



National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Maha

Reg. number: 23-NTU-CS-1170

Lab 6 (class task)

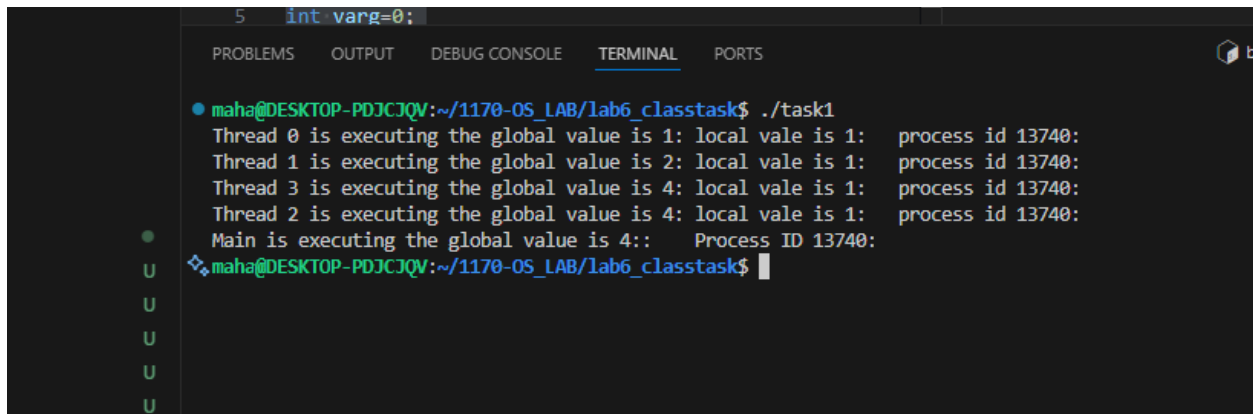
Semester: 5th

Task 1:

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         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18     else
19     {
20         for (int i = 0; i < NUM_ITERATIONS; i++)
21             count++;
22     }
23     //printf("Process %d has updated count to %d\n", process, count);
24     //printf("Process %d is leaving the critical section\n", process);
25 }
26
27 // Peterson's Algorithm function for process 0
28 void *process0(void *arg) {
29
30     pthread_mutex_lock(&mutex); // lock
31
32     // Critical section
33     critical_section(0);
34     // Exit section
35
36     pthread_mutex_unlock(&mutex); // unlock
37
38     return NULL;
39 }
40
41 // Peterson's Algorithm function for process 1
42 void *process1(void *arg) {
43
44     pthread_mutex_lock(&mutex); // lock
45
46     // Critical section
47     critical_section(1);
48     // Exit section
49
50     pthread_mutex_unlock(&mutex); // unlock
51
52     return NULL;
53 }
54
55
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 }
```

Terminal:



```
5 int varg=0;

