



# National Textile University

## Department of Computer Science

**Subject:**

Operating System

**Submitted to:**

Sir Naseer

**Submitted by:**

Ayema

**Reg number:**

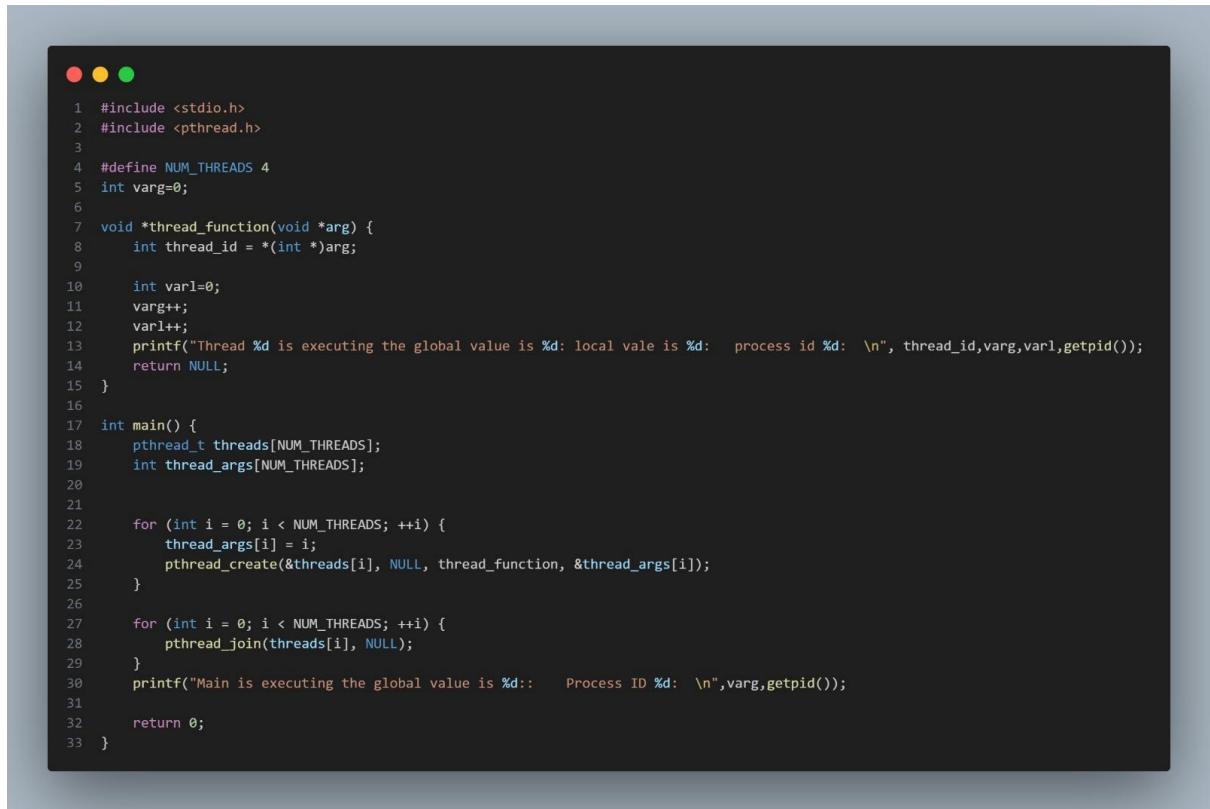
23-NTU-CS-1142

**Lab no. :**

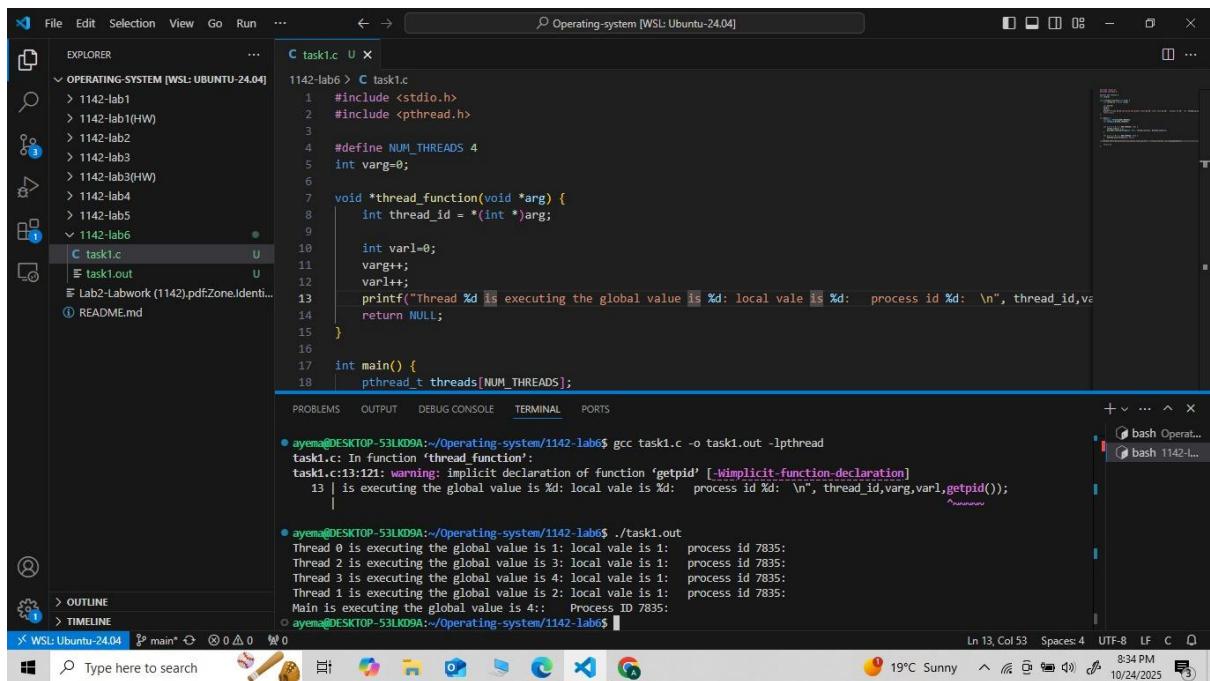
05

**Semester:** 5Th

## Task 1:



```
1 #include <stdio.h>
2 #include <pthread.h>
3
4 #define NUM_THREADS 4
5 int varg=0;
6
7 void *thread_function(void *arg) {
8     int thread_id = *(int *)arg;
9
10    int varl=0;
11    varg++;
12    varl++;
13    printf("Thread %d is executing the global value is %d: local vale is %d: process id %d: \n", thread_id,varg,varl,getpid());
14    return NULL;
15 }
16
17 int main() {
18     pthread_t threads[NUM_THREADS];
19     int thread_args[NUM_THREADS];
20
21
22     for (int i = 0; i < NUM_THREADS; ++i) {
23         thread_args[i] = i;
24         pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
25     }
26
27     for (int i = 0; i < NUM_THREADS; ++i) {
28         pthread_join(threads[i], NULL);
29     }
30     printf("Main is executing the global value is %d: Process ID %d: \n",varg,getpid());
31
32     return 0;
33 }
```



The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a project structure under "OPERATING-SYSTEM [WSL: UBUNTU-24.04]". It includes files: task1.c, task1.out, and README.md.
- Code Editor:** The current file is task1.c, which contains the provided C code for multi-threading.
- Terminal:** The terminal window shows the build command and its output:

```
ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ gcc task1.c -o task1.out
task1.c: In function 'thread function':
task1.c:13:12: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
  13 |     is executing the global value is %d: local vale is %d: process id %d: \n", thread_id,varg,varl,getpid());
               |                                         ^
ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ ./task1.out
```
- Status Bar:** Shows the terminal path as "ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6\$". It also displays system information like temperature (19°C), battery status, and date/time (10/24/2025).

