

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 with a C program in a file named `task1.c`. The program is designed to demonstrate thread synchronization using pthreads. It includes `<stdio.h>`, `<pthread.h>`, and `<unistd.h>` for the `getpid()` function. A macro `NUM_THREADS 4` is defined. A global variable `varg` is initialized to 0, and a `pthread_mutex_t` lock is declared. The `thread_function` takes a void pointer `arg` and casts it to an integer. It increments the global `varg` while holding the mutex lock and prints a message showing the thread ID, global value, local value, and process ID. The main function creates four threads, each calling `thread_function` with its own ID. After joining all threads, it prints the final global value and its process ID.

The terminal output shows the execution of the program. It starts with the command `cd ../lab6`, then `gcc task1.c -o out1`, and finally `./out1`. The output shows four threads executing in parallel, each incrementing the global value. The final output is: "Main is executing. Final Global value = 4 | Process ID = 5734".

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;
}

```

The screenshot shows a Visual Studio Code editor with a C program in a file named `task2.c`. The code defines a critical section function and uses `pthread_join` to wait for three threads. The terminal output shows the final count as -634474.

```

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 1000000
5
6 int count=10;
7
8 // Critical section function
9 void critical_section(int process) {
10     //printf("Process %d is in the critical section\n", process);
11     //sleep(1); // Simulate some work in the critical section
12     if(process==0){
13         for (int i = 0; i < NUM_ITERATIONS; i++)
14             count--;
15     }
16     else
17     {
18         for (int i = 0; i < NUM_ITERATIONS; i++)
19             count++;
20     }
21 }

```

The terminal output shows the final count as -634474.

