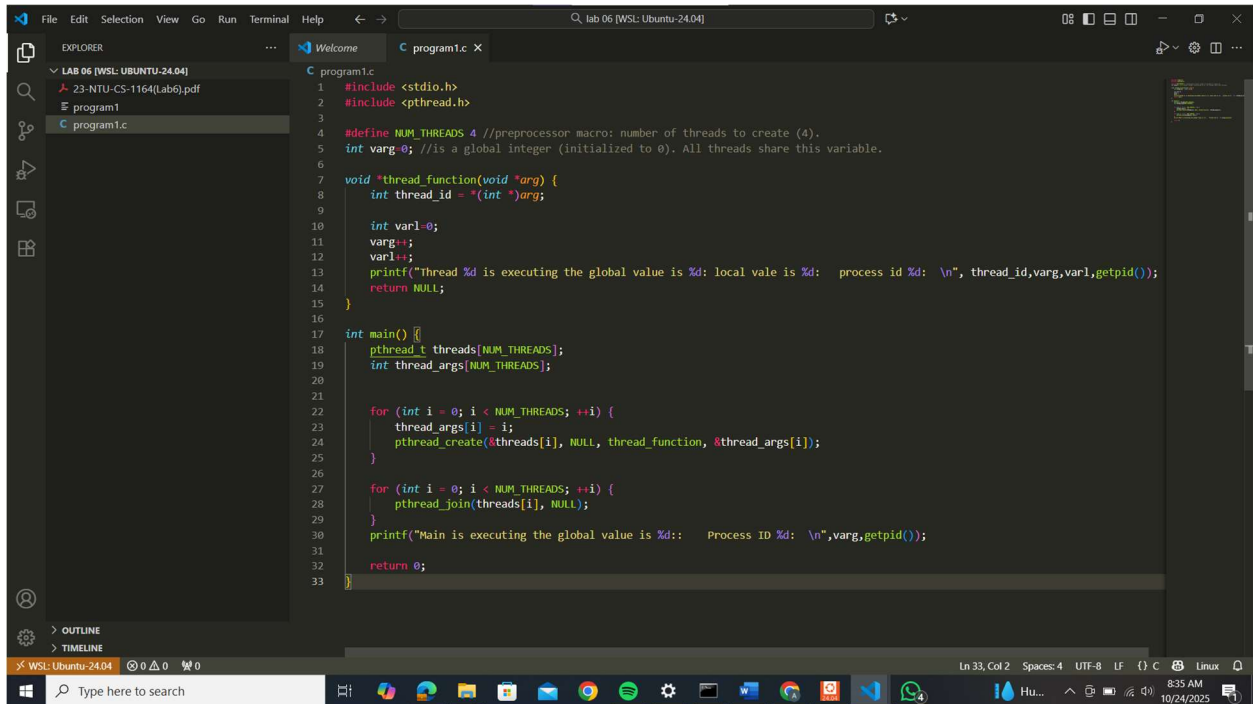
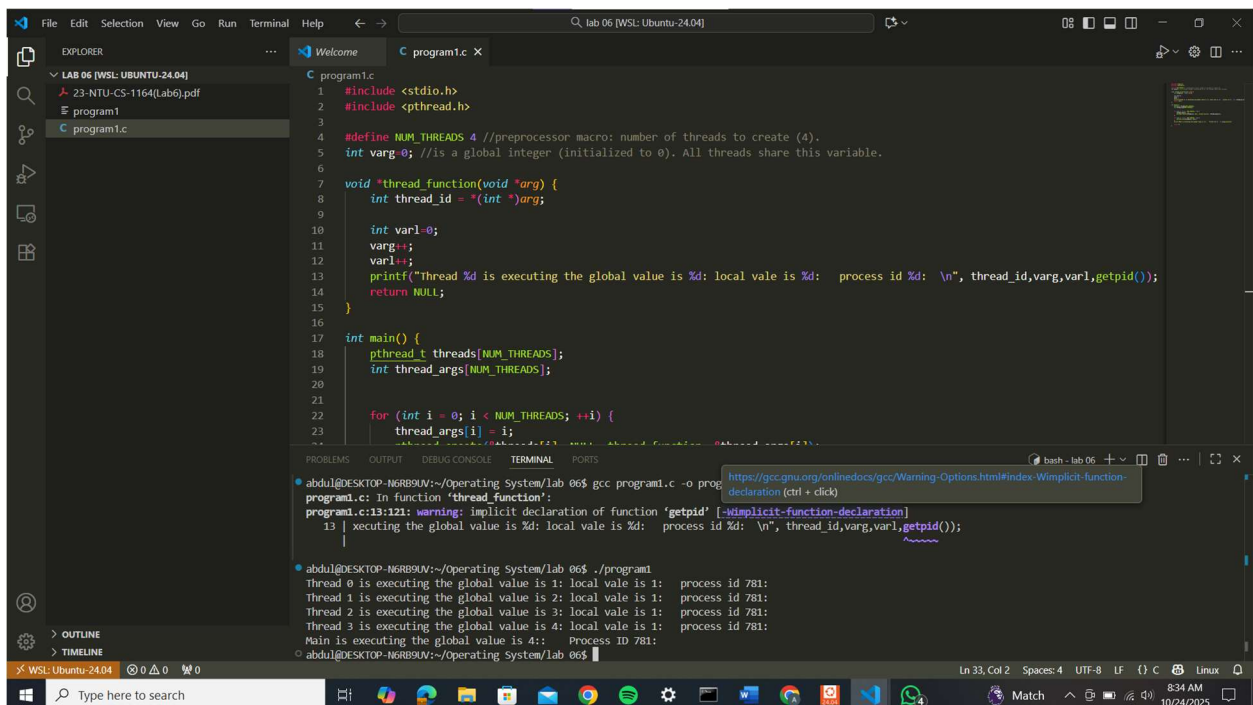


