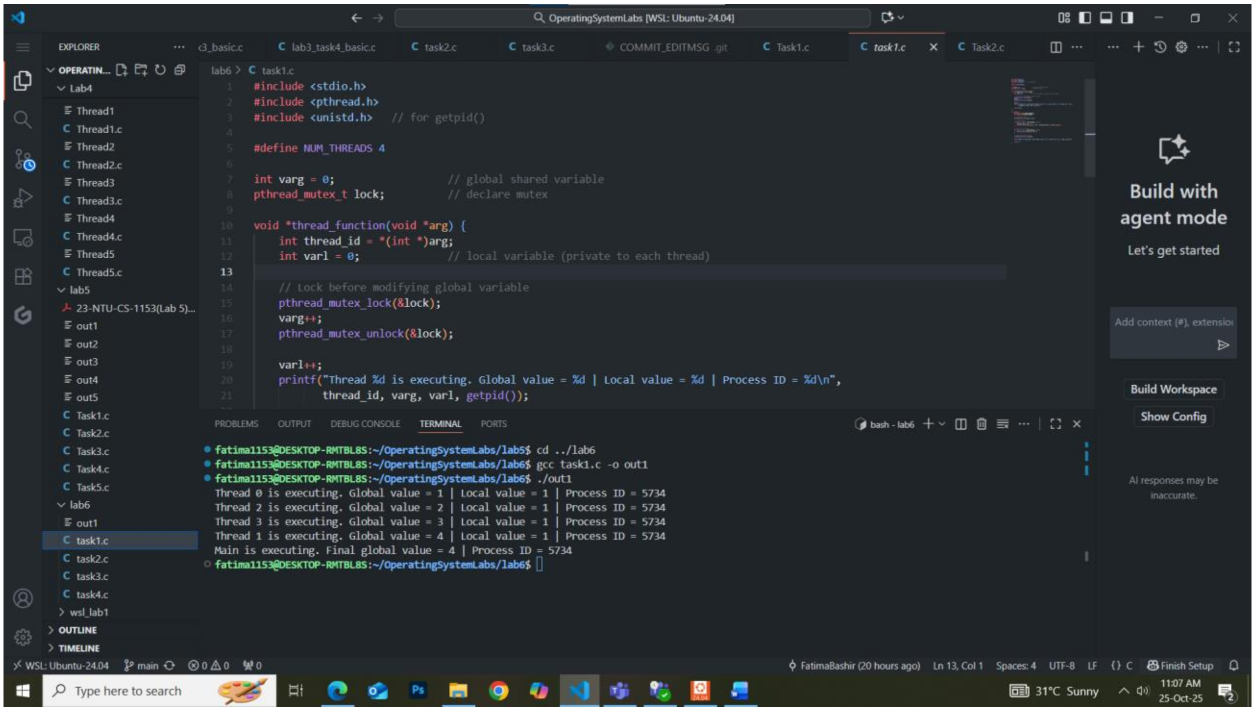
Task 1:

```
#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;
}
```



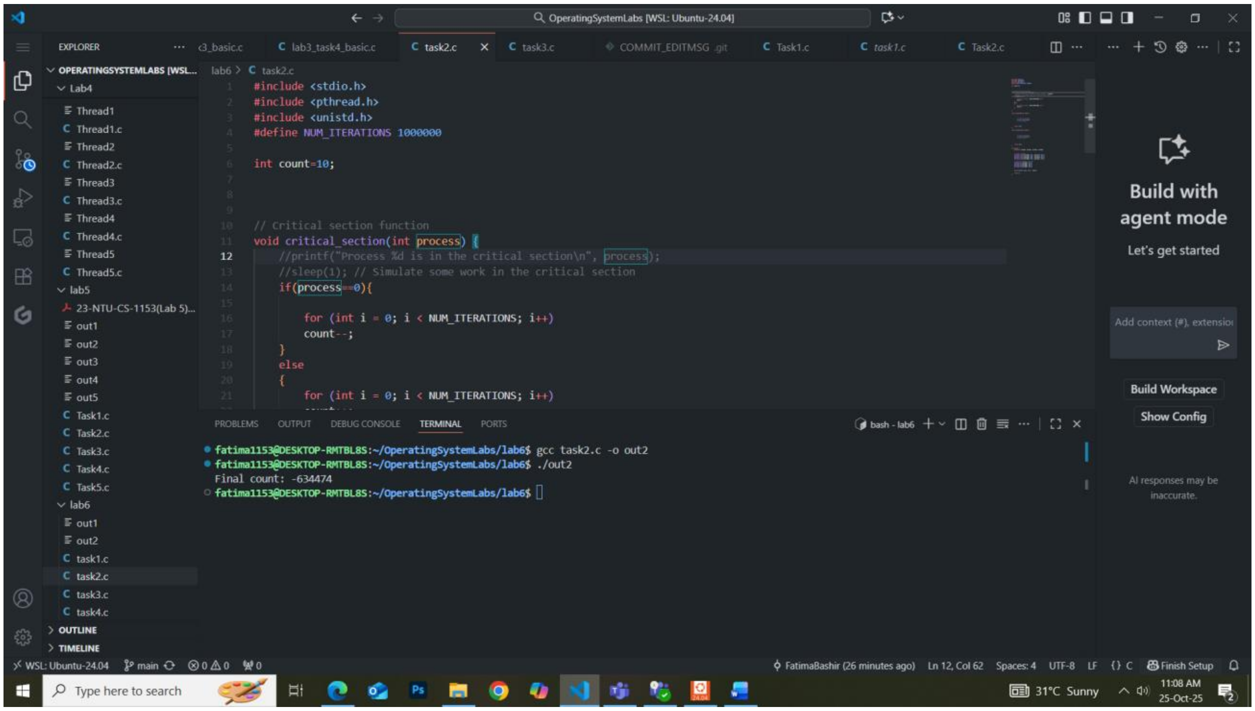
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>   // for getpid()

