



National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Amina

Reg. number: 23-NTU-CS-1136

Lab no.: lab6

Semester:5th

Task1:

Code:

```
1 #include <stdio.h>
2 #include <pthread.h>
3
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 }
```

Terminal:

```
● amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ ./task1
Thread 0 is executing the global value is 1: local vale is 1:   process id 4697:
Thread 1 is executing the global value is 2: local vale is 1:   process id 4697:
Thread 2 is executing the global value is 3: local vale is 1:   process id 4697:
Thread 3 is executing the global value is 4: local vale is 1:   process id 4697:
Main is executing the global value is 4::   Process ID 4697:
```

Task2:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

int count=10;

// Critical section function
void critical_section(int process) {
    //printf("Process %d is in the critical section\n", process);
    //sleep(1); // Simulate some work in the critical section
    if(process==0){
```

```

        for (int i = 0; i < NUM_ITERATIONS; i++)
            count--;
    }
    else
    {
        for (int i = 0; i < NUM_ITERATIONS; i++)
            count++;
    }
}

void *process0(void *arg) {

    // Critical section
    critical_section(0);
    // Exit section

    return NULL;
}

void *process1(void *arg) {

    // Critical section
    critical_section(1);
    // Exit section

    return NULL;
}

int main() {
    pthread_t thread0, thread1, thread2, thread3;

    // Create threads
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process0, NULL);
    pthread_create(&thread3, NULL, process1, NULL);

```

```
// Wait for threads to finish
pthread_join(thread0, NULL);
pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
pthread_join(thread3, NULL);

printf("Final count: %d\n", count);

return 0;
}
```

Terminal:

```
● amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ gcc Task2.c -o task2 -lpthread
● amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ ./task2
Final count: -28292
● amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ ./task2
Final count: 16439
```

Task3:

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     // 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     flag[0] = 1;
32     turn = 1;
33     while (flag[1]==1 && turn == 1) {
34         // Busy wait
35     }
36     // Critical section
37     critical_section(0);
38     // Exit section
39     flag[0] = 0;
40     //sleep(1);
41
42     pthread_exit(NULL);
43 }
44
45 // Peterson's Algorithm function for process 1
46 void *process1(void *arg) {
47
48     flag[1] = 1;
49     turn = 0;
50     while (flag[0] ==1 && turn == 0) {
51         // Busy wait
52     }
53     // Critical section
54     critical_section(1);
55     // Exit section
56     flag[1] = 0;
57     //sleep(1);
58
59     pthread_exit(NULL);
60 }
61
62 int main() {
63     pthread_t thread0, thread1;
64
65     // Initialize shared variables
66     flag[0] = 0;
67     flag[1] = 0;
68     turn = 0;
69
70     // Create threads
71     pthread_create(&thread0, NULL, process0, NULL);
72     pthread_create(&thread1, NULL, process1, NULL);
73
74     // Wait for threads to finish
75     pthread_join(thread0, NULL);
76     pthread_join(thread1, NULL);
77
78     printf("Final count: %d\n", count);
79
80     return 0;
81 }

```

Terminal:

```
• amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ gcc Task3.c -o task3 -lpthread
• amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ ./task3
Final count: 0
```

Task4:

```

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 }

```

Terminal:

```
amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ gcc Task4.c -o task4 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ .
./
.n8n-7ptIYyXf .n8n.cmd-nPL5YlWL .n8n.ps1-Jupiu3j8
amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ ./task4
Final count: 10
amina@DESKTOP-SEP18NK:~/OSLabs/lab6$
```

Task4Update:

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{
25         for (int i = 0; i < NUM_ITERATIONS; i++)
26             count+=2;
27     }
28     //printf("Process %d has updated count to %d\n", process, count);
29     //printf("Process %d is leaving the critical section\n", process);
30 }
31
32 // Peterson's Algorithm function for process 0
33 void *process0(void *arg) {
34
35     pthread_mutex_lock(&mutex); // lock
36
37     // Critical section
38     critical_section(0);
39     // Exit section
40
41     pthread_mutex_unlock(&mutex); // unlock
42
43     return NULL;
44 }
45
46 // Peterson's Algorithm function for process 1
47 void *process1(void *arg) {
48
49     pthread_mutex_lock(&mutex); // lock
50
51     // Critical section
52     critical_section(1);
53     // Exit section
54
55     pthread_mutex_unlock(&mutex); // unlock
56
57     return NULL;
58 }
59
60 void *process2(void *arg) {
61
62     pthread_mutex_lock(&mutex); // lock
63
64     // Critical section
65     critical_section(2);
66     // Exit section
67
68     pthread_mutex_unlock(&mutex); // unlock
69
70     return NULL;
71 }
72
73
74 }
75
76 int main() {
77     pthread_t thread0, thread1, thread2, thread3;
78
79     pthread_mutex_init(&mutex, NULL); // initialize mutex
80
81     // Create threads
82     pthread_create(&thread0, NULL, process0, NULL);
83     pthread_create(&thread1, NULL, process1, NULL);
84     pthread_create(&thread2, NULL, process0, NULL);
85     pthread_create(&thread3, NULL, process1, NULL);
86     pthread_create(&thread2, NULL, process0, NULL);
87     pthread_create(&thread3, NULL, process2, NULL);
88
89     // Wait for threads to finish
90     pthread_join(thread0, NULL);
91     pthread_join(thread1, NULL);
92     pthread_join(thread2, NULL);
93     pthread_join(thread3, NULL);
94
95     pthread_mutex_destroy(&mutex); // destroy mutex
96
97     printf("Final count: %d\n", count);
98
99     return 0;
100 }

```

Terminal:

```
● amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ gcc Task4Update.c -o task4Update -lpthread
● amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ ./task4Update
Final count: 1000010
○ amina@DESKTOP-SEP18NK:~/OSLabs/lab6$ █
⊗ 0 △ 0 (A) 0
```

Compare Peterson and mutex lock:

Feature	Peterson's Algorithm	Mutex Lock
Type	Software-based (uses shared variables like <code>flag</code> and <code>turn</code>)	Hardware/OS-based (uses system-level synchronization primitives)
Threads supported	Works for 2 threads only (basic version)	Works for many threads
Efficiency	Slow , uses busy waiting and consumes CPU time	Fast , threads sleep while waiting — no CPU wastage
Reliability	May fail on modern CPUs due to instruction reordering	Highly reliable , supported by OS and hardware
Ease of use	Harder to implement and understand	Very easy to use — just <code>lock()</code> and <code>unlock()</code>
Usage	Mostly educational or theoretical	Practical and widely used in real-world programming
Practicality	Not used in modern systems	Highly practical — used in multithreading, databases, and OS kernels