



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir

Submitted by:

Esha Mubashir Khan

Reg number:

23-NTU-CS-1151

Lab no: **6**

Semester: **5th**

TASK 1:

```
1  #include <stdio.h>
2  #include <pthread.h>
3
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 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 int main() {
17     pthread_t threads[NUM_THREADS];
18     int thread_args[NUM_THREADS];
19
20     for (int i = 0; i < NUM_THREADS; ++i) {
21         thread_args[i] = i;
22         pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
23     }
24
25     for (int i = 0; i < NUM_THREADS; ++i) {
26         pthread_join(threads[i], NULL);
27     }
28     printf("Main is executing the global value is %d::   Process ID %d:  \n",varg,getpid());
29
30     return 0;
31 }
```

The screenshot shows the Visual Studio Code interface with a C file named `q1.c` open. The file contains a multi-threaded program. The `main` function creates four threads, each of which increments a global variable `varg` and prints its value along with the local `varl` and the process ID. The terminal shows the compilation and execution of the program, with a warning about an implicit declaration of `getpid`.

```
q1.c > main()
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 int main() {
17     pthread_t threads[NUM_THREADS];
18     int thread_args[NUM_THREADS];
```

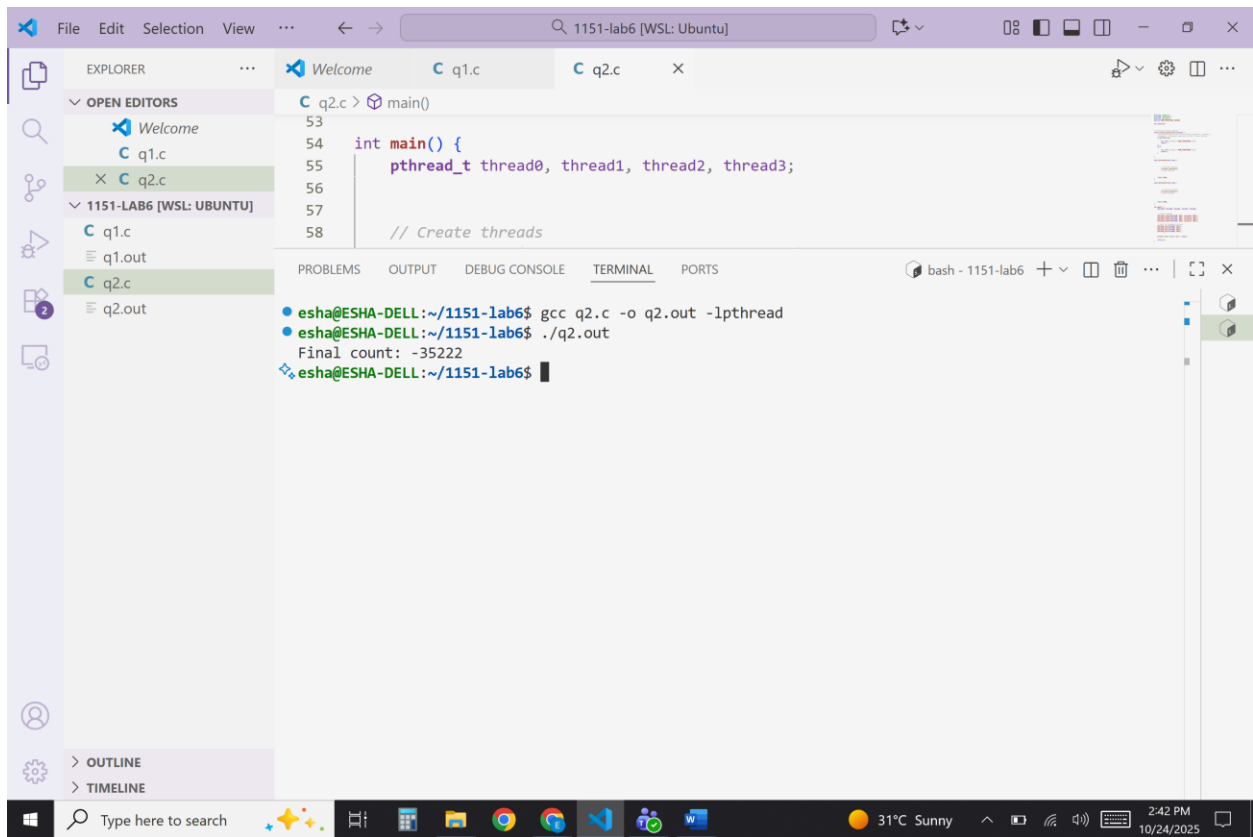
Terminal Output:

```
esha@ESHA-DELL:~/1151-lab6$ gcc q1.c -o q1.out -lpthread
q1.c: In function 'thread_function':
q1.c:13:121: warning: implicit declaration of function 'getpid' [-Wimplicit-function-declaration]
   13 |     printf("Thread %d is executing the global value is %d: local vale is %d:  process id %d:  \n", thread_id, varg, varl, getpid());
      |                                                                                                                              ~~~~~
esha@ESHA-DELL:~/1151-lab6$ ./q1.out
Thread 0 is executing the global value is 1: local vale is 1:  process id 1994:
Thread 1 is executing the global value is 2: local vale is 1:  process id 1994:
Thread 2 is executing the global value is 3: local vale is 1:  process id 1994:
Thread 3 is executing the global value is 4: local vale is 1:  process id 1994:
Main is executing the global value is 4:  Process ID 1994:
esha@ESHA-DELL:~/1151-lab6$
```

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
27 void *process0(void *arg) {
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 }
```



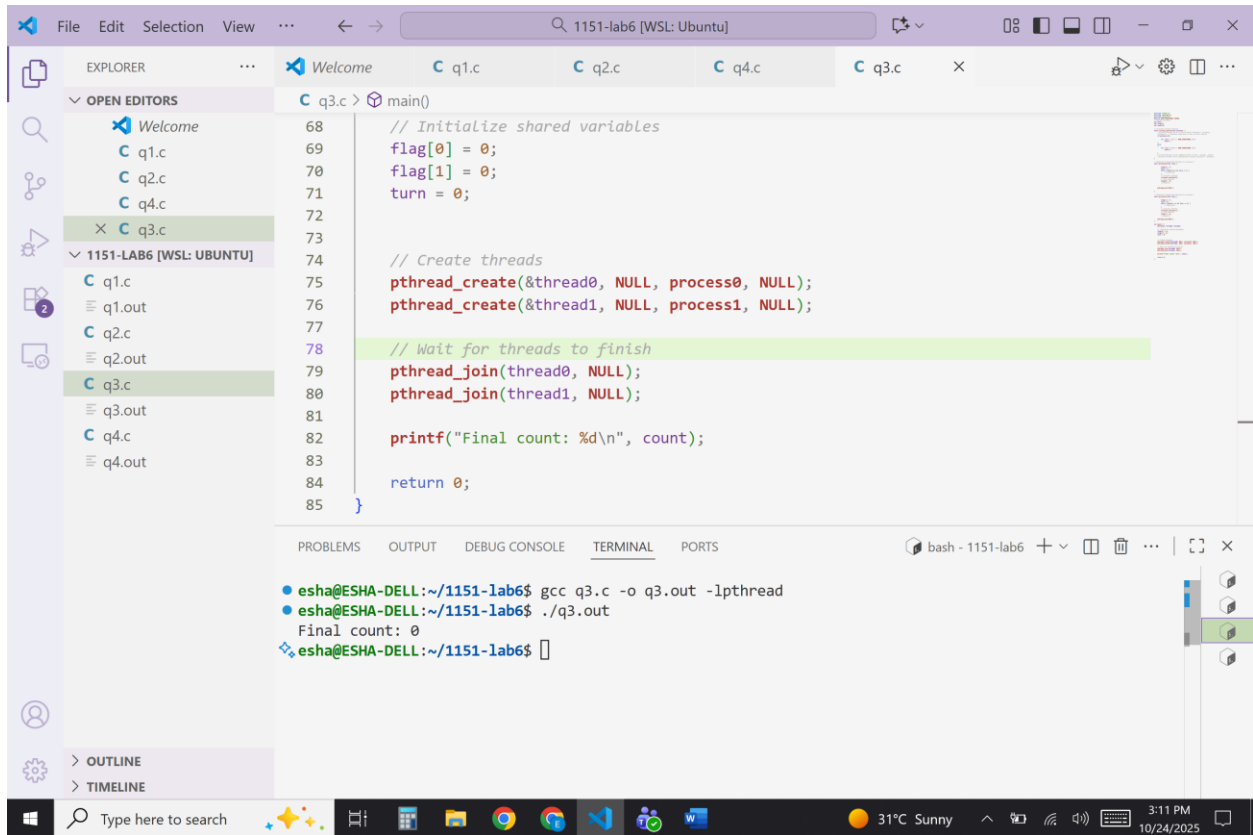
TASK 3:

```

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
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 // Peterson's Algorithm function for process 1
48 void *process1(void *arg) {
49
50     flag[1] = 1;
51     turn = 0;
52     while (flag[0] ==1 && turn == 0) {
53         // Busy wait
54     }
55     // Critical section
56     critical_section(1);
57     // Exit section
58     flag[1] = 0;
59     //sleep(1);
60
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 }

```

TASK 4:




The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the file structure, including a project named '1151-LAB6 [WSL: UBUNTU]' containing files q1.c, q2.c, q3.c, q4.c, and their corresponding .out files. The main editor window shows the code for q3.c, which is a C program using pthreads. The code includes headers for pthread.h and stdio.h, defines shared variables (flag, turn, count), creates two threads (process0 and process1), and waits for them to finish before printing the final count. The terminal at the bottom shows the execution of the program, which outputs 'Final count: 0'.

```
q3.c > main()
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 }
```

```
esha@ESHA-DELL:~/1151-lab6$ gcc q3.c -o q3.out -lpthread
esha@ESHA-DELL:~/1151-lab6$ ./q3.out
Final count: 0
esha@ESHA-DELL:~/1151-lab6$
```

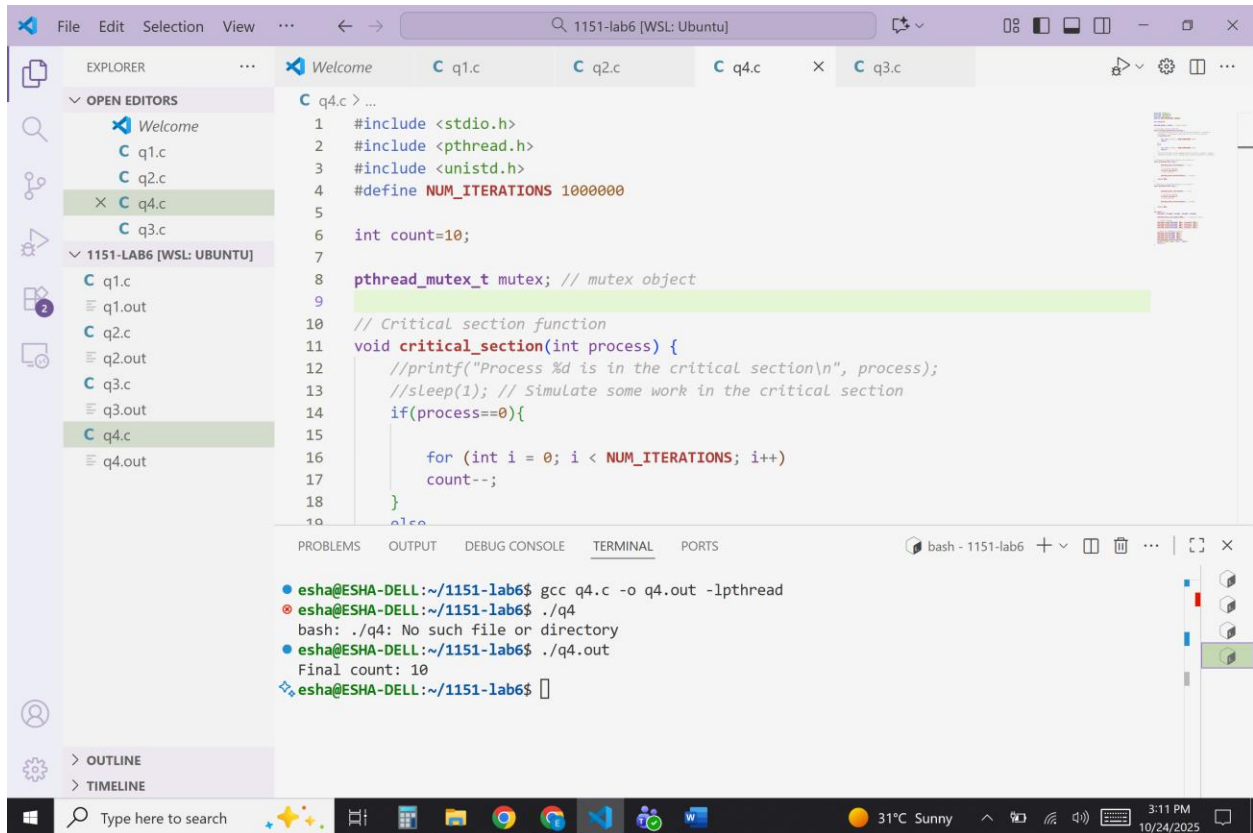
TASK 5:



```

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
46     pthread_mutex_lock(&mutex); // Lock
47
48     // Critical section
49     critical_section(1);
50     // Exit section
51
52     pthread_mutex_unlock(&mutex); // unlock
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     pthread_mutex_destroy(&mutex);
75     printf("Final count: %d\n", count);
76     return 0;
77 }

```



TASK 6:



```

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 decrementers, 3 incrementers)
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 }

```

The screenshot shows the Visual Studio Code editor interface. The Explorer pane on the left lists files in the '1151-LAB6 [WSL: UBUNTU]' workspace, including q1.c, q2.c, q4.c, q4new.c, q3.out, and q3.c. The main editor window displays the code for q4new.c, which includes functions process2, process3, and a main function that creates five threads. The code uses pthread_mutex_t for mutual exclusion and critical sections. The terminal at the bottom shows the execution of the program: 'gcc q4new.c -o q4new.out -lpthread', followed by the command './q4new.out' which outputs 'Final count: 860076'.

```
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     pthread_mutex_init(&mutex, NULL); // initialize mutex
```

```
esha@ESHA-DELL:~/1151-lab6$ gcc q4new.c -o q4new.out -lpthread
esha@ESHA-DELL:~/1151-lab6$ ./q4new.out
Final count: 860076
esha@ESHA-DELL:~/1151-lab6$
```

Difference between Peterson’s Algorithm & Mutex

Feature	Peterson’s Algorithm	Mutex
Type	Software-based algorithm	Hardware/OS-supported mechanism
Number of Processes	Only 2 (basic form)	Any number of threads/processes
Implementation	Uses shared variables (flag, turn)	Uses OS/system calls (pthread_mutex_*)
Busy Waiting	Yes (spinlock)	No (threads sleep if lock unavailable)
Performance	Inefficient (CPU wasting)	Efficient (uses blocking)
Portability	Theoretical / Educational	Real-world use

Feature	Peterson's Algorithm	Mutex
Guarantees	Mutual exclusion, progress, bounded waiting	Mutual exclusion (progress depends on scheduler)
Complexity	Simple, but limited	Abstracted by OS (complex internally)
Use Case	Teaching synchronization concepts	Real-world concurrent programming