



National Textile University

Department of Computer Science

Subject:
Operating System

Submitted to:
Sir Nasir

Submitted by:
Hafsa Amjad

Reg number:
23-NTU-CS-1162

Lab no:

6

Semester:
5th

Task 1:

```
#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#define NUM_THREADS 4

int varg = 0;

void *thread_function(void *arg) {
    int thread_id = *(int *)arg;

    int varl = 0;

    varg++; // global variable increment
    varl++; // local variable increment

    printf("Thread %d is executing. Global value = %d | Local value = %d | Process ID = %d\n",
        thread_id, varg, varl, getpid());

    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_args[NUM_THREADS];

    for (int i = 0; i < NUM_THREADS; ++i) {
        thread_args[i] = i;

        pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
    }

    for (int i = 0; i < NUM_THREADS; ++i) {
```

```

pthread_join(threads[i], NULL);
}

printf("Main is executing. Final Global value = %d | Process ID = %d\n", varg, getpid());

return 0;
}

```

The screenshot shows a Visual Studio Code editor window with the following content:

Task1.c

```

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h> // getpid() ke liye include karna zaroori hai
4
5 #define NUM_THREADS 4
6 int varg = 0;
7
8 void *thread_function(void *arg) {
9     int thread_id = *(int *)arg;
10
11     int varl = 0;
12     varg++; // global variable increment
13     varl++; // local variable increment
14     printf("Thread %d is executing. Global value = %d | Local value = %d | Process ID = %d\n",
15           thread_id, varg, varl, getpid());
16     return NULL;
17 }
18
19 int main() {
20     pthread_t threads[NUM_THREADS];
21     int thread_args[NUM_THREADS];

```

Terminal Output:

```

hafsai1162@DESKTOP-LA005TK:~/Lab_6$ ./output4
Final count: 4000010
hafsai1162@DESKTOP-LA005TK:~/Lab_6$ gcc Task1.c -o output -lpthread
hafsai1162@DESKTOP-LA005TK:~/Lab_6$ ./output
Thread 0 is executing. Global value = 1 | Local value = 1 | Process ID = 13971
Thread 1 is executing. Global value = 2 | Local value = 1 | Process ID = 13971
Thread 2 is executing. Global value = 3 | Local value = 1 | Process ID = 13971
Thread 3 is executing. Global value = 4 | Local value = 1 | Process ID = 13971
Main is executing. Final Global value = 4 | Process ID = 13971
hafsai1162@DESKTOP-LA005TK:~/Lab_6$

```

Task 2:

```

#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;
```

```
}
```

```
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++)
```

```
hafs@1162@DESKTOP-LA0D5TK:~/Lab_6$ gcc Task2.c -o task2 -lpthread
hafs@1162@DESKTOP-LA0D5TK:~/Lab_6$ ./task2
Final count: -41774
hafs@1162@DESKTOP-LA0D5TK:~/Lab_6$
```

Task 3:

```
#include <stdio.h>
```

```
#include <pthread.h>
```

```
#include <unistd.h>
```

```
#define NUM_ITERATIONS 100000
```

```
// Shared variables
```

```
int turn;
```

```

int flag[2];
int count=0;

// 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++;

    }
    // printf("Process %d has updated count to %d\n", process, count);
    //printf("Process %d is leaving the critical section\n", process);
}

```

```

// Peterson's Algorithm function for process 0

```

```

void *process0(void *arg) {

    flag[0] = 1;
    turn = 1;
    while (flag[1]==1 && turn == 1) {
        // Busy wait
    }
}

```

```

    }

    // Critical section

    critical_section(0);

    // Exit section

    flag[0] = 0;

    //sleep(1);


pthread_exit(NULL);

}


// Peterson's Algorithm function for process 1
void *process1(void *arg) {

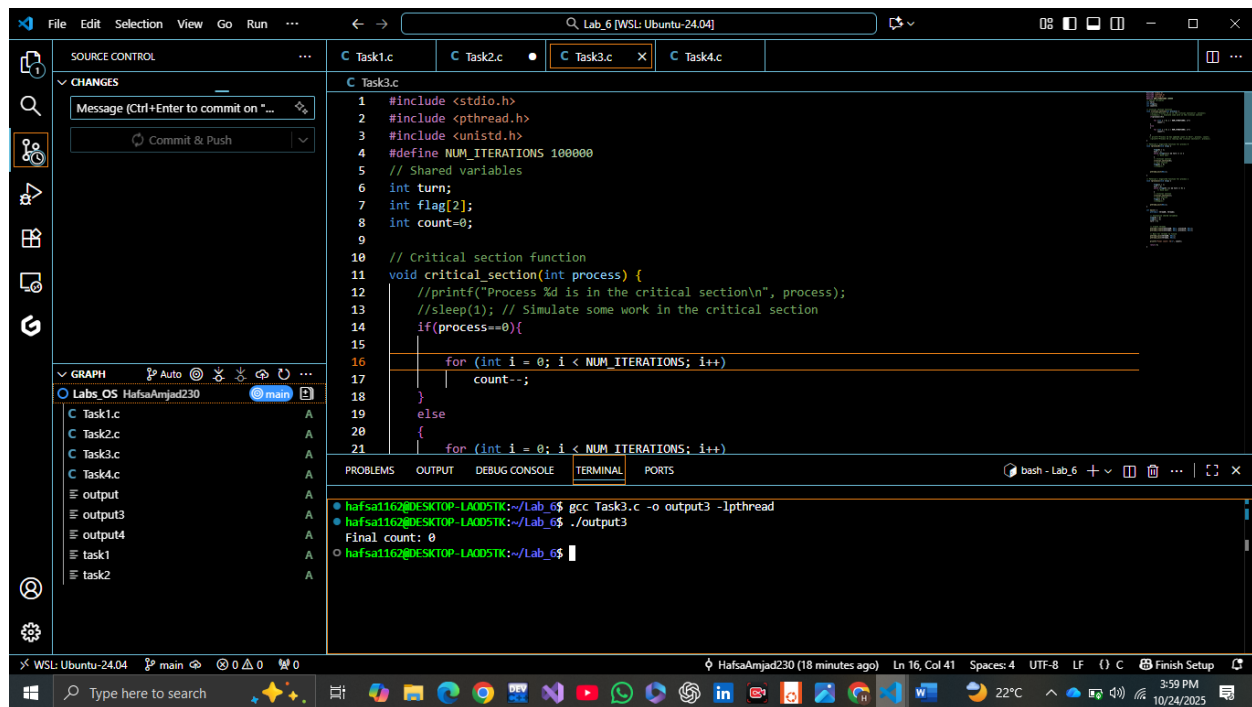
    flag[1] = 1;
    turn = 0;
    while (flag[0] == 1 && turn == 0) {
        // Busy wait
    }
    // Critical section
    critical_section(1);
    // Exit section
    flag[1] = 0;
    //sleep(1);


pthread_exit(NULL);
}