#define NUM_THREADS 4

int varg = 0;                  // global shared variable
pthread_mutex_t lock;          // declare mutex

void *thread_function(void *arg) {
    int thread_id = *(int *)arg;
    int varl = 0;              // local variable (private to each thread)

    // Lock before modifying global variable
    pthread_mutex_lock(&lock);
    varg++;
    pthread_mutex_unlock(&lock);

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

```
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab5$ cd ../lab6
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ gcc task1.c -o out1
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ ./out1
Thread 0 is executing. Global value = 1 | Local value = 1 | Process ID = 5734
Thread 2 is executing. Global value = 2 | Local value = 1 | Process ID = 5734
Thread 3 is executing. Global value = 3 | Local value = 1 | Process ID = 5734
Thread 1 is executing. Global value = 4 | Local value = 1 | Process ID = 5734
Main is executing. Final global value = 4 | Process ID = 5734
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$
```

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;

}
```

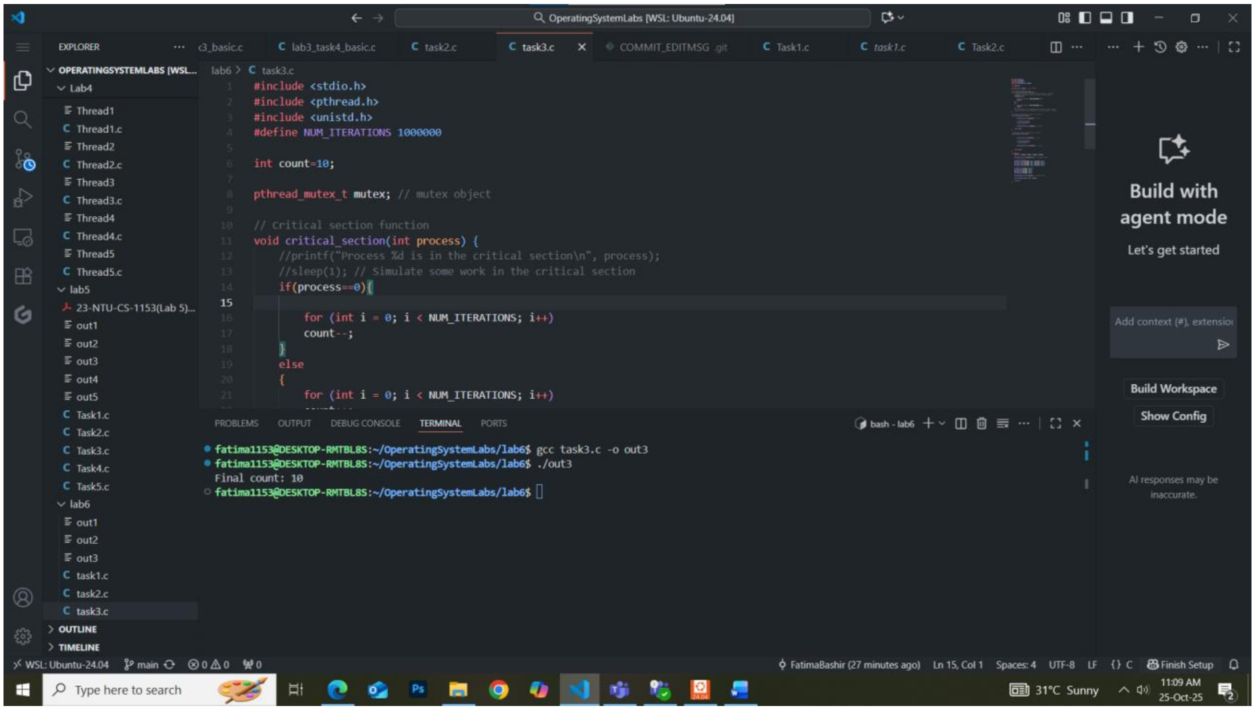