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task1
Thread 0 is executing the global value is 1: local vale is 1: process id 13740:
Thread 1 is executing the global value is 2: local vale is 1: process id 13740:
Thread 3 is executing the global value is 4: local vale is 1: process id 13740:
Thread 2 is executing the global value is 4: local vale is 1: process id 13740:
Main is executing the global value is 4:: Process ID 13740:
❖ maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$
```

Question 2:

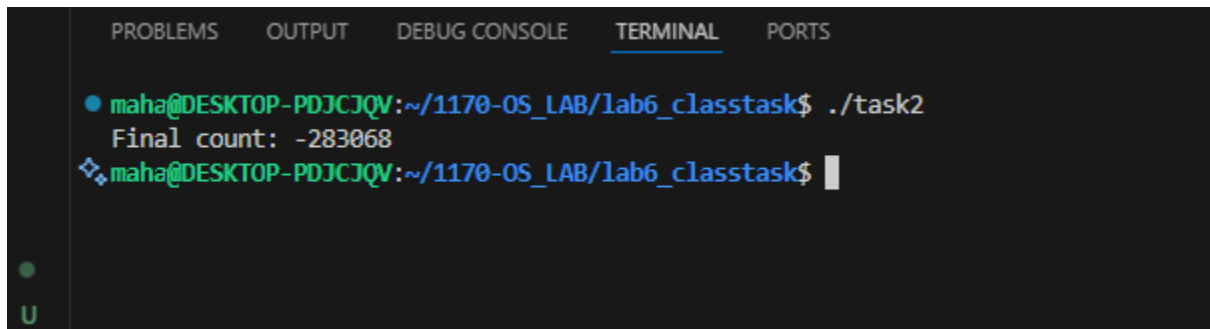
Code:

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4
5  #define NUM_ITERATIONS 1000000
6  int count=10;
7  // Critical section function
8  void critical_section(int process) {
9      //printf("Process %d is in the critical section\n", process);
10     //sleep(1); // Simulate some work in the critical section
11     if(process==0){
12         for (int i = 0; i < NUM_ITERATIONS; i++)
13             count--;
14     }
15     else
16     {
17         for (int i = 0; i < NUM_ITERATIONS; i++)
18             count++;
19     }
20 }
21
22 void *process0(void *arg) {
23     // Critical section
24     critical_section(0);
25     // Exit section
26     return NULL;
27 }
28 void *process1(void *arg) {
29     // Critical section
30     critical_section(1);
31     // Exit section
32     return NULL;
33 }
34 int main() {
35     pthread_t thread0, thread1, thread2, thread3;
36     // Create threads
37     pthread_create(&thread0, NULL, process0, NULL);
38     pthread_create(&thread1, NULL, process1, NULL);
39     pthread_create(&thread2, NULL, process0, NULL);
40     pthread_create(&thread3, NULL, process1, NULL);
41     // Wait for threads to finish
42     pthread_join(thread0, NULL);
43     pthread_join(thread1, NULL);
44     pthread_join(thread2, NULL);
45     pthread_join(thread3, NULL);
46     printf("Final count: %d\n", count);
47     return 0;
48 }

```

Terminal:



A screenshot of a terminal window with a dark background. At the top, there is a horizontal menu bar with five items: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is underlined), and 'PORTS'. Below the menu bar, the terminal shows a prompt 'maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask\$' followed by the command './task2'. The output of the command is 'Final count: -283068'. Below the output, the prompt is shown again: 'maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask\$' with a cursor. On the left side of the terminal window, there is a vertical sidebar with a green dot and the letter 'U' below it.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
● maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task2  
Final count: -283068  
❖ maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$
```

Question 3:

Peterson algorithm:

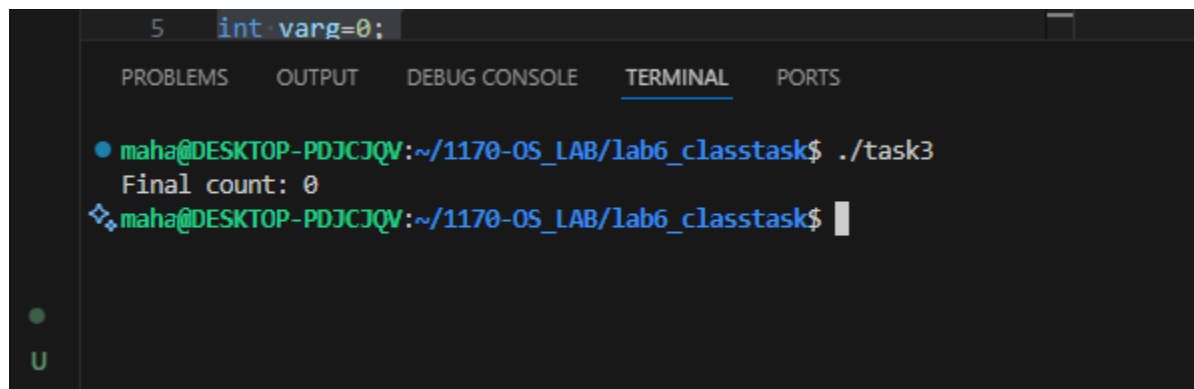
Code:

```

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  // Critical section function
10 void critical_section(int process) {
11     //printf("Process %d is in the critical section\n", process);
12     //sleep(1); // Simulate some work in the critical section
13     if(process==0){
14
15         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18     else
19     {
20         for (int i = 0; i < NUM_ITERATIONS; i++)
21             count++;
22     }
23     // printf("Process %d has updated count to %d\n", process, count);
24     //printf("Process %d is leaving the critical section\n", process);
25 }
26
27 // Peterson's Algorithm function for process 0
28 void *process0(void *arg) {
29     flag[0] = 1;
30     turn = 1;
31     while (flag[1]==1 && turn == 1) {
32         // Busy wait
33     }
34     // Critical section
35     critical_section(0);
36     // Exit section
37     flag[0] = 0;
38     //sleep(1);
39     pthread_exit(NULL);
40 }
41
42 // Peterson's Algorithm function for process 1
43 void *process1(void *arg) {
44
45     flag[1] = 1;
46     turn = 0;
47     while (flag[0] ==1 && turn == 0) {
48         // Busy wait
49     }
50     // Critical section
51     critical_section(1);
52     // Exit section
53     flag[1] = 0;
54     //sleep(1);
55     pthread_exit(NULL);
56 }
57 int main() {
58     pthread_t thread0, thread1;
59     // Initialize shared variables
60     flag[0] = 0;
61     flag[1] = 0;
62     turn = 0;
63     // Create threads
64     pthread_create(&thread0, NULL, process0, NULL);
65     pthread_create(&thread1, NULL, process1, NULL);
66     // Wait for threads to finish
67     pthread_join(thread0, NULL);
68     pthread_join(thread1, NULL);
69     printf("Final count: %d\n", count);
70     return 0;
71 }

```

Terminal:



The image shows a screenshot of a terminal window with a dark background. At the top, there is a tab labeled '5' and a code snippet 'int varg=0;'. Below the tab, there are several tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The terminal content shows a prompt 'maha@DESKTOP-PDJCJQV:~/1170-05_LAB/lab6_classtask\$' followed by the command './task3'. The output of the command is 'Final count: 0'. Below the output, there is another prompt 'maha@DESKTOP-PDJCJQV:~/1170-05_LAB/lab6_classtask\$' with a cursor. On the left side of the terminal window, there is a green dot and the letter 'U'.

```
5  int varg=0;

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● maha@DESKTOP-PDJCJQV:~/1170-05_LAB/lab6_classtask$ ./task3
  Final count: 0
❖ maha@DESKTOP-PDJCJQV:~/1170-05_LAB/lab6_classtask$
```

Question 4:

mutex locks

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  // Critical section function
10 void critical_section(int process) {
11     //printf("Process %d is in the critical section\n", process);
12     //sleep(1); // Simulate some work in the critical section
13     if(process==0){
14
15         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18     else
19     {
20         for (int i = 0; i < NUM_ITERATIONS; i++)
21             count++;
22     }
23     // printf("Process %d has updated count to %d\n", process, count);
24     //printf("Process %d is leaving the critical section\n", process);
25 }
26
27 // Peterson's Algorithm function for process 0
28 void *process0(void *arg) {
29     flag[0] = 1;
30     turn = 1;
31     while (flag[1]==1 && turn == 1) {
32         // Busy wait
33     }
34     // Critical section
35     critical_section(0);
36     // Exit section
37     flag[0] = 0;
38     //sleep(1);
39     pthread_exit(NULL);
40 }
41
42 // Peterson's Algorithm function for process 1
43 void *process1(void *arg) {
44
45     flag[1] = 1;
46     turn = 0;
47     while (flag[0] ==1 && turn == 0) {
48         // Busy wait
49     }
50     // Critical section
51     critical_section(1);
52     // Exit section
53     flag[1] = 0;
54     //sleep(1);
55     pthread_exit(NULL);
56 }
57 int main() {
58     pthread_t thread0, thread1;
59     // Initialize shared variables
60     flag[0] = 0;
61     flag[1] = 0;
62     turn = 0;
63     // Create threads
64     pthread_create(&thread0, NULL, process0, NULL);
65     pthread_create(&thread1, NULL, process1, NULL);
66     // Wait for threads to finish
67     pthread_join(thread0, NULL);
68     pthread_join(thread1, NULL);
69     printf("Final count: %d\n", count);
70     return 0;
71 }

