National Textile University

Department of Computer Science

Subject

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Kashmir Jamshaid

Registration Number

23-NTU-CS-1167

Lab No.

06

Semester

5th

Task_1

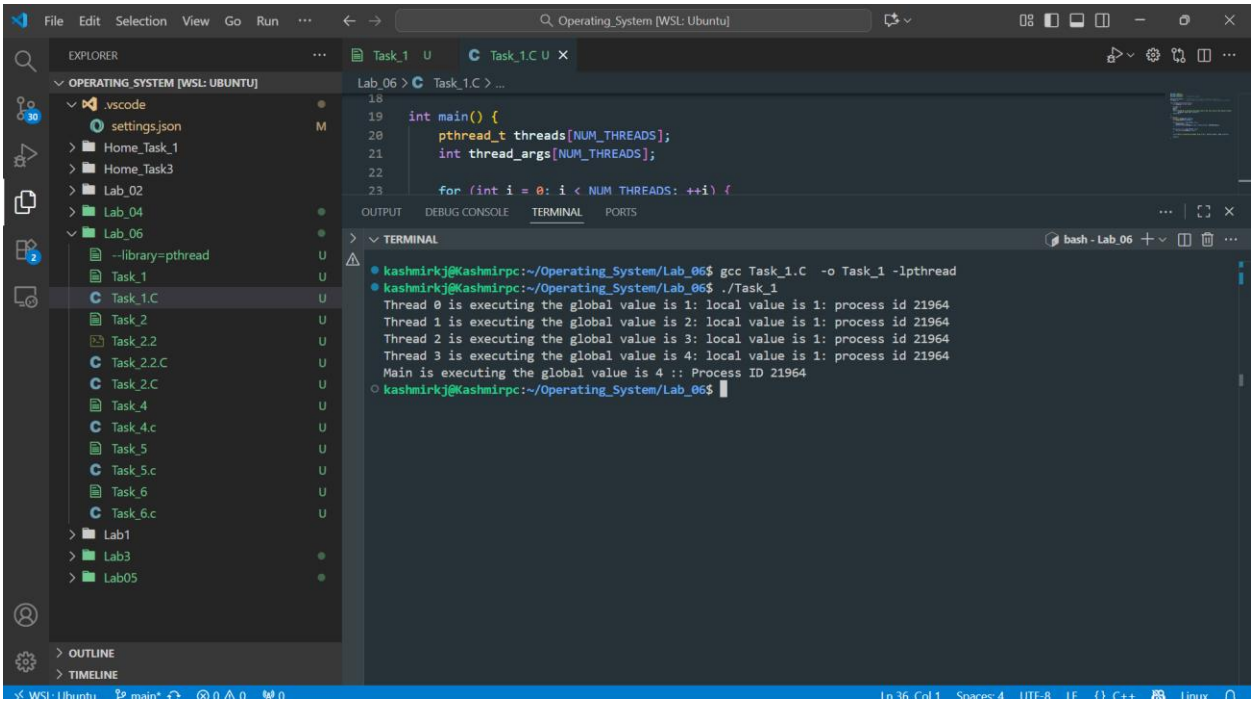
Code:

Input:

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

#define NUM_THREADS 4 // preprocessor macro: number of threads to create (4).
int varg = 0;       // global integer (initialized to 0). All threads share this variable.

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

   int varl = 0;
   varg++;
   varl++;
   printf("Thread %d is executing the global value is %d: local value is %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 the global value is %d :: Process ID %d\n", varg, getpid());

    return 0;
}
```

Output:



Task_2:

