



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Akasha Fatima

Reg. number:

23-NTU-CS-FL-1132

Semester:

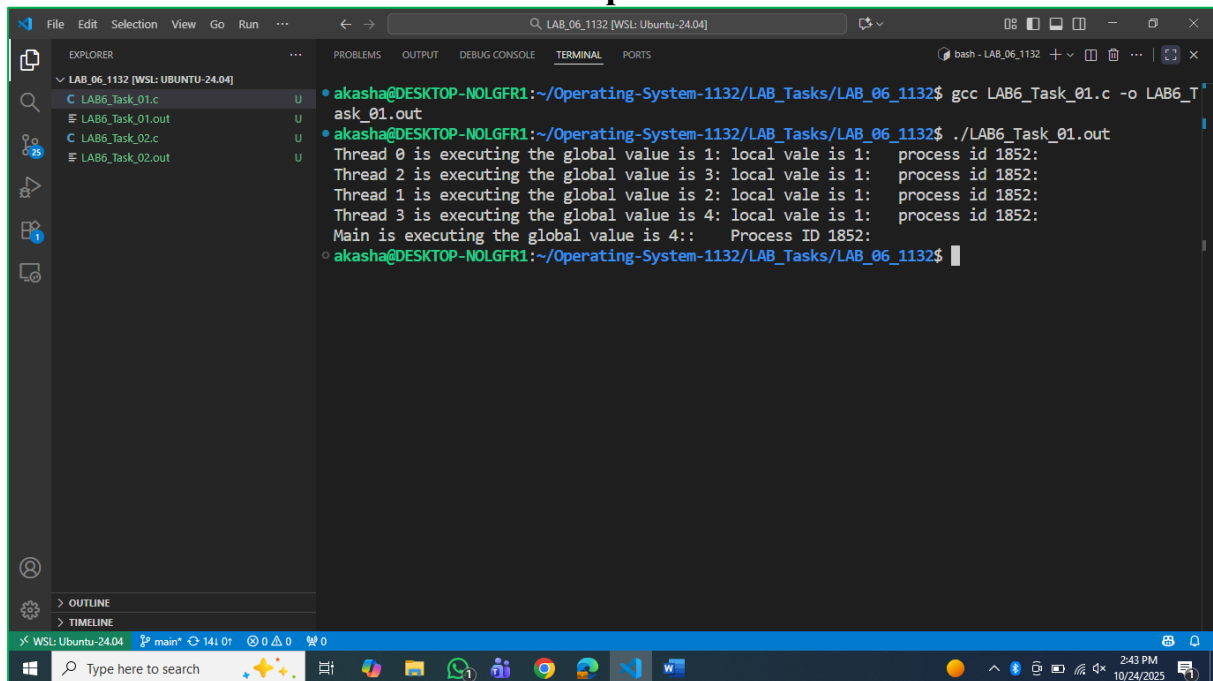
5th - A

LAB-06

Task_01: Simple threads Execution: CODE:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
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,var1,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",var,getpid());
31
32     return 0;
33 }
```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_01.c -o LAB6_T
ask_01.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_01.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 1852:
Thread 2 is executing the global value is 3: local vale is 1:   process id 1852:
Thread 1 is executing the global value is 2: local vale is 1:   process id 1852:
Thread 3 is executing the global value is 4: local vale is 1:   process id 1852:
Main is executing the global value is 4::   Process ID 1852:
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

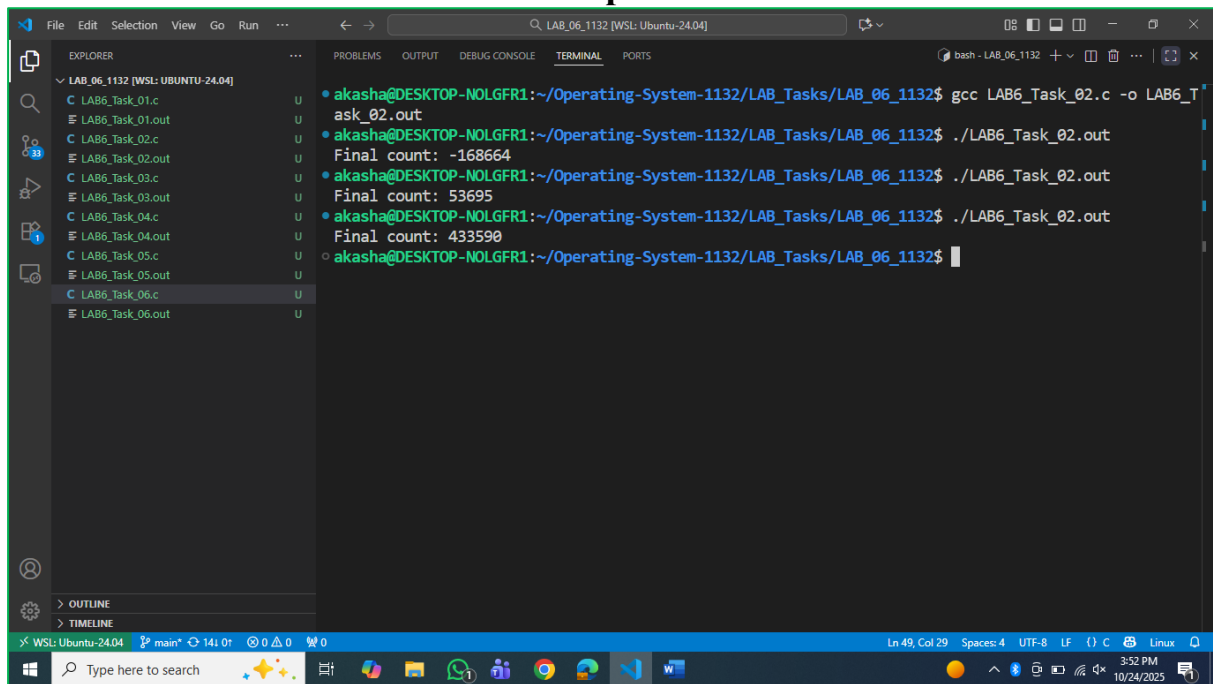
Task_02: Synchronization Problem CODE:

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #define NUM_ITERATIONS 1000000
5
6  int count=10;
7
8
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
27 void *process0(void *arg) {
28
29     // Critical section
30     critical_section(0);
31     // Exit section
32
33     return NULL;
34 }
35
36 void *process1(void *arg) {
37
38     // Critical section
39     critical_section(1);
40     // Exit section
41
42     return NULL;
43 }
44
45 int main() {
46     pthread_t thread0, thread1, thread2, thread3;
47
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
62     printf("Final count: %d\n", count);
63
64     return 0;
65 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_02.c -o LAB6_Task_02.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_02.out
Final count: -168664
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_02.out
Final count: 53695
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_02.out
Final count: 433590
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

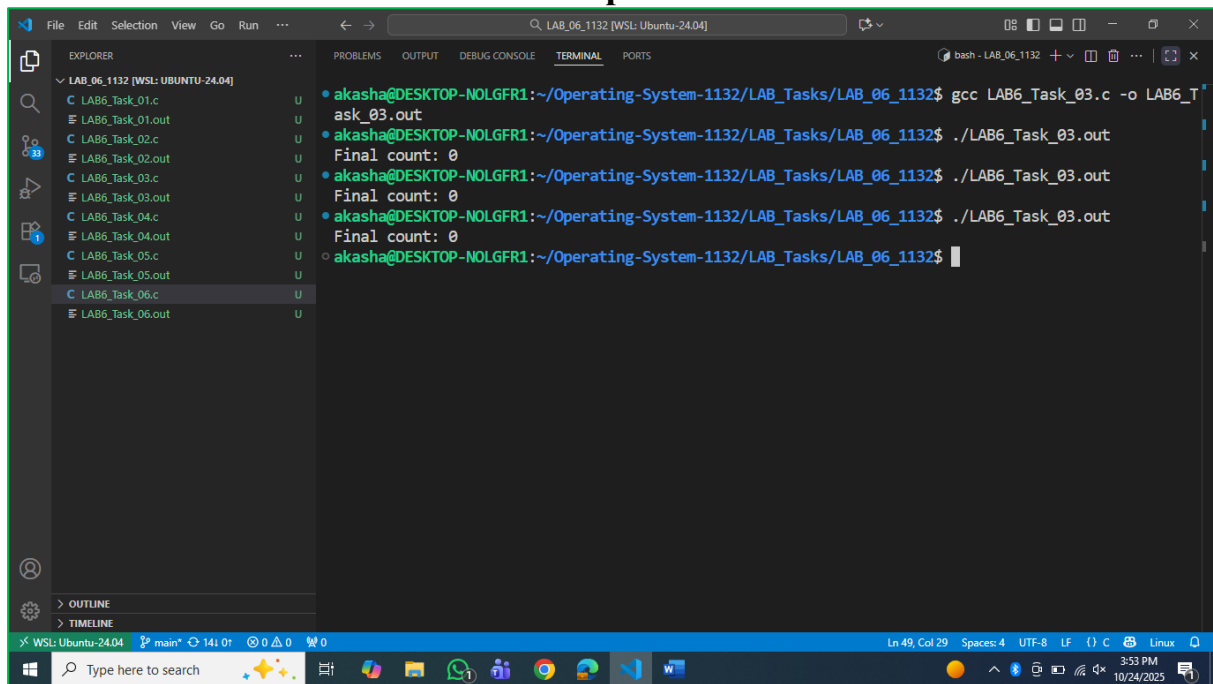
Task_03: Peterson Solution for Synchronization Issue of Task 02: CODE:

```

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
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     // printf("Process %d has updated count to %d\n", process, count);
26     //printf("Process %d is leaving the critical section\n", process);
27 }
28
29 // Peterson's Algorithm function for process 0
30 void *process0(void *arg) {
31
32     flag[0] = 1;
33     turn = 1;
34     while (flag[1]==1 && turn == 1) {
35         // Busy wait
36     }
37     // Critical section
38     critical_section(0);
39     // Exit section
40     flag[0] = 0;
41     //sleep(1);
42
43
44     pthread_exit(NULL);
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
74     // Create threads
75     pthread_create(&thread0, NULL, process0, NULL);
76     pthread_create(&thread1, NULL, process1, NULL);
77
78     // Wait for threads to finish
79     pthread_join(thread0, NULL);
80     pthread_join(thread1, NULL);
81
82     printf("Final count: %d\n", count);
83
84     return 0;
85 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_03.c -o LAB6_Task_03.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_03.out
Final count: 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_03.out
Final count: 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_03.out
Final count: 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

Task_04: Mutex Solution for Synchronization Issue of Task 02 with two processes only:

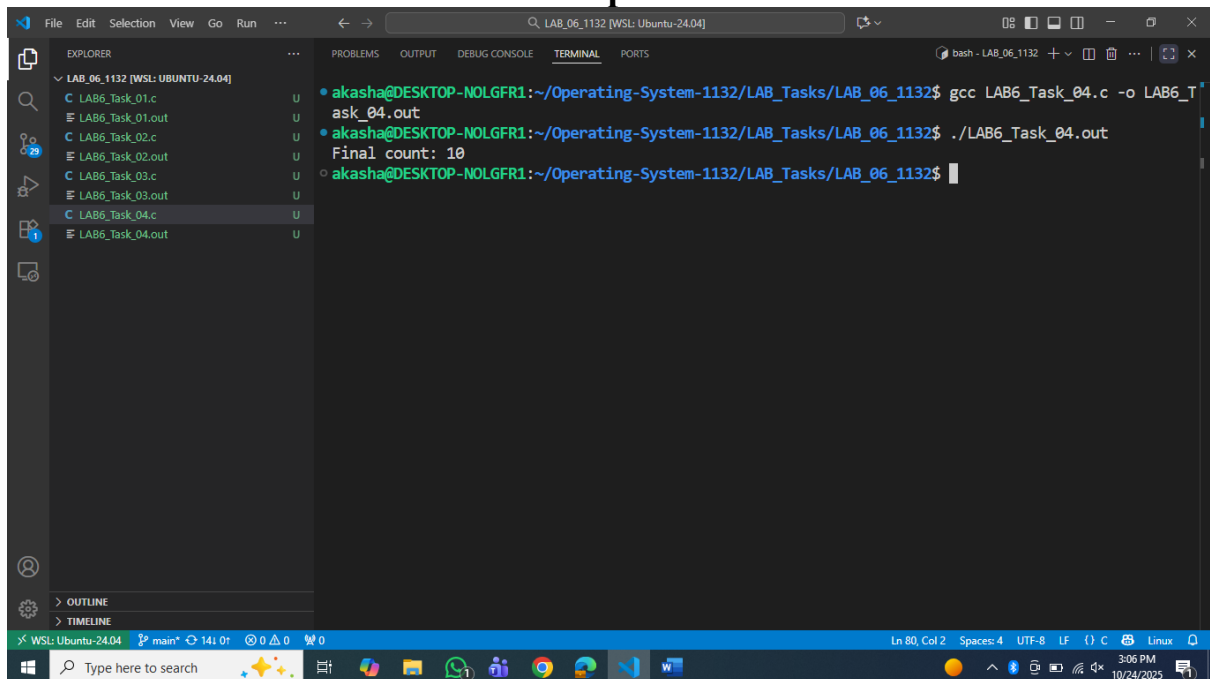
CODE:

```

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
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     return NULL;
40 }
41
42 // Peterson's Algorithm function for process 1
43 void *process1(void *arg) {
44
45     pthread_mutex_lock(&mutex); // lock
46
47     // Critical section
48     critical_section(1);
49     // Exit section
50
51     pthread_mutex_unlock(&mutex); // unlock
52
53
54
55     return NULL;
56 }
57
58 int main() {
59     pthread_t thread0, thread1, thread2, thread3;
60
61     pthread_mutex_init(&mutex, NULL); // initialize mutex
62
63     // Create threads
64     pthread_create(&thread0, NULL, process0, NULL);
65     pthread_create(&thread1, NULL, process1, NULL);
66     pthread_create(&thread2, NULL, process0, NULL);
67     pthread_create(&thread3, NULL, process1, NULL);
68
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 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB_Task_04.c -o LAB_Task_04.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB_Task_04.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

Difference between Mutex and Peterson Methods:

Peterson	Mutex
It uses the while loops.	It uses the build-in Lock and Unlock commands
The code is totally written by the user at the program written time.	The code is the build in for the Lock and Unlock commands.
It is used for only two processes.	It can be used for any numbers of the process

Task_05: Mutex Solution for Synchronization Issue of Task 02 with three processes:

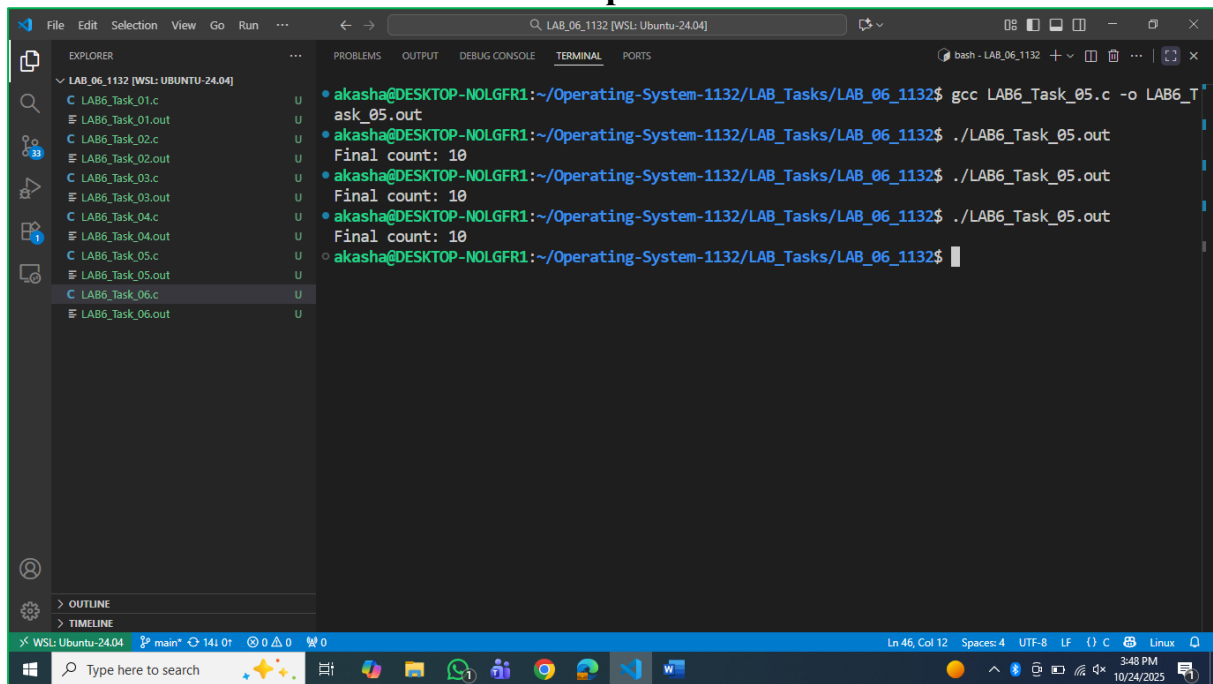
CODE:


```

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
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     return NULL;
40 }
41
42 // Peterson's Algorithm function for process 1
43 void *process1(void *arg) {
44
45     pthread_mutex_lock(&mutex); // lock
46
47     // Critical section
48     critical_section(1);
49     // Exit section
50
51     pthread_mutex_unlock(&mutex); // unlock
52
53     return NULL;
54 }
55
56 // Peterson's Algorithm function for process 2
57 void *process2(void *arg) {
58
59     pthread_mutex_lock(&mutex); // lock
60
61     // Critical section
62     critical_section(2);
63     // Exit section
64
65     pthread_mutex_unlock(&mutex); // unlock
66
67     return NULL;
68 }
69
70 int main() {
71     pthread_t thread0, thread1, thread2, thread3;
72
73     pthread_mutex_init(&mutex, NULL); // initialize mutex
74
75     // Create threads
76     pthread_create(&thread0, NULL, process0, NULL);
77     pthread_create(&thread1, NULL, process1, NULL);
78     pthread_create(&thread2, NULL, process0, NULL);
79     pthread_create(&thread3, NULL, process2, NULL);
80
81     // Wait for threads to finish
82     pthread_join(thread0, NULL);
83     pthread_join(thread1, NULL);
84     pthread_join(thread2, NULL);
85     pthread_join(thread3, NULL);
86
87     pthread_mutex_destroy(&mutex); // destroy mutex
88
89     printf("Final count: %d\n", count);
90
91     return 0;
92 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_05.c -o LAB6_Task_05.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_05.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_05.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_05.out
Final count: 10
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```