```

Terminal:


```
6 int count=10;  
7  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
● maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task4  
Final count: 10  
❖ maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$
```

Question 5:

Do task4 with three process

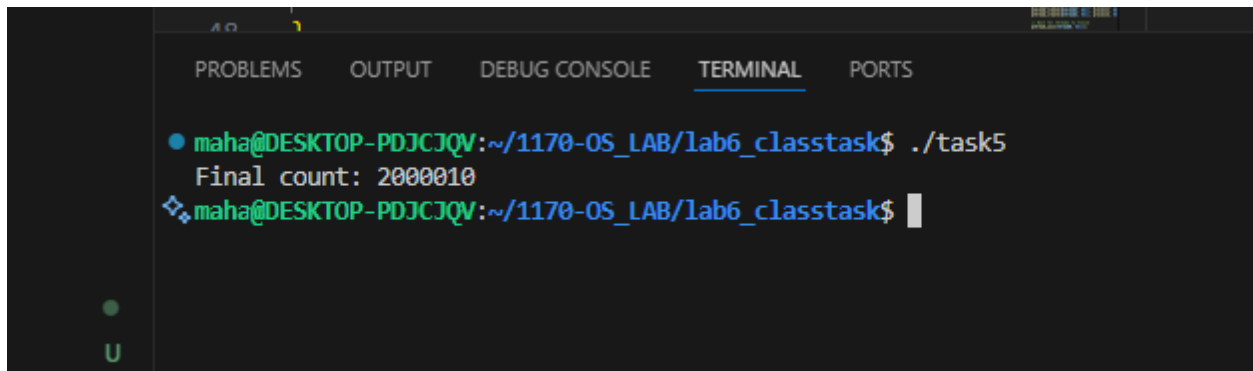
Code:

```

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  // Critical section function
10 void critical_section(int process) {
11     //printf("Process %d is in the critical section\n", process);
12     //sleep(1); // Simulate some work in the critical section
13     if(process==0){
14
15         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18     else
19     {
20         for (int i = 0; i < NUM_ITERATIONS; i++)
21             count++;
22     }
23     // printf("Process %d has updated count to %d\n", process, count);
24     //printf("Process %d is leaving the critical section\n", process);
25 }
26
27 // Peterson's Algorithm function for process 0
28 void *process0(void *arg) {
29     flag[0] = 1;
30     turn = 1;
31     while (flag[1]==1 && turn == 1) {
32         // Busy wait
33     }
34     // Critical section
35     critical_section(0);
36     // Exit section
37     flag[0] = 0;
38     //sleep(1);
39     pthread_exit(NULL);
40 }
41
42 // Peterson's Algorithm function for process 1
43 void *process1(void *arg) {
44
45     flag[1] = 1;
46     turn = 0;
47     while (flag[0] ==1 && turn == 0) {
48         // Busy wait
49     }
50     // Critical section
51     critical_section(1);
52     // Exit section
53     flag[1] = 0;
54     //sleep(1);
55     pthread_exit(NULL);
56 }
57 int main() {
58     pthread_t thread0, thread1;
59     // Initialize shared variables
60     flag[0] = 0;
61     flag[1] = 0;
62     turn = 0;
63     // Create threads
64     pthread_create(&thread0, NULL, process0, NULL);
65     pthread_create(&thread1, NULL, process1, NULL);
66     // Wait for threads to finish
67     pthread_join(thread0, NULL);
68     pthread_join(thread1, NULL);
69     printf("Final count: %d\n", count);
70     return 0;
71 }

```

Terminal:



```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$ ./task5
Final count: 2000010
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/lab6_classtask$
```

Compare Peterson and mutex locks:

| Feature | Peterson's Algorithm | Mutex Lock |
|---------------------|---------------------------|---------------------------|
| Type | Software-based | Hardware/library-based |
| Processes Supported | Only 2 | Multiple |
| Implementation | Manual (flags & turn) | Simple API (lock, unlock) |
| Performance | Slow (busy waiting) | Fast (no busy waiting) |
| Use | Theoretical / Educational | Practical / Real-world |
| Fairness | Ensured | Depends on system |
| Deadlock Risk | No (if correct) | Yes (if misused) |