CODE



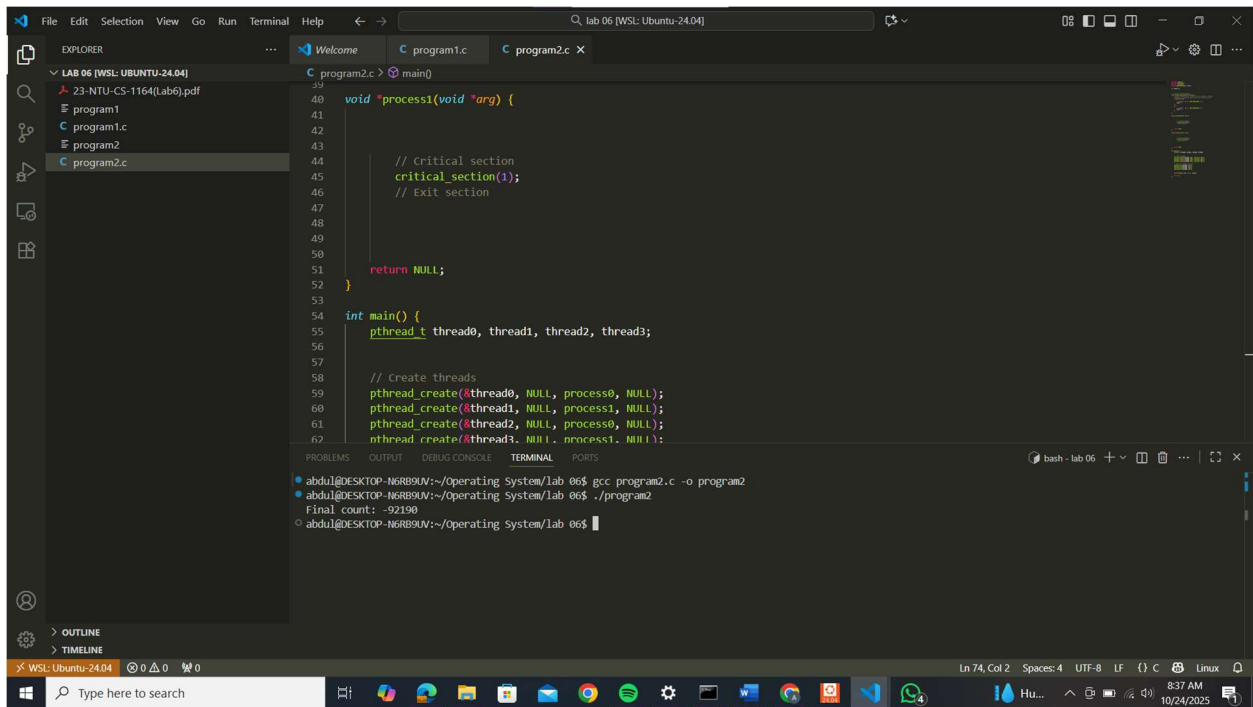
```
1 #include <stdio.h>
2 #include <pthread.h>
3
4 #define NUM_THREADS 4 //preprocessor macro: number of threads to create (4).
5 int varg = 0; //is a global integer (initialized to 0). All threads share this variable.
6
7 void *thread_function(void *arg) {
8     int thread_id = *(int *)arg;
9
10    int varl = 0;
11    varg++;
12    varl++;
13    printf("Thread %d is executing the global value is %d: local value 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 }
```