```



```
int main() {  
    pthread_t thread0, thread1;  
  
    // Initialize shared variables  
    flag[0] = 0;  
    flag[1] = 0;  
    turn = 0;  
  
    // Create threads  
    pthread_create(&thread0, NULL, process0, NULL);  
    pthread_create(&thread1, NULL, process1, NULL);  
  
    // Wait for threads to finish  
    pthread_join(thread0, NULL);  
    pthread_join(thread1, NULL);  
  
    printf("Final count: %d\n", count);  
  
    return 0;  
}
```



Task 4:

```
#include <stdio.h>
```

```
#include <pthread.h>
```

```
#include <unistd.h>
```

```
#define NUM_ITERATIONS 1000000
```

```
int count = 10;
```

```
pthread_mutex_t mutex; // mutex object
```

```
// Critical section function
```

```
void critical_section(int process) {
```

```
    if (process == 0) {
```

```
        for (int i = 0; i < NUM_ITERATIONS; i++)
```

```
            count--;
```


```
}  
  
else if (process == 1) {  
    for (int i = 0; i < NUM_ITERATIONS; i++)  
        count++;  
}  
  
else if (process == 2) {  
    for (int i = 0; i < NUM_ITERATIONS; i++)  
        count += 2; // third process modifies differently  
}  
}
```

// Process 0

```
void *process0(void *arg) {  
    pthread_mutex_lock(&mutex); // lock  
    critical_section(0);  
    pthread_mutex_unlock(&mutex); // unlock  
    return NULL;  
}
```

// Process 1

```
void *process1(void *arg) {  
    pthread_mutex_lock(&mutex);  
    critical_section(1);  
    pthread_mutex_unlock(&mutex);  
    return NULL;  
}
```

//  Process 2 (newly added)

```
void *process2(void *arg) {
```

```

pthread_mutex_lock(&mutex);
critical_section(2);
pthread_mutex_unlock(&mutex);
return NULL;
}

int main() {
    pthread_t thread0, thread1, thread2, thread3, thread4, thread5;

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

    // Create threads for all processes
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process2, NULL);
    pthread_create(&thread3, NULL, process0, NULL);
    pthread_create(&thread4, NULL, process1, NULL);
    pthread_create(&thread5, NULL, process2, NULL);

    // Wait for all threads to complete
    pthread_join(thread0, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);
    pthread_join(thread5, NULL);

    pthread_mutex_destroy(&mutex); // destroy mutex

```

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

```
return 0;
```

```
}
```

The screenshot displays the Visual Studio Code interface with a C program using pthreads. The code is as follows:

```
50 int main() {
51     pthread_t thread0, thread1, thread2, thread3, thread4, thread5;
52
53     pthread_mutex_init(&mutex, NULL); // initialize mutex
54
55     // Create threads for all processes
56     pthread_create(&thread0, NULL, process0, NULL);
57     pthread_create(&thread1, NULL, process1, NULL);
58     pthread_create(&thread2, NULL, process2, NULL);
59     pthread_create(&thread3, NULL, process0, NULL);
60     pthread_create(&thread4, NULL, process1, NULL);
61     pthread_create(&thread5, NULL, process2, NULL);
62
63     // Wait for all threads to complete
64     pthread_join(thread0, NULL);
65     pthread_join(thread1, NULL);
66     pthread_join(thread2, NULL);
67     pthread_join(thread3, NULL);
68     pthread_join(thread4, NULL);
69     pthread_join(thread5, NULL);
70 }
```

The terminal output shows the compilation and execution of Task3.c and Task4.c:

```
hafsai162@DESKTOP-LAC05TK:~/Lab_6$ gcc Task3.c -o output3 -lpthread
hafsai162@DESKTOP-LAC05TK:~/Lab_6$ ./output3
Final count: 0
hafsai162@DESKTOP-LAC05TK:~/Lab_6$ gcc Task4.c -o output4 -lpthread
hafsai162@DESKTOP-LAC05TK:~/Lab_6$ ./output4
Final count: 4800010
hafsai162@DESKTOP-LAC05TK:~/Lab_6$
```