



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Ghulam Mohyuddin

Reg number:

23-NTU-CS-1158

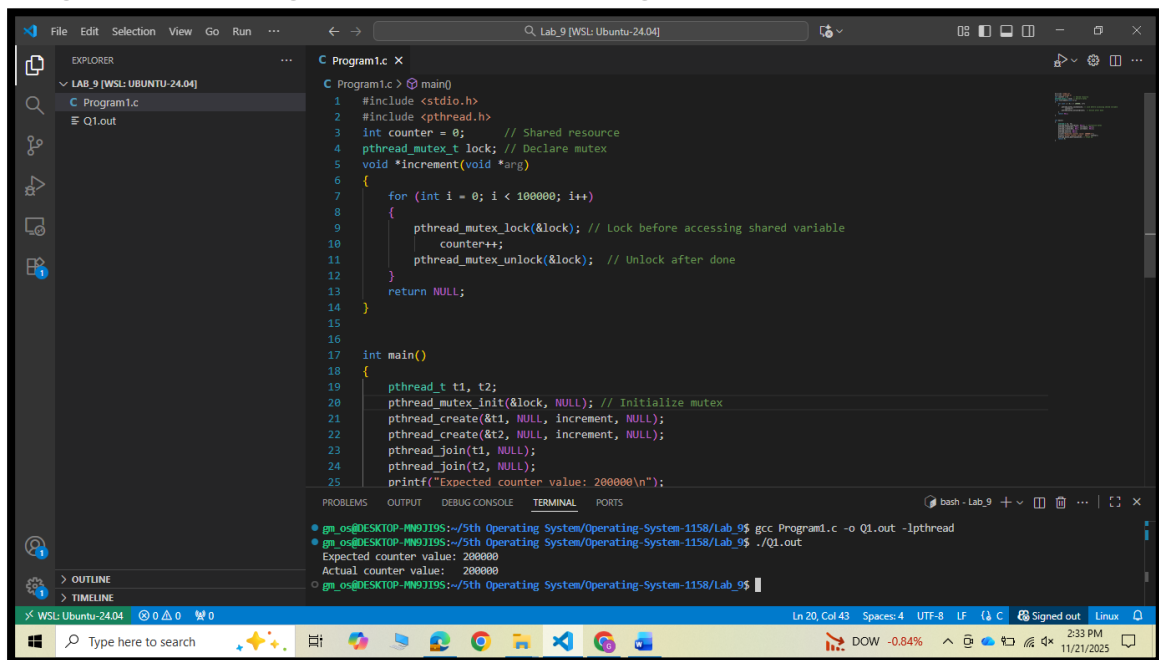
Lab number:

9th

Semester:

5th

Program#1: Fixing Race Condition using Mutex

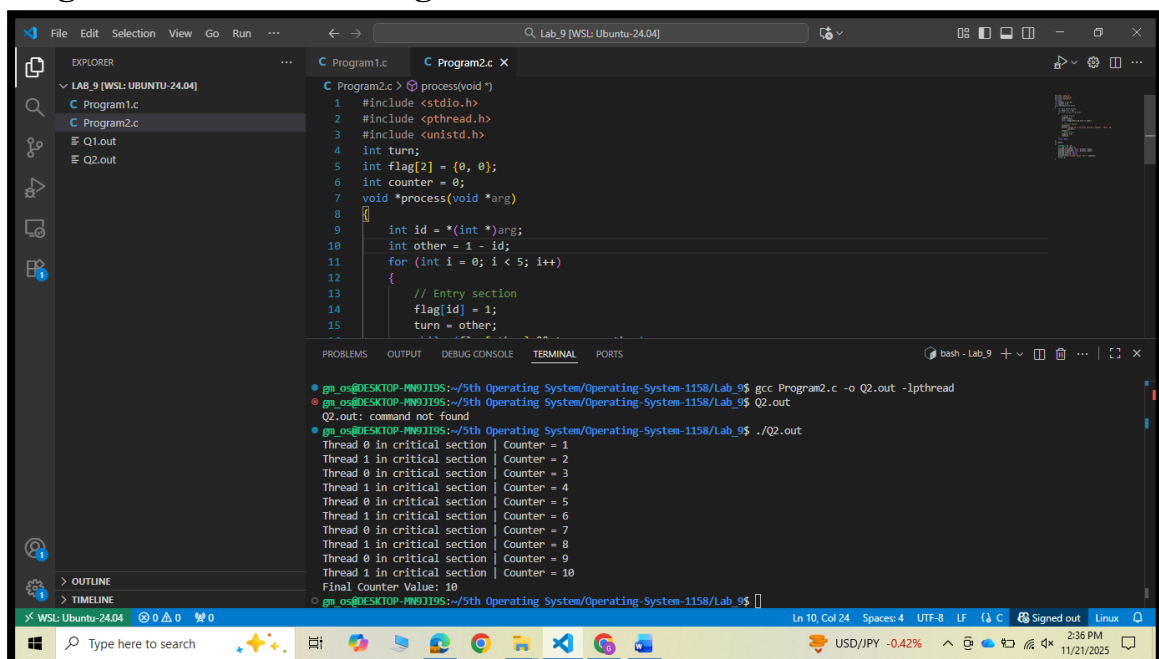


```
1 #include <stdio.h>
2 #include <pthread.h>
3 int counter = 0; // Shared resource
4 pthread_mutex_t lock; // Declare mutex
5 void *increment(void *arg)
6 {
7     for (int i = 0; i < 100000; i++)
8     {
9         pthread_mutex_lock(&lock); // Lock before accessing shared variable
10        counter++;
11        pthread_mutex_unlock(&lock); // Unlock after done
12    }
13    return NULL;
14 }
15
16 int main()
17 {
18     pthread_t t1, t2;
19     pthread_mutex_init(&lock, NULL); // Initialize mutex
20     pthread_create(&t1, NULL, increment, NULL);
21     pthread_create(&t2, NULL, increment, NULL);
22     pthread_join(t1, NULL);
23     pthread_join(t2, NULL);
24     printf("Expected counter value: 200000\n");
25 }
```

Terminal Output:

```
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program1.c -o Q1.out -lpthread
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q1.out
Expected counter value: 200000
Actual counter value: 200000
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Program#2: Peterson's Algorithm Simulation

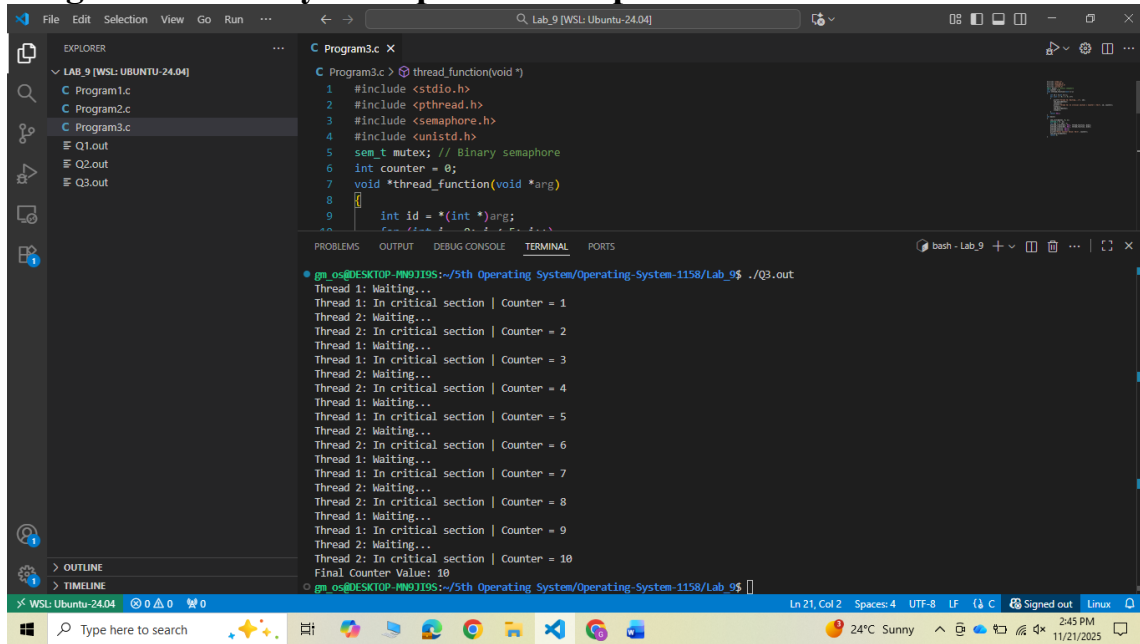