OUTPUT



```
abdu@DESKTOP-NGR89UV:~/Operating System/Lab 06$ gcc program1.c -o program1
program1.c: In function 'thread_function':
program1.c:13:121: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
13 |   printf("Thread %d is executing the global value is %d: local value is %d: process id %d: \n", thread_id, varg, varl, getpid());
   |                                                             ^~~~~~
   |                                                             |
   |                                                             https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html#Index-Wimplicit-function-declaration (ctrl + click)
abdu@DESKTOP-NGR89UV:~/Operating System/Lab 06$ ./program1
Thread 0 is executing the global value is 1: local value is 1: process id 781:
Thread 1 is executing the global value is 2: local value is 1: process id 781:
Thread 2 is executing the global value is 3: local value is 1: process id 781:
Thread 3 is executing the global value is 4: local value is 1: process id 781:
Main is executing the global value is 4: Process ID 781:
abdu@DESKTOP-NGR89UV:~/Operating System/Lab 06$
```

CODE



```
1 void *process1(void *arg) {
2     // Critical section
3     critical_section(1);
4     // Exit section
5
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread0, thread1, thread2, thread3;
11
12     // Create threads
13     pthread_create(&thread0, NULL, process0, NULL);
14     pthread_create(&thread1, NULL, process1, NULL);
15     pthread_create(&thread2, NULL, process0, NULL);
16     pthread_create(&thread3, NULL, process1, NULL);
17
18     // Wait for threads to finish
19     pthread_join(thread0, NULL);
20     pthread_join(thread1, NULL);
21     pthread_join(thread2, NULL);
22     pthread_join(thread3, NULL);
23
24     printf("Final count: %d\n", count);
25     return 0;
26 }
```

bash - lab 06

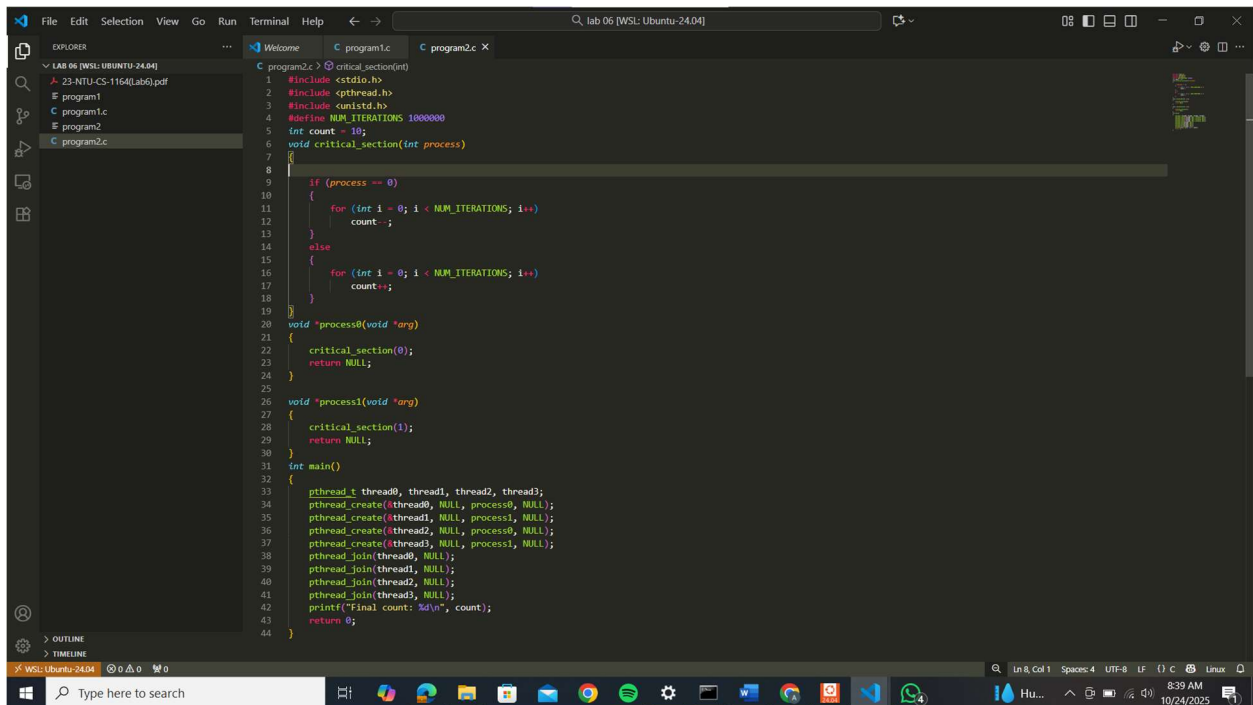
abdu@DESKTOP-N6R89UV:~/Operating System/lab 06\$ gcc program2.c -o program2

