



National Textile University
Department of Computer Science

Subject: Operating System

Submitted to: Nasir Mehmood

Submitted by: Nimra Tanveer

Reg number:23-NTU-CS-1201

Semester:5th

Lab manual 06

Task 1:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_THREADS 4
5 int varg=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
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 }
```

The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the following commands and output:

```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task2.c -o task2.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task2.out
Final count: 211687
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task1.c -o task1.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task1.out
Thread 0 is executing the global value is 1: local vale is 1: process id 5975:
Thread 1 is executing the global value is 2: local vale is 1: process id 5975:
Thread 2 is executing the global value is 3: local vale is 1: process id 5975:
Thread 3 is executing the global value is 4: local vale is 1: process id 5975:
Main is executing the global value is 4:: Process ID 5975:
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
```

The Explorer panel on the left shows the file structure of the project, including `task1.c`, `task1.out`, `task2.c`, and `task2.out`. The terminal window is titled `bash - lab6_OS`.

Task 02:

```

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 }

```

The screenshot shows the Visual Studio Code interface with a file explorer on the left displaying a project named 'LAB6_OS [WSL: UBUNTU-22.04]'. The file explorer lists 'task1.c', 'task1.out', 'task2.c', and 'task2.out'. The main editor window shows the code for 'task2.c', which contains a simple C program with a main function that returns 0. The terminal at the bottom shows the execution of the program, including compilation with 'gcc' and the output of the program, which displays thread execution details and a final count of 211687.

```
task2.c
54 int main() {
73     return 0;
74 }
```

```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task2.c -o task2.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task2.out
Final count: 211687
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ gcc task1.c -o task1.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$ ./task1.out
Thread 0 is executing the global value is 1: local vale is 1: process id 5975:
Thread 1 is executing the global value is 2: local vale is 1: process id 5975:
Thread 2 is executing the global value is 3: local vale is 1: process id 5975:
Thread 3 is executing the global value is 4: local vale is 1: process id 5975:
Main is executing the global value is 4:: Process ID 5975:
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/lab6_OS$
```

Task 03:

```

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 }

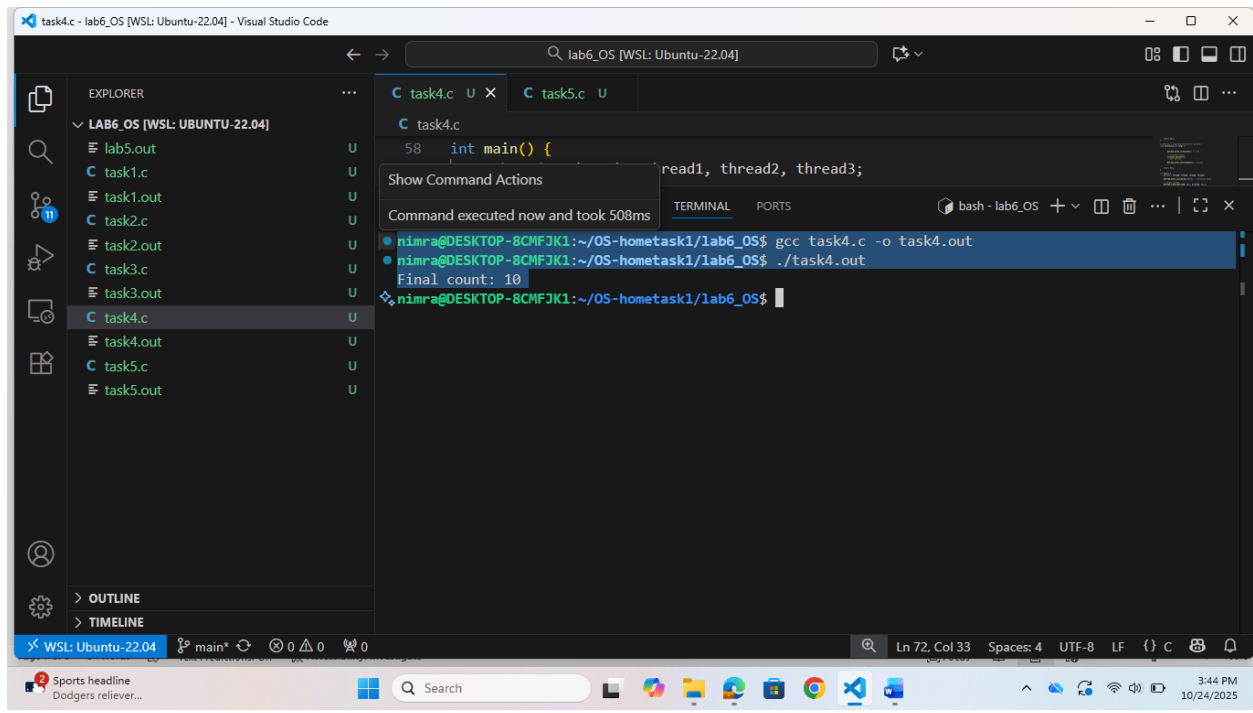
```

Task 04:

```

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
75     pthread_mutex_destroy(&mutex); // destroy mutex
76
77     printf("Final count: %d\n", count);
78
79     return 0;
80 }

```

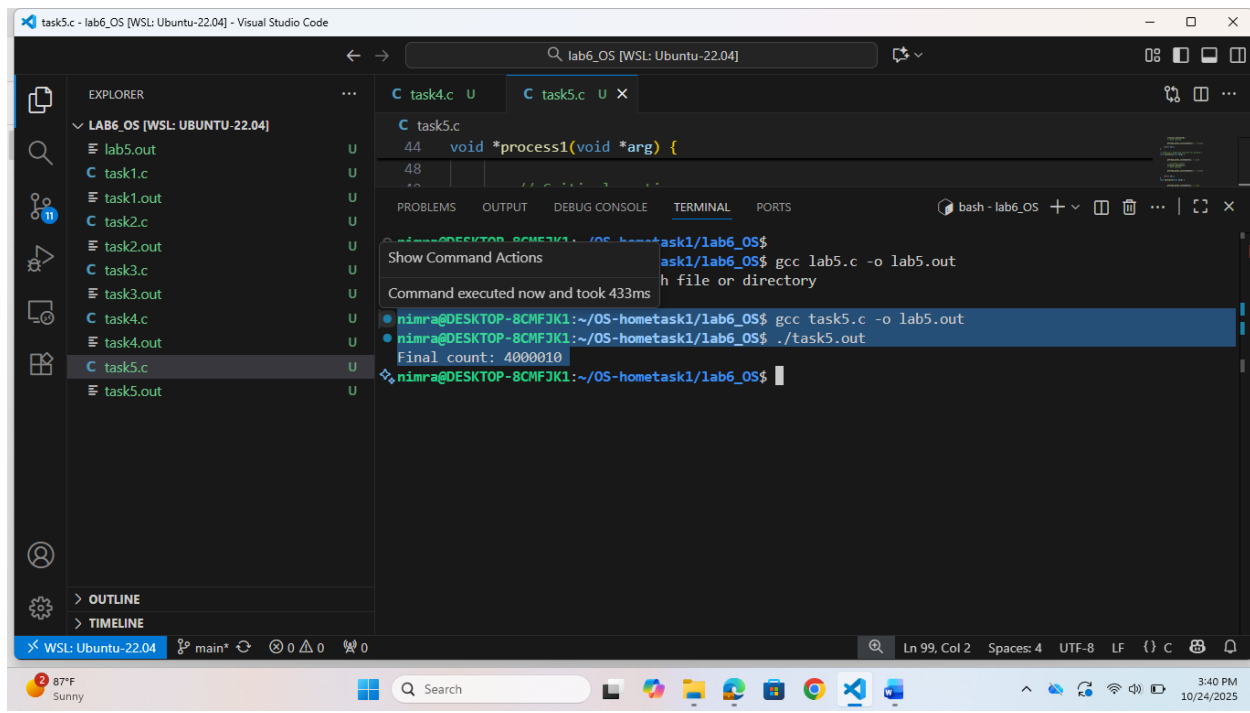



Task 05:

```

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     if (process == 0) {
13         for (int i = 0; i < NUM_ITERATIONS; i++)
14             count--; // Process 0 decreases count
15     }
16     else if (process == 1) {
17         for (int i = 0; i < NUM_ITERATIONS; i++)
18             count++; // Process 1 increases count
19     }
20     else if (process == 2) {
21         for (int i = 0; i < NUM_ITERATIONS; i++)
22             count += 2; // Process 2 adds 2 (you can change logic)
23     }
24 }
25 // Peterson's Algorithm function for process 0
26 void *process0(void *arg) {
27
28     pthread_mutex_lock(&mutex); // lock
29     // mutex lock find kr ky next line ma chla jy ga .nhi mily ga tu mangta rhy ga lock
30     //lock pl or p0 ma sy kisi ik ko mily ga .
31     // jis ko lock mily ga wo run hujay ga proocess or dosra wait ma phas jay ga
32     // jab kam hujay ga process ka tu wo unlock hujay ga or dosra process ko lock mily ga or wo run hujay ga
33
34     // Critical section
35     critical_section(0);
36     // Exit section
37
38     pthread_mutex_unlock(&mutex); // unlock
39
40     return NULL;
41 }
42
43 // Peterson's Algorithm function for process 1
44 void *process1(void *arg) {
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     return NULL;
55 }
56
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
71 }
72
73 int main() {
74     pthread_t thread0, thread1, thread2, thread3, thread4 , thread5;
75
76     pthread_mutex_init(&mutex, NULL); // initialize mutex
77
78     // Create threads
79     pthread_create(&thread0, NULL, process0, NULL);
80     pthread_create(&thread1, NULL, process1, NULL);
81     pthread_create(&thread2, NULL, process0, NULL);
82     pthread_create(&thread3, NULL, process1, NULL);
83     pthread_create(&thread4, NULL, process2, NULL);
84     pthread_create(&thread5, NULL, process2, NULL);
85
86     // Wait for threads to finish
87     pthread_join(thread0, NULL);
88     pthread_join(thread1, NULL);
89     pthread_join(thread2, NULL);
90     pthread_join(thread3, NULL);
91     pthread_join(thread4, NULL);
92     pthread_join(thread5, NULL);
93
94     pthread_mutex_destroy(&mutex); // destroy mutex
95
96     printf("Final count: %d\n", count);
97
98     return 0;
99 }

```



Task 06:

```

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     if (process == 0) {
13         for (int i = 0; i < NUM_ITERATIONS; i++)
14             count--; // Process 0 decreases count
15     }
16     else if (process == 1) {
17         for (int i = 0; i < NUM_ITERATIONS; i++)
18             count++; // Process 1 increases count
19     }
20     else if (process == 2) {
21         for (int i = 0; i < NUM_ITERATIONS; i++)
22             count += 3; // Process 2 adds 2
23     }
24 }
25 // Peterson's Algorithm function for process 0
26 void *process0(void *arg) {
27
28     pthread_mutex_lock(&mutex); // lock
29     // mutex lock find kr ky next line ma chla jy ga .nhi mily ga tu mangta rhy ga lock
30     //lock p1 or p0 ma sy kisi ik ko mily ga .
31     // jis ko lock mily ga wo run hujay ga proocess or dosra wait ma phas jay ga
32     // jab kam hujay ga process ka tu wo unlock hujay ga or dosra process ko lock mily ga or wo run hujay ga
33
34     // Critical section
35     critical_section(0);
36     // Exit section
37
38     pthread_mutex_unlock(&mutex); // unlock
39
40     return NULL;
41 }
42
43 // Peterson's Algorithm function for process 1
44 void *process1(void *arg) {
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     return NULL;
55 }
56
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 }
71
72 int main() {
73     pthread_t thread0, thread1, thread2, thread3, thread4 , thread5;
74
75     pthread_mutex_init(&mutex,NULL); // initialize mutex
76
77     // Create threads
78     pthread_create(&thread0, NULL, process0, NULL);
79     pthread_create(&thread1, NULL, process1, NULL);
80     pthread_create(&thread2, NULL, process0, NULL);
81     pthread_create(&thread3, NULL, process1, NULL);
82     pthread_create(&thread4, NULL, process2, NULL);
83     pthread_create(&thread5, NULL, process2, NULL);
84
85     // Wait for threads to finish
86     pthread_join(thread0, NULL);
87     pthread_join(thread1, NULL);
88     pthread_join(thread2, NULL);
89     pthread_join(thread3, NULL);
90     pthread_join(thread4, NULL);
91     pthread_join(thread5, NULL);
92
93     pthread_mutex_destroy(&mutex); // destroy mutex
94
95     printf("Final count: %d\n", count);
96
97     return 0;
98 }
99

```

The screenshot shows a Visual Studio Code editor window with a file explorer on the left listing files like lab5.out, task1.c, task2.c, task3.c, task4.c, task5.c, task6.c, and task6.out. The main editor displays the code for task6.c, which includes a main function that creates three threads (pthread3, process1, process2) and prints the final count. A terminal window at the bottom shows the command prompt 'nirm@DESKTOP-8CMF3K1:~/OS-hometask1/lab6_OS\$' and the execution of 'gcc task6.c -o task6.out' and './task6.out', outputting 'Final count: 3762846'.

Difference between Peterson’s Algorithm and Mutex Algorithm?

Feature	Peterson’s Algorithm	Mutex Algorithm
Type	Software-based algorithm	Hardware/OS-supported mechanism
Implementation	Uses shared variables (flag, turn)	Uses OS/system calls (pthread_mutex_*)
Performance	Inefficient (CPU wasting)	Efficient (uses blocking)
Portability	Theoretical / Educational	Real-world use
Complexity	Simple, but limited	Abstracted by OS (complex)
Use Case	Teaching	Real-world concurrent

	synchronization concepts	programming
--	-----------------------------	-------------