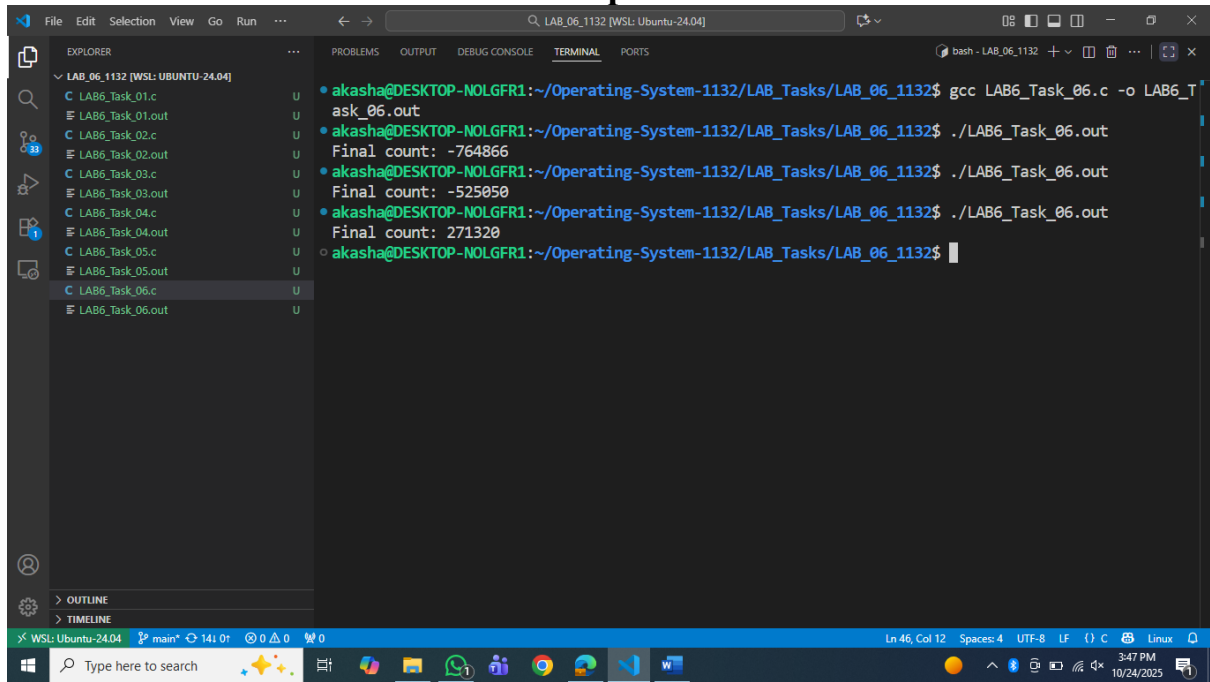
Task_06: Commenting the “Lock” and “Unlock” commands: CODE:

```

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
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     return NULL;
40 }
41
42 // Peterson's Algorithm function for process 1
43 void *process1(void *arg) {
44
45     // pthread_mutex_lock(&mutex); // lock
46
47     // Critical section
48     critical_section(1);
49     // Exit section
50
51     // pthread_mutex_unlock(&mutex); // unlock
52
53
54
55     return NULL;
56 }
57
58 int main() {
59     pthread_t thread0, thread1, thread2, thread3;
60
61     pthread_mutex_init(&mutex, NULL); // initialize mutex
62
63     // Create threads
64     pthread_create(&thread0, NULL, process0, NULL);
65     pthread_create(&thread1, NULL, process1, NULL);
66     pthread_create(&thread2, NULL, process0, NULL);
67     pthread_create(&thread3, NULL, process1, NULL);
68
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 }

```

Output:



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the output of a C program that calculates the sum of squares of integers from 1 to 10. The program is compiled and run in a WSL environment (Ubuntu-24.04). The output shows the final count for each iteration of the loop, which is the sum of squares up to that point.

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ gcc LAB6_Task_06.c -o LAB6_Task_06.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_06.out
Final count: -764866
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_06.out
Final count: -525050
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$ ./LAB6_Task_06.out
Final count: 271320
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_06_1132$
```