abdu@DESKTOP-N6R89UV:~/Operating System/lab 06\$./program2

Final count: -92190

abdu@DESKTOP-N6R89UV:~/Operating System/lab 06\$

OUTPUT



```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 1000000
5 int count = 0;
6 void critical_section(int process)
7 {
8     if (process == 0)
9     {
10         for (int i = 0; i < NUM_ITERATIONS; i++)
11             count++;
12     }
13     else
14     {
15         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18 }
19
20 void *process0(void *arg)
21 {
22     critical_section(0);
23     return NULL;
24 }
25
26 void *process1(void *arg)
27 {
28     critical_section(1);
29     return NULL;
30 }
31
32 int main()
33 {
34     pthread_t thread0, thread1, thread2, thread3;
35     pthread_create(&thread0, NULL, process0, NULL);
36     pthread_create(&thread1, NULL, process1, NULL);
37     pthread_create(&thread2, NULL, process0, NULL);
38     pthread_create(&thread3, NULL, process1, NULL);
39     pthread_join(thread0, NULL);
40     pthread_join(thread1, NULL);
41     pthread_join(thread2, NULL);
42     pthread_join(thread3, NULL);
43     printf("Final count: %d\n", count);
44     return 0;
45 }
```

Ln 8, Col 1 Spaces: 4 UTF-8 LF {} C Linux

8:39 AM 10/24/2025

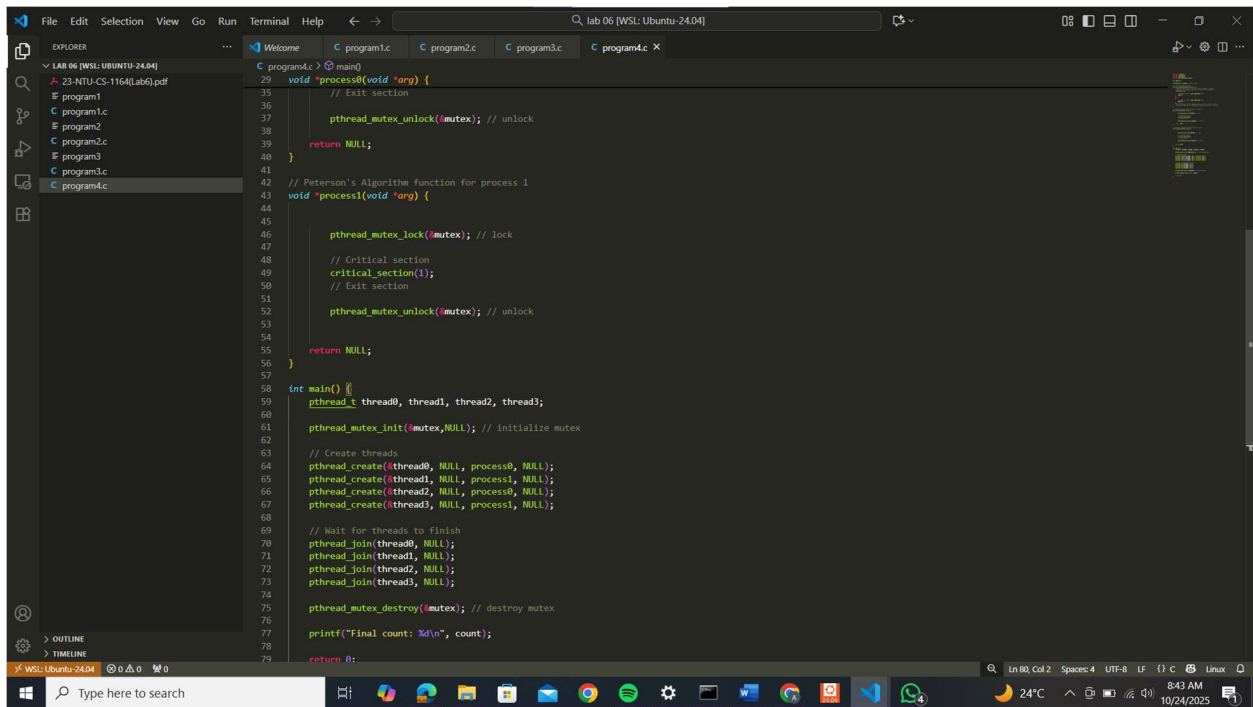
CODE

```
11 void critical_section(int process)
12 {
13     if (process == 0)
14         count++;
15     }
16     else
17     {
18         for (int i = 0; i < NUM_ITERATIONS; i++)
19             count++;
20     }
21 }
22
23 void *process0(void *arg)
24 {
25     flag[0] = 1;
26     turn = 1;
27     while (flag[1] == 1 && turn == 1)
28     {
29         // Busy wait
30     }
31     critical_section(0);
32     flag[0] = 0;
33     pthread_exit(NULL);
34 }
35
36 void *process1(void *arg)
37 {
38     flag[1] = 1;
39     turn = 0;
40     while (flag[0] == 1 && turn == 0)
41     {
42     }
43     critical_section(1);
44     // Exit section
45     flag[1] = 0;
46     // sleep(1);
47     pthread_exit(NULL);
48 }
49
50 int main()
51 {
52     pthread_t thread0, thread1;
53     // Initialize shared variables
54     flag[0] = 0;
55     flag[1] = 0;
```

