock
62
63
64
65     return NULL;
66 }
67 void *process2(void *arg) {
68
69
70     pthread_mutex_lock(&mutex); // lock
71
72     // Critical section
73     critical_section(2);
74     // Exit section
75
76     pthread_mutex_unlock(&mutex); // unlock
77
78
79
80     return NULL;
81 }
82
83 int main() {
84     pthread_t t1, t2, t3, t4;
85     void *arg1, *arg2, *arg3, *arg4;
86     arg1 = (void *)1;
87     arg2 = (void *)2;
88     arg3 = (void *)3;
89     arg4 = (void *)4;
90     pthread_create(&t1, NULL, process1, arg1);
91     pthread_create(&t2, NULL, process1, arg2);
92     pthread_create(&t3, NULL, process2, arg3);
93     pthread_create(&t4, NULL, process2, arg4);
94     pthread_join(t1, NULL);
95     pthread_join(t2, NULL);
96     pthread_join(t3, NULL);
97     pthread_join(t4, NULL);
98     printf("Final count: %d\n", count);
99     return 0;
100 }
```

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.