## Task 2:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 1000000
5
6 int count=10;
7
8
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     else
20     {
21         for (int i = 0; i < NUM_ITERATIONS; i++)
22             count++;
23     }
24 }
25
26 void *process0(void *arg) {
27
28
29
30
31     // Critical section
32     critical_section(0);
33     // Exit section
34
35
36
37     return NULL;
38 }
39
40 void *process1(void *arg) {
41
42
43
44     // Critical section
45     critical_section(1);
46     // Exit section
47
48
49
50
51     return NULL;
52 }
53
54 int main() {
55     pthread_t thread0, thread1, thread2, thread3;
56
57
58     // Create threads
59     pthread_create(&thread0, NULL, process0, NULL);
60     pthread_create(&thread1, NULL, process1, NULL);
61     pthread_create(&thread2, NULL, process0, NULL);
62     pthread_create(&thread3, NULL, process1, NULL);
63
64     // Wait for threads to finish
65     pthread_join(thread0, NULL);
66     pthread_join(thread1, NULL);
67     pthread_join(thread2, NULL);
68     pthread_join(thread3, NULL);
69
70
71     printf("Final count: %d\n", count);
72
73     return 0;
74 }
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running in a WSL Ubuntu 24.04 environment. The main area displays the file `task2.c` with the following code:

```
1142-lab6 > C task2.c
40 void *process1(void *arg) {
41     critical_section(1);
42     // Exit section
43
44     return NULL;
45 }
46
47
48
49
50
51
52 }
53
54 int main() {
55     pthread_t thread0, thread1, thread2, thread3;
56
57     // Create threads
58     pthread_create(&thread0, NULL, process0, NULL);
59     pthread_create(&thread1, NULL, process1, NULL);
60     pthread_create(&thread2, NULL, process0, NULL);
61 }
```

The terminal tab at the bottom shows the following command-line session:

```
● ayem@DESKTOP-53LK09A:~/Operating-system$ cd 1142-lab6
● ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ gcc task2.c -o task2.out -lpthread
● ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ ./task2.out
Final count: 197961
● ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$
```

## TASK 3:

```

1 //include <stdio.h>
2 #include <pthread.h>
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         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18     else{
19         for (int i = 0; i < NUM_ITERATIONS; i++)
20             count++;
21     }
22 }
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     flag[0] = 1;
32     turn = 1;
33     while ((flag[1]==1 && turn == 1) ||
34            // Busy wait
35            )
36         // Critical section
37         critical_section(0);
38         // Exit section
39         flag[0] = 0;
40         //sleep(1);
41
42
43     pthread_exit(NULL);
44 }
45
46 }
47
48 // Peterson's Algorithm function for process 1
49 void *process1(void *arg) {
50
51     flag[1] = 1;
52     turn = 0;
53     while ((flag[0]==1 && turn == 0) ||
54            // Busy wait
55            )
56         // Critical section
57         critical_section(1);
58         // Exit section
59         flag[1] = 0;
60         //sleep(1);
61
62     pthread_exit(NULL);
63 }
64
65 int main() {
66     pthread_t thread0, thread1;
67
68     // Initialize shared variables
69     flag[0] = 0;
70     flag[1] = 0;
71     turn = 0;
72
73     // Create threads
74     pthread_create(&thread0, NULL, process0, NULL);
75     pthread_create(&thread1, NULL, process1, NULL);
76
77     // Wait for threads to finish
78     pthread_join(thread0, NULL);
79     pthread_join(thread1, NULL);
80
81     printf("Final count: %d\n", count);
82
83     return 0;
84 }
85 }
```

5

```

File Edit Selection View Go Run ...
← → ⌂ Operating-system [WSL: Ubuntu-24.04]
EXPLORER ... C task1.c U C task2.c U C task3.c U X
OPERATING-SYSTEM [WSL: UBUNTU-24.04]
> 1142-lab1
> 1142-lab1(HW)
> 1142-lab2
> 1142-lab3
> 1142-lab3(HW)
> 1142-lab4
> 1142-lab5
> 1142-lab6
C task1.c U
task1.out U
C task2.c U
task2.out U
C task3.c U
task3.out U
Lab2-Labwork (1142).pdf Zone.Identifier
README.md

task1.c
task2.c
task3.c
task3.out
Lab2-Labwork (1142).pdf Zone.Identifier
README.md

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ayena@DESKTOP-53LK09A:~/Operating-system$ cd 1142-lab6
ayena@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ gcc task3.c -o task3.out
ayena@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ ./task3.out
Final count: 0
ayena@DESKTOP-53LK09A:~/Operating-system/1142-lab6$
```

## TASK 4:

```
1 // add two more process in code 4
2
3 #include <stdio.h>
4 #include <pthread.h>
5 #include <unistd.h>
6 #define NUM_ITERATIONS 1000000
7
8 int count = 10;
9 pthread_mutex_t mutex; // mutex object
10
11 // Critical section function
12 void critical_section(int process) {
13     if (process == 0 || process == 2) { // decrement processes
14         for (int i = 0; i < NUM_ITERATIONS; i++)
15             count--;
16     } else { // increment processes
17         for (int i = 0; i < NUM_ITERATIONS; i++)
18             count++;
19     }
20 }
21
22 // Process 0 (decrement)
23 void *process0(void *arg) {
24     pthread_mutex_lock(&mutex);
25     critical_section(0);
26     pthread_mutex_unlock(&mutex);
27     return NULL;
28 }
29
30 // Process 1 (increment)
31 void *process1(void *arg) {
32     pthread_mutex_lock(&mutex);
33     critical_section(1);
34     pthread_mutex_unlock(&mutex);
35     return NULL;
36 }
37
38 // Process 2 (decrement)
39 void *process2(void *arg) {
40     // pthread_mutex_lock(&mutex);
41     critical_section(2);
42     // pthread_mutex_unlock(&mutex);
43     return NULL;
44 }
45
46 // Process 3 (increment)
47 void *process3(void *arg) {
48     pthread_mutex_lock(&mutex);
49     critical_section(3);
50     pthread_mutex_unlock(&mutex);
51     return NULL;
52 }
53
54 int main() {
55     pthread_t thread0, thread1, thread2, thread3, thread4, thread5;
56
57     pthread_mutex_init(&mutex, NULL); // initialize mutex
58
59     // Create 6 threads (3 decreasers, 3 increasers)
60     pthread_create(&thread0, NULL, process0, NULL);
61     pthread_create(&thread1, NULL, process1, NULL);
62     pthread_create(&thread2, NULL, process2, NULL);
63     pthread_create(&thread3, NULL, process3, NULL);
64     pthread_create(&thread4, NULL, process0, NULL); // reuse decrement
65     pthread_create(&thread5, NULL, process1, NULL); // reuse increment
66
67     // Wait for all threads to finish
68     pthread_join(thread0, NULL);
69     pthread_join(thread1, NULL);
70     pthread_join(thread2, NULL);
71     pthread_join(thread3, NULL);
72     pthread_join(thread4, NULL);
73     pthread_join(thread5, NULL);
74
75     pthread_mutex_destroy(&mutex); // destroy mutex
76
77     printf("Final count: %d\n", count);
78
79     return 0;
80 }
```

```

File Edit Selection View Go Run ... ← → ⌂ Operating-system [WSL: Ubuntu-24.04]
EXPLORER ... C task1.c U C task2.c U C task3.c U C task4.c U X
OPERATING-SYSTEM [WSL: UBUNTU-24.04]
> 1142-lab1
> 1142-lab1(HW)
> 1142-lab2
> 1142-lab3
> 1142-lab3(HW)
> 1142-lab4
> 1142-lab5
1142-lab6
C task1.c U
task1.out U
C task2.c U
task2.out U
C task3.c U
task3.out U
C task4.c U
task4.out U
Lab2-Labwork (1142).pdf Zone.Identifier README.md
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ayem@DESKTOP-53LK09A:~/Operating-system$ cd 1142-lab6
ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ gcc task4.c -o task4.out -lpthread
ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ ./task4.out
Final count: 82939
ayem@DESKTOP-53LK09A:~/Operating-system/1142-lab6$ 
+ v ... ^ x
bash Operat...
bash 1142-l...
bash 1142-l...
bash 1142-l...
bash 1142-l...
Ln 80, Col 2 Spaces: 4 UTF-8 LF C
Type here to search 8:42 PM 10/24/2025 (3)

```

## 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.