



National Textile University
Department of Computer Science

Subject

Operating system

Submitted to:

Sir Nasir Mehmood

Submitted by:

Hussain Naweed

Registration Number

23-NTU-CS-1165

Lab No.

06

Semester

5th

Program1

The image shows a Visual Studio Code editor window with a C program named `program1.c` open. The program is a multi-threaded application that creates four threads, each with a unique global value (1, 2, 3, 4) and a local value (1). The main thread prints the global value for each thread and the process ID (18392). The output in the terminal shows the execution of the program, with the main thread printing the global value for each thread and the process ID.

```
1 // program1.c
2 #include <stdio.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 #define NUM_THREADS 4
7
8 pthread_t threads[NUM_THREADS];
9 int global_val = 1;
10 int local_val = 1;
11
12 void* thread_function(void* arg) {
13     printf("Thread %d is executing the global value is %d: local vale is %d: process id %d: \n",
14            thread_id, varg, varl, getpid());
15     return NULL;
16 }
17
18 int main() {
19     pthread_args_t args[NUM_THREADS];
20     pthread_create(&threads[i], NULL, thread_function, &args[i]);
21 }
22
23 for (int i = 0; i < NUM_THREADS; ++i) {
24     pthread_join(threads[i], NULL);
25 }
26
27 printf("Main is executing the global value is %d::\n", global_val);
28
29 return 0;
30 }
```

Terminal Output:

```
hussain@DESKTOP-NNNR4M1:~/OS/Lab6$ ./program1
Thread 0 is executing the global value is 1: local vale is 1: process id 18392:
Thread 1 is executing the global value is 2: local vale is 1: process id 18392:
Thread 2 is executing the global value is 3: local vale is 1: process id 18392:
Thread 3 is executing the global value is 4: local vale is 1: process id 18392:
Main is executing the global value is 4:: Process ID 18392:
hussain@DESKTOP-NNNR4M1:~/OS/Lab6$
```

Program 2

File Edit Selection View Go Run ...

Lab6 [WSL: Ubuntu-24.04]

EXPLORER

LAB6 [WSL: UBUNTU-24.04]

program1

program1.c

program2

program2.c

program3.c

program4_mutex.c

Task1_MultipleProcess.c

OUTLINE

TIMELINE

program1.c

program2.c

program2.c > ...

```
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++){
```

Loading...

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

bash - Lab6

hussain@DESKTOP-NNNR4M3:~/OS/Lab6\$ gcc program2.c -o program2 -lpthread

hussain@DESKTOP-NNNR4M3:~/OS/Lab6\$./program2

Final count: 33301

hussain@DESKTOP-NNNR4M3:~/OS/Lab6\$./program2

Final count: -472386

hussain@DESKTOP-NNNR4M3:~/OS/Lab6\$./program2

Final count: -878723

hussain@DESKTOP-NNNR4M3:~/OS/Lab6\$

CHAT

Ask about your code

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

program2.c

Add context (#), extensions (@), commands

Ask

WSL: Ubuntu-24.04

0 0 0 0

Type here to search

21°C

9:57 AM

10/24/2025

Program3:

The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the file structure of a project named 'LAB6 [WSL: UBUNTU-24.04]', including files like 'program1', 'program1.c', 'program2', 'program2.c', 'program3', 'program3.c', 'program4_mutex.c', and 'Task1_MultipleProcess.c'. The main editor window is open to 'program3.c', which contains the following C code:

```
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             // Critical section logic
17         }
18     }
19 }
```

Below the code editor, the TERMINAL panel shows the execution of the program. The commands and their outputs are as follows:

```
hussain@DESKTOP-NNNR4M0:~/OS/Lab6$ gcc program3.c -o program3 -lpthread
hussain@DESKTOP-NNNR4M0:~/OS/Lab6$ ./program3
Final count: 0
hussain@DESKTOP-NNNR4M0:~/OS/Lab6$ ./program3
Final count: 0
hussain@DESKTOP-NNNR4M0:~/OS/Lab6$ ./program3
Final count: 0
hussain@DESKTOP-NNNR4M0:~/OS/Lab6$
```

On the right side of the editor, there is a CHAT panel with the heading 'Ask about your code'. It includes a note that 'AI responses may be inaccurate' and a link to 'Generate Agent Instructions' to onboard AI onto the codebase. Below this, there is a search bar with the text 'Add context (#), extensions (@), commands' and an 'Ask' button.

Program4:

Visual Studio Code interface showing the implementation of a mutex program in C.

Explorer: LAB6 [WSL: UBUNTU-24.04]

- program1
- program1.c
- program2
- program2.c
- program3
- program3.c
- program4_mutex
- program4_mutex.c
- Task1_MultipleProcess.c

Code Editor: C program4_mutex.c

```
69 // Wait for threads to finish
70 pthread_join(thread0, NULL);
71 pthread_join(thread1, NULL);
72 pthread_join(thread2, NULL);
73 pthread_join(thread3, NULL);
74
75 pthread_mutex_destroy(&mutex); // destroy mutex
76
77 printf("Final count: %d\n", count);
78
79 return 0;
80 }
```

Terminal: bash - Lab6

```
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./program3
Final count: 0
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ gcc program4_mutex.c -o program4_mutex -lpthread
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./program4_mutex
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./program4_mutex
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./program4_mutex
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./program4_mutex
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$
```

Chat: Ask about your code. AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.

task1:

```
File Edit Selection View Go Run ... Lab6 [WSL: Ubuntu-24.04]
```

EXPLORER

- LAB6 [WSL: UBUNTU-24.04]
 - program1
 - program1.c
 - program2
 - program2.c
 - program3
 - program3.c
 - program4_mutex
 - program4_mutex.c
 - Task1_MultipleProcess
 - Task1_MultipleProcess.c

Task1_MultipleProcess.c

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 1000000
5
6 int count=10;
7
8 pthread_mutex_t mutex; // mutex object
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++)
```

TERMINAL

```
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ gcc Task1_MultipleProcess.c -o Task1_MultipleProcess -lpthread
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./Task1_MultipleProcess
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./Task1_MultipleProcess
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$ ./Task1_MultipleProcess
Final count: 10
hussain@DESKTOP-NNNR4M3:~/OS/Lab6$
```

Chat

Ask about your code

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Task1_MultipleProcess.c

Add context (#), extensions (@), commands

Ask

WSL: Ubuntu-24.04

Type here to search

20°C

10:47 AM 10/24/2025

Output:

Analysis of Peterson Algorithms and Mutex Based Code:

Peterson's Algorithm:

Mutex Based Code

| | |
|---|--|
| <ul style="list-style-type: none">✚ Uses two shared variables: flag[] (interest) and turn (whose turn it is) to enforce mutual exclusion. | <ul style="list-style-type: none">✚ Uses pthread_mutex_lock() and pthread_mutex_unlock() — system calls that block or wake threads safely. |
| <ul style="list-style-type: none">✚ Only supports 2 processes (threads). Hard to extend to 3+ safely. | <ul style="list-style-type: none">✚ Supports any number of threads/processes with minimal code change. |
| <ul style="list-style-type: none">✚ Controlled by flag and turn variables manually coded. | <ul style="list-style-type: none">✚ Controlled by pthread_mutex_lock() internally in the OS. |
| <ul style="list-style-type: none">✚ Must manually reset flags and turn. | <ul style="list-style-type: none">✚ Simply call pthread_mutex_unlock(). |
| <ul style="list-style-type: none">✚ Works only with strict memory ordering may fail on modern multi-core systems. | <ul style="list-style-type: none">✚ Uses kernel-level atomic operations and hardware-supported locking. |