```

* fatimall153@DESKTOP-RMTBLS:~/OperatingSystemLabs/lab6$ gcc task2.c -o out2
* fatimall153@DESKTOP-RMTBLS:~/OperatingSystemLabs/lab6$ ./out2
Final count: -634474
* fatimall153@DESKTOP-RMTBLS:~/OperatingSystemLabs/lab6$

```

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;

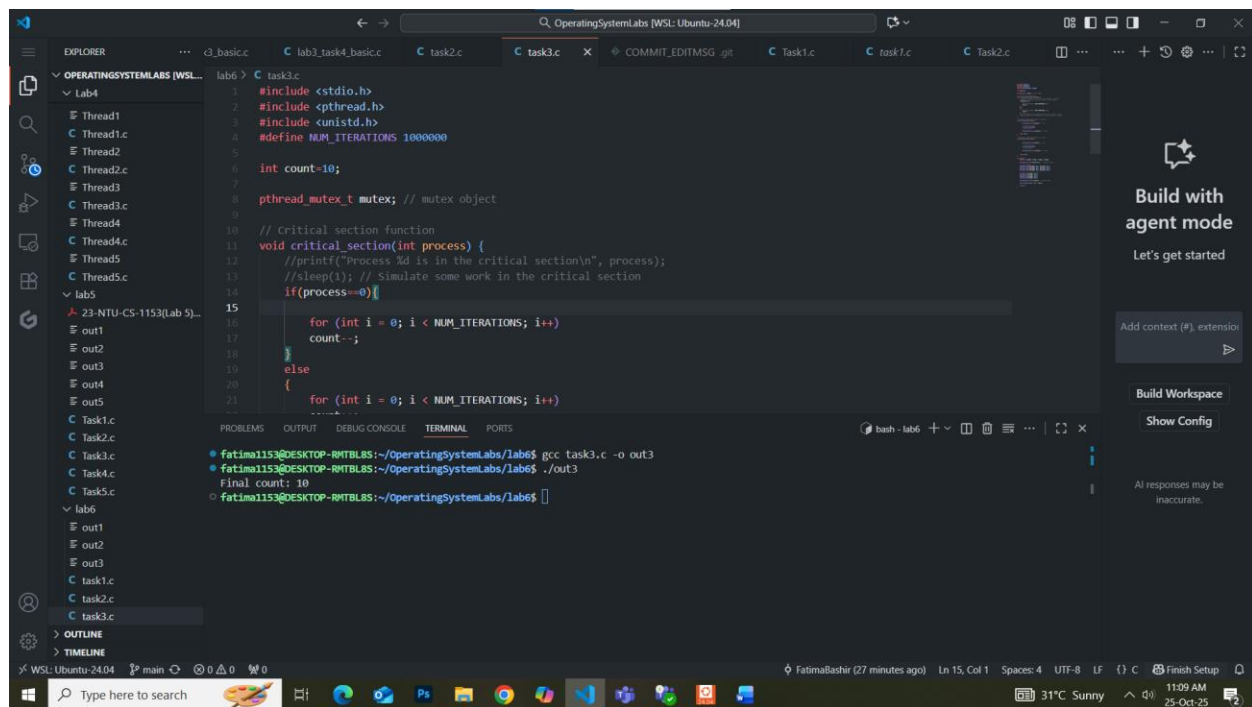
// Initialize shared variables
flag[0] = 0;
flag[1] = 0;
turn = 0;

// 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);

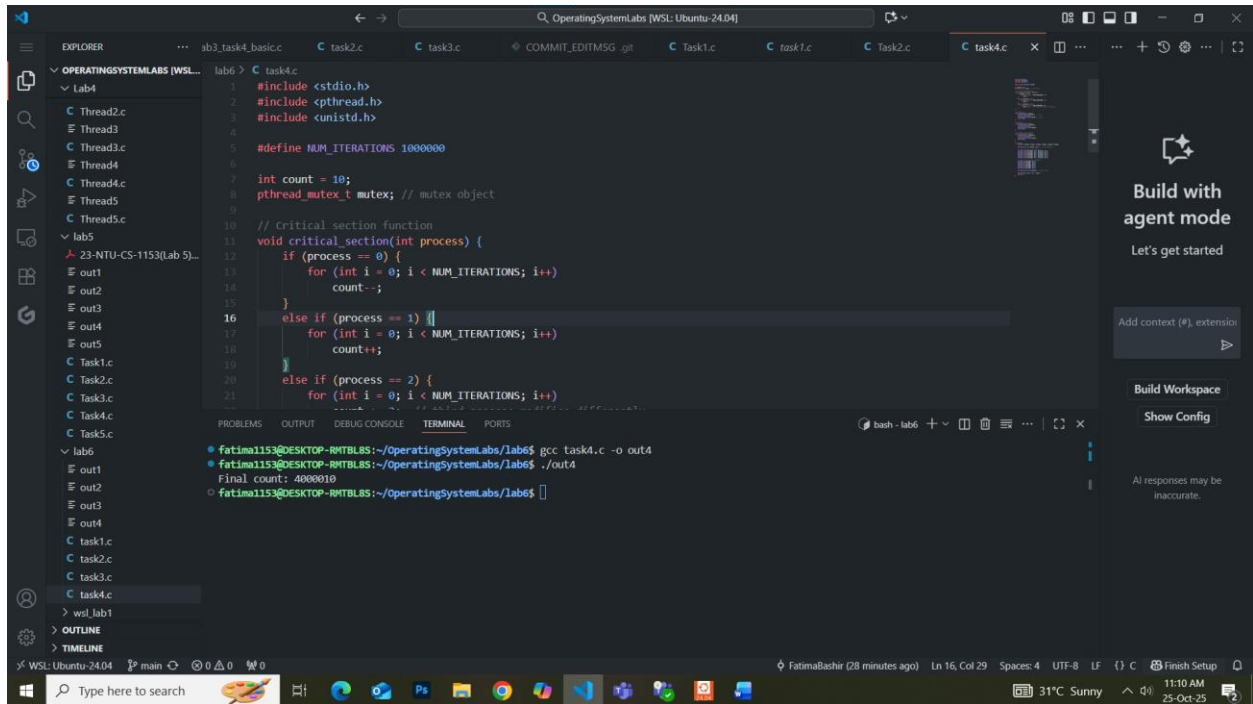
    pthread_mutex_destroy(&mutex); // destroy mutex

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

```

```
return 0;
```

```
}
```



The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays a file tree for 'OPERATINGSYSTEMLABS [WSL: Ubuntu-24.04]'. The main editor window shows the file 'task4.c' with the following C code:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 #define NUM_ITERATIONS 1000000
6
7 int count = 10;
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--;
15     }
16     else if (process == 1) {
17         for (int i = 0; i < NUM_ITERATIONS; i++)
18             count++;
19     }
20     else if (process == 2) {
21         for (int i = 0; i < NUM_ITERATIONS; i++)
```

The TERMINAL panel at the bottom shows the execution of the program:

```
fatima1153@DESKTOP-RMTBLBS:~/OperatingSystemLabs/lab6$ gcc task4.c -o out4
fatima1153@DESKTOP-RMTBLBS:~/OperatingSystemLabs/lab6$ ./out4
Final count: 4000010
fatima1153@DESKTOP-RMTBLBS:~/OperatingSystemLabs/lab6$
```

The status bar at the bottom indicates the file is 'Ln 16, Col 29' and the workspace is 'FatimaBashir (28 minutes ago)'.