OUTPUT

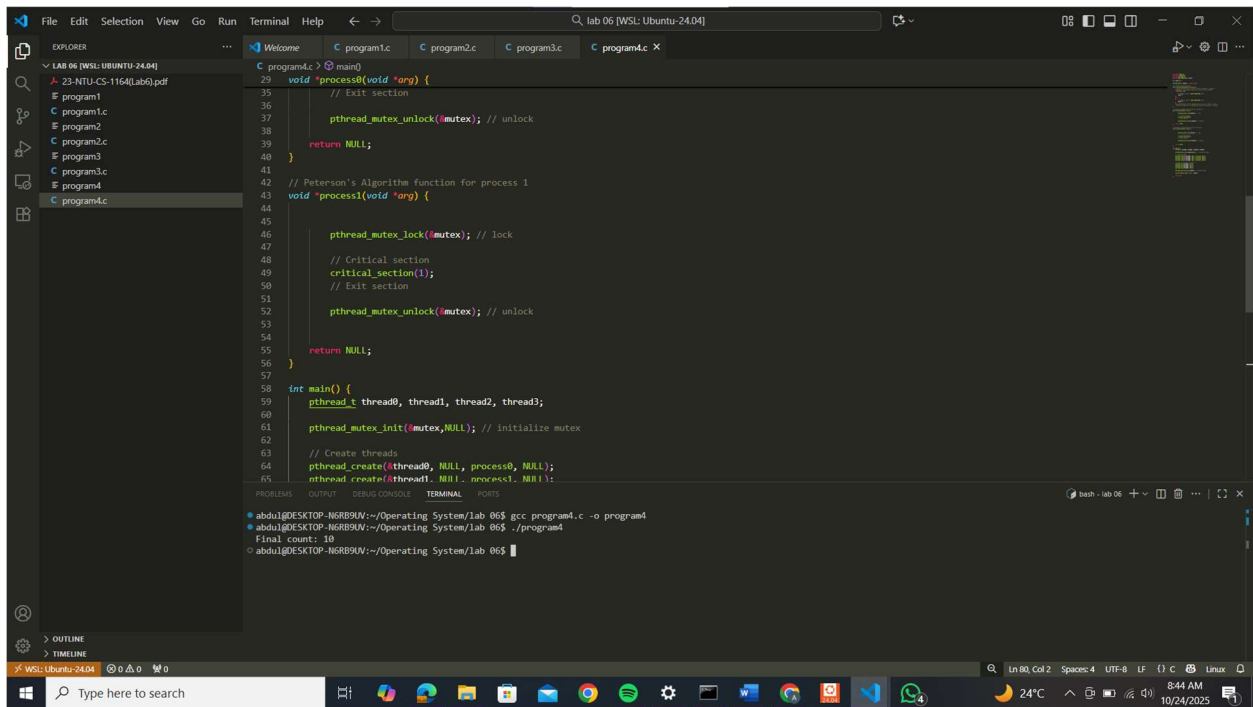
```
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$ gcc program3.c -o program3
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$ ./program3
Final count: 0
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$
```

CODE



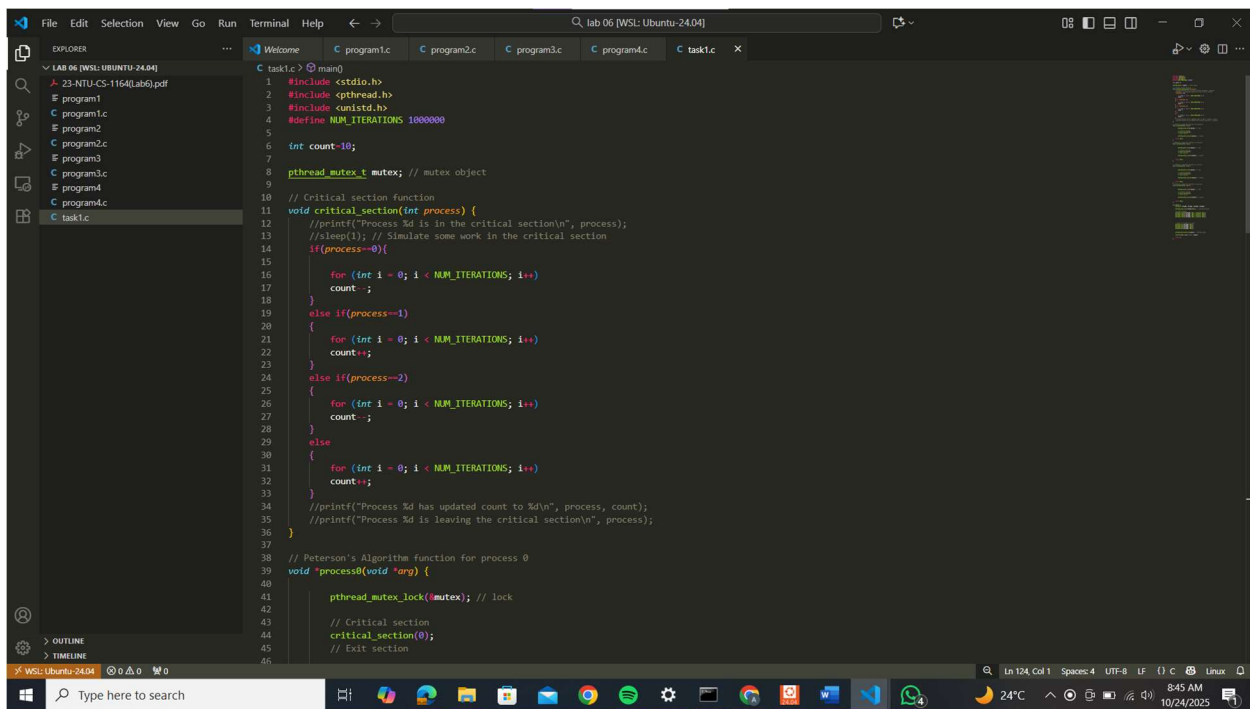
```
29 void *process0(void *arg) {
30     // Exit section
31     pthread_mutex_unlock(&mutex); // unlock
32     return NULL;
33 }
34
35 // Peterson's Algorithm function for process 1
36 void *process1(void *arg) {
37     pthread_mutex_lock(&mutex); // lock
38     // Critical section
39     critical_section(1);
40     // Exit section
41     pthread_mutex_unlock(&mutex); // unlock
42     return NULL;
43 }
44
45 int main() {
46     pthread_t thread0, thread1, thread2, thread3;
47     pthread_mutex_init(&mutex, NULL); // initialize mutex
48
49     // Create threads
50     pthread_create(&thread0, NULL, process0, NULL);
51     pthread_create(&thread1, NULL, process1, NULL);
52     pthread_create(&thread2, NULL, process0, NULL);
53     pthread_create(&thread3, NULL, process1, NULL);
54
55     // Wait for threads to finish
56     pthread_join(thread0, NULL);
57     pthread_join(thread1, NULL);
58     pthread_join(thread2, NULL);
59     pthread_join(thread3, NULL);
60
61     pthread_mutex_destroy(&mutex); // destroy mutex
62     printf("Final count: %d\n", count);
63     return 0;
64 }
```

OUTPUT



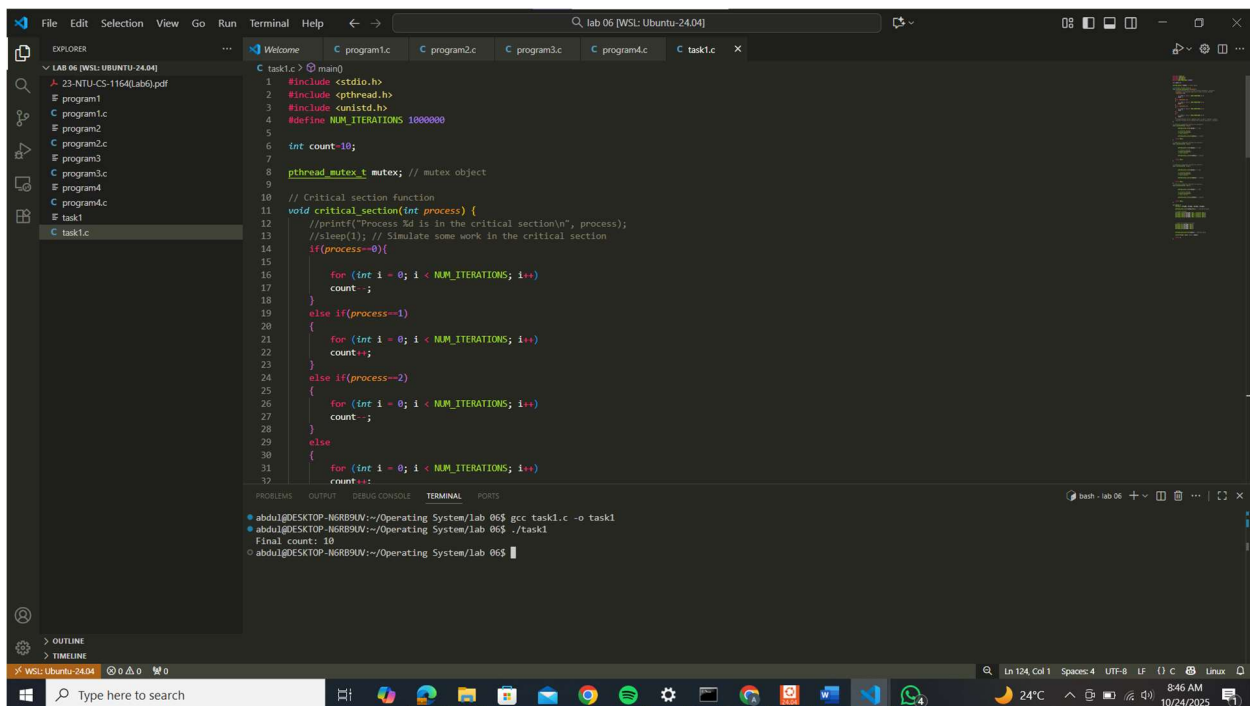
```
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$ gcc program4.c -o program4
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$ ./program4
Final count: 10
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$
```

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 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++)
17             count--;
18     }
19     else if(process==1)
20     {
21         for (int i = 0; i < NUM_ITERATIONS; i++)
22             count++;
23     }
24     else if(process==2)
25     {
26         for (int i = 0; i < NUM_ITERATIONS; i++)
27             count--;
28     }
29     else
30     {
31         for (int i = 0; i < NUM_ITERATIONS; i++)
32             count++;
33     }
34     //printf("Process %d has updated count to %d\n", process, count);
35     //printf("Process %d is leaving the critical section\n", process);
36 }
37
38 // Peterson's Algorithm function for process 0
39 void *process0(void *arg) {
40
41     pthread_mutex_lock(&mutex); // lock
42
43     // Critical section
44     critical_section(0);
45     // Exit section
46 }
```

OUTPUT



```
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$ gcc task1.c -o task1
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$ ./task1
Final count: 10
abdu1@DESKTOP-N6R89UN:~/Operating System/lab 06$
```