```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

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++)
```

```
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ gcc task2.c -o out2
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ ./out2
Final count: -634474
fatima1153@DESKTOP-RMTBL8S:~/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;
}
```



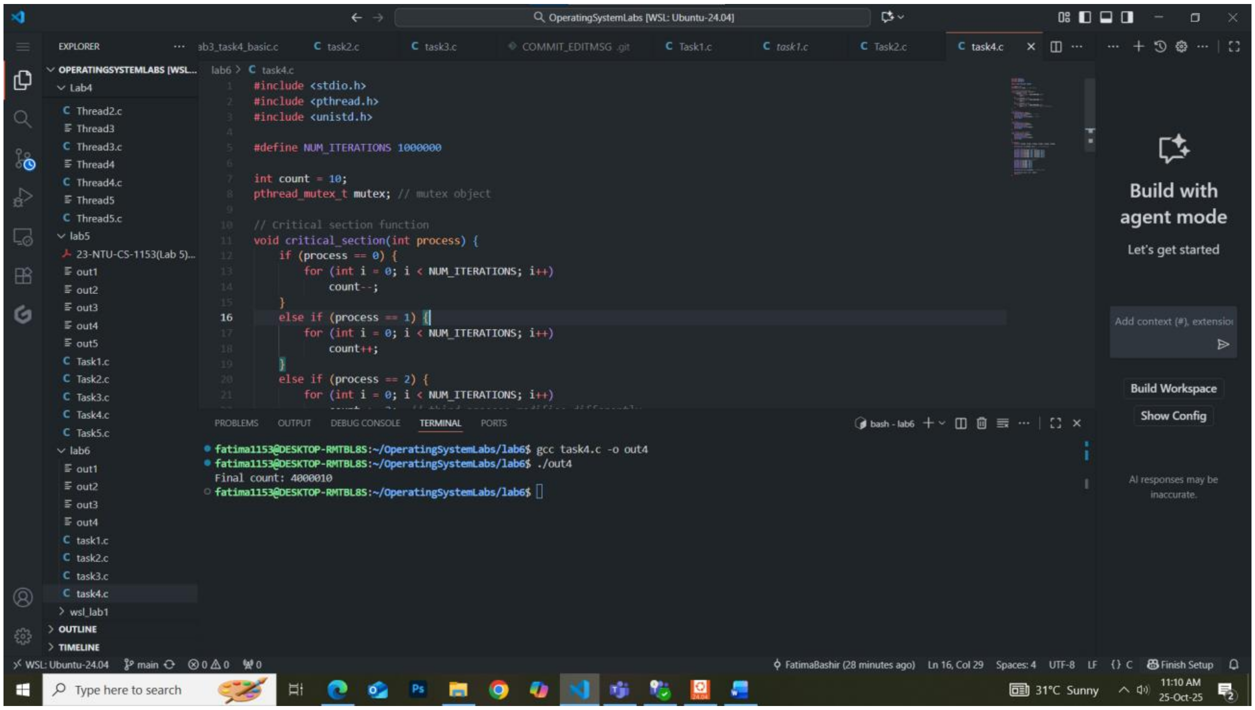
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_ITERATIONS 1000000

int count=10;

pthread_mutex_t mutex; // mutex object

// Critical section function
void critical_section(int process) {
    //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++)
```

```
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ gcc task3.c -o out3
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ ./out3
Final count: 10
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$
```

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;

}
```



```c
#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++)
```

```
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ gcc task4.c -o out4
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$ ./out4
Final count: 4000010
fatima1153@DESKTOP-RMTBL8S:~/OperatingSystemLabs/lab6$
```