```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 int turn;
5 int flag[2] = {0, 0};
6 int counter = 0;
7 void *process(void *arg)
8 {
9     int id = *(int *)arg;
10    int other = 1 - id;
11    for (int i = 0; i < 5; i++)
12    {
13        // Entry section
14        flag[id] = 1;
15        turn = other;
16    }
17 }
```

Terminal Output:

```
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program2.c -o Q2.out -lpthread
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q2.out
Q2.out: command not found
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q2.out
Thread 0 in critical section | Counter = 1
Thread 1 in critical section | Counter = 2
Thread 0 in critical section | Counter = 3
Thread 1 in critical section | Counter = 4
Thread 0 in critical section | Counter = 5
Thread 1 in critical section | Counter = 6
Thread 0 in critical section | Counter = 7
Thread 1 in critical section | Counter = 8
Thread 0 in critical section | Counter = 9
Thread 1 in critical section | Counter = 10
Final Counter Value: 10
gn_os@DESKTOP-MN9J19S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Program#3: Binary Semaphore Example



The screenshot shows the Visual Studio Code editor with a file explorer on the left containing files like Program1.c, Program2.c, Program3.c, Q1.out, Q2.out, and Q3.out. The main editor displays Program3.c, which includes headers for stdio, pthread, semaphore, andunistd, and defines a binary semaphore mutex. The thread function increments a counter from 0 to 10. The terminal window shows the execution output, where two threads (Thread 1 and Thread 2) take turns incrementing the counter, with each thread printing 'Waiting...' and 'In critical section | Counter = X' messages. The final counter value is 10.

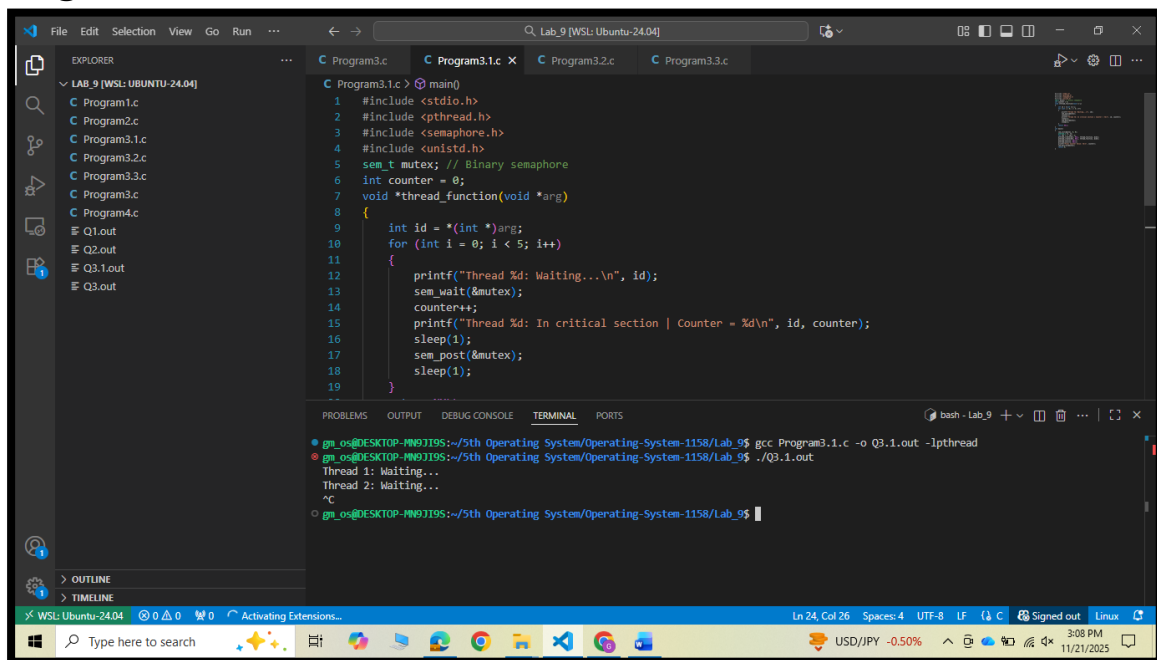
```
C Program3.c X
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 sem_t mutex; // Binary semaphore
6 int counter = 0;
7 void *thread_function(void *arg)
8 {
9     int id = *(int *)arg;
10    for (int i = 0; i < 10; i++)
11    {
12        printf("Thread %d: Waiting...\n", id);
13        sem_wait(&mutex);
14        counter++;
15        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
16        sleep(1);
17        sem_post(&mutex);
18        sleep(1);
19    }
20 }
```

```
bash - Lab_9
gn_os@DESKTOP-MN9J1T9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q3.out
Thread 1: Waiting...
Thread 2: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 2
Thread 1: Waiting...
Thread 1: In critical section | Counter = 3
Thread 2: Waiting...
Thread 2: In critical section | Counter = 4
Thread 1: Waiting...
Thread 1: In critical section | Counter = 5
Thread 2: Waiting...
Thread 2: In critical section | Counter = 6
Thread 1: Waiting...
Thread 1: In critical section | Counter = 7
Thread 2: Waiting...
Thread 2: In critical section | Counter = 8
Thread 1: Waiting...
Thread 1: In critical section | Counter = 9
Thread 2: Waiting...
Thread 2: In critical section | Counter = 10
Final Counter Value: 10
gn_os@DESKTOP-MN9J1T9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Description:

As the Semaphore value is 1, so it run the Wait, and then decrease S by 1 and enter to critical section.

Program#3.1



The screenshot shows the Visual Studio Code editor with a file explorer on the left containing files like Program1.c, Program2.c, Program3.1.c, Program3.2.c, Program3.3.c, Program4.c, Q1.out, Q2.out, Q3.1.out, and Q3.out. The main editor displays Program3.1.c, which includes headers for stdio, pthread, semaphore, andunistd, and defines a binary semaphore mutex. The main function calls pthread_create to create two threads, which then increment a counter from 0 to 5. The terminal window shows the execution output, where both threads (Thread 1 and Thread 2) are in a 'Waiting...' state because the semaphore value is zero.

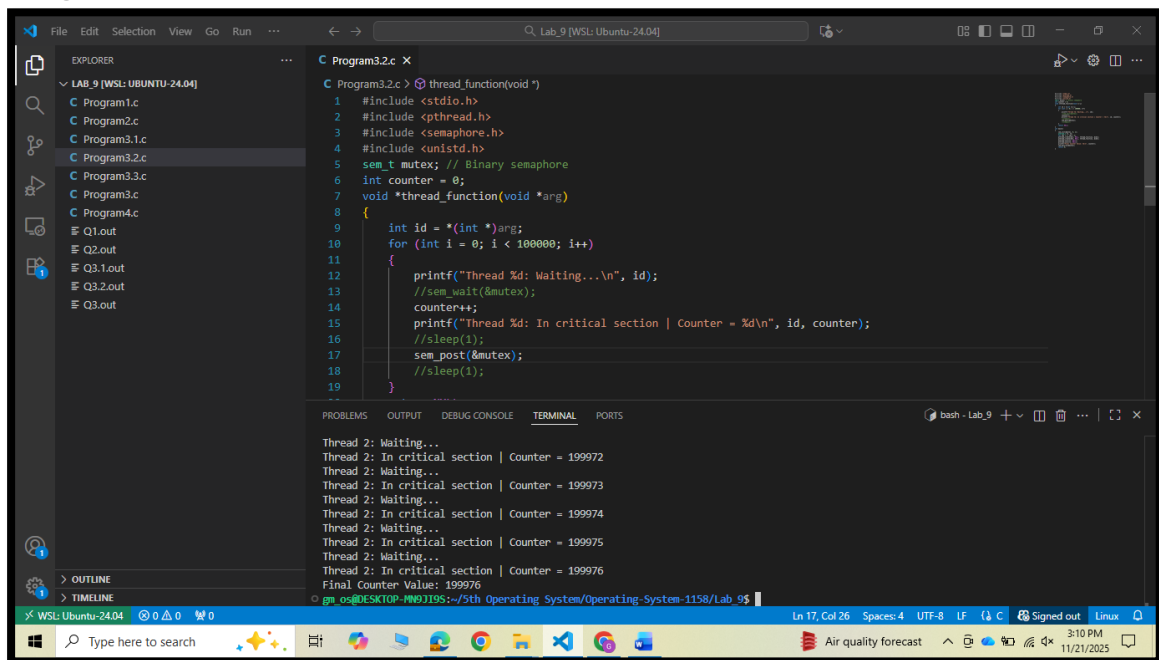
```
C Program3.1.c X
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 sem_t mutex; // Binary semaphore
6 int counter = 0;
7 void *thread_function(void *arg)
8 {
9     int id = *(int *)arg;
10    for (int i = 0; i < 5; i++)
11    {
12        printf("Thread %d: Waiting...\n", id);
13        sem_wait(&mutex);
14        counter++;
15        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
16        sleep(1);
17        sem_post(&mutex);
18        sleep(1);
19    }
20 }
```

```
bash - Lab_9
gn_os@DESKTOP-MN9J1T9S:~/5th Operating System/Operating-System-1158/Lab_9$ gcc Program3.1.c -o Q3.1.out -lpthread
gn_os@DESKTOP-MN9J1T9S:~/5th Operating System/Operating-System-1158/Lab_9$ ./Q3.1.out
Thread 1: Waiting...
Thread 2: Waiting...
^C
gn_os@DESKTOP-MN9J1T9S:~/5th Operating System/Operating-System-1158/Lab_9$
```

Description:

As S is zero, so the condition of Wait is not true, so we All the process are in waiting.

Program#3.2



The screenshot shows the Visual Studio Code editor with a file explorer on the left displaying a directory structure for 'LAB_9 [WSL: UBUNTU-24.04]'. The main editor window shows 'Program3.2.c' with the following code:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 sem_t mutex; // Binary semaphore
6 int counter = 0;
7 void *thread_function(void *arg)
8 {
9     int id = *(int *)arg;
10    for (int i = 0; i < 100000; i++)
11    {
12        printf("Thread %d: Waiting...\n", id);
13        //sem_wait(&mutex);
14        counter++;
15        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
16        //sleep(1);
17        sem_post(&mutex);
18        //sleep(1);
19    }
20 }
```

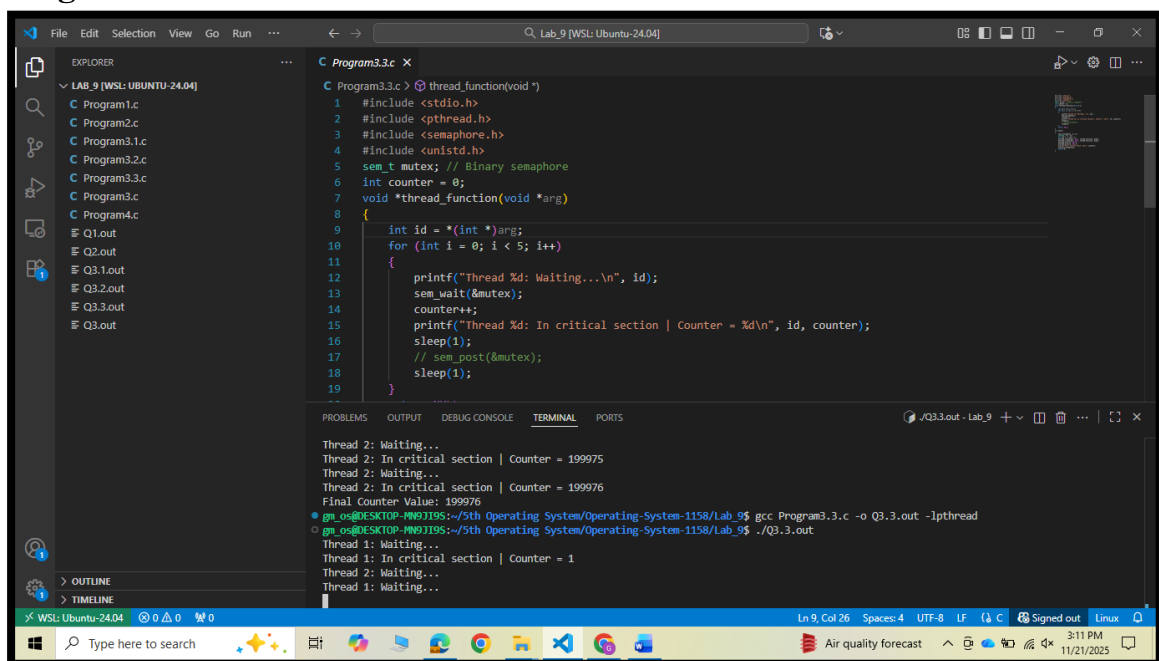
The terminal output shows the execution of the program, where Thread 2 is shown waiting and then entering the critical section multiple times, incrementing the counter. The final output is:

```
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199972
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199973
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199974
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199975
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199976
Final Counter Value: 199976
```

Description:

After changing loop iteration and, commenting Wait and Sleep, it provides the wrong value, the value of thread must be the iteration of loop but it comes wrong.

Program#3.3



The screenshot shows the Visual Studio Code editor with a file explorer on the left displaying a directory structure for 'LAB_9 [WSL: UBUNTU-24.04]'. The main editor window shows 'Program3.3.c' with the following code:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 sem_t mutex; // Binary semaphore
6 int counter = 0;
7 void *thread_function(void *arg)
8 {
9     int id = *(int *)arg;
10    for (int i = 0; i < 5; i++)
11    {
12        printf("Thread %d: Waiting...\n", id);
13        sem_wait(&mutex);
14        counter++;
15        printf("Thread %d: In critical section | Counter = %d\n", id, counter);
16        sleep(1);
17        // sem_post(&mutex);
18        sleep(1);
19    }
20 }
```

The terminal output shows the execution of the program, where Thread 2 is shown waiting and then entering the critical section multiple times, incrementing the counter. The final output is:

```
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199975
Thread 2: Waiting...
Thread 2: In critical section | Counter = 199976
Final Counter Value: 199976
gcc Program3.3.c -o Q3.3.out -lpthread
./Q3.3.out
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
```

Description:

After commenting Signal, the value of S in not increase so the value of Wait is false and all process goes on Waiting.

Program#3.3

The screenshot shows a Visual Studio Code editor window with a C program named 'Program3.4.c'. The program includes headers for `<stdio.h>`, `<pthread.h>`, `<semaphore.h>`, and `<unistd.h>`. It defines a binary semaphore `sem_t mutex;` and a counter `int counter = 0;`. A function `void *thread_function(void *arg)` is defined, which takes an integer argument `id` and increments the counter in a loop from `i = 0` to `i < 5`. The main function calls `pthread_t t1, t2, t3, t4;` and `pthread_create` to create four threads, each with its own ID. The threads are then joined using `pthread_join`. The output window shows the execution of the program, displaying the thread IDs and the counter value at each step. The final output is 'Final Counter Value: 0'.

Description:

- ❖ Create a new function in which decrementing the counter, so the value after whole execution of Thread is 0.
- ❖ Because in thread_function counter is incrementing by 1 in each iteration, and in thread_function1 counter is decrementing by 1 in each iteration, so after the whole execution of main the counter becomes 0 again.

Task#4

Mutex:	Semaphore:
Mutex used lock and unlock for entry in critical section, or for synchronization.	Semaphore uses Wait and Signal for synchronization.
Mutex allows only one thread to change the critical section at a time.	Semaphore use Wait to allow a thread to change if the condition becomes true.
Mutex ownership is the thread that lock it only unlock it.	Semaphore use Signal to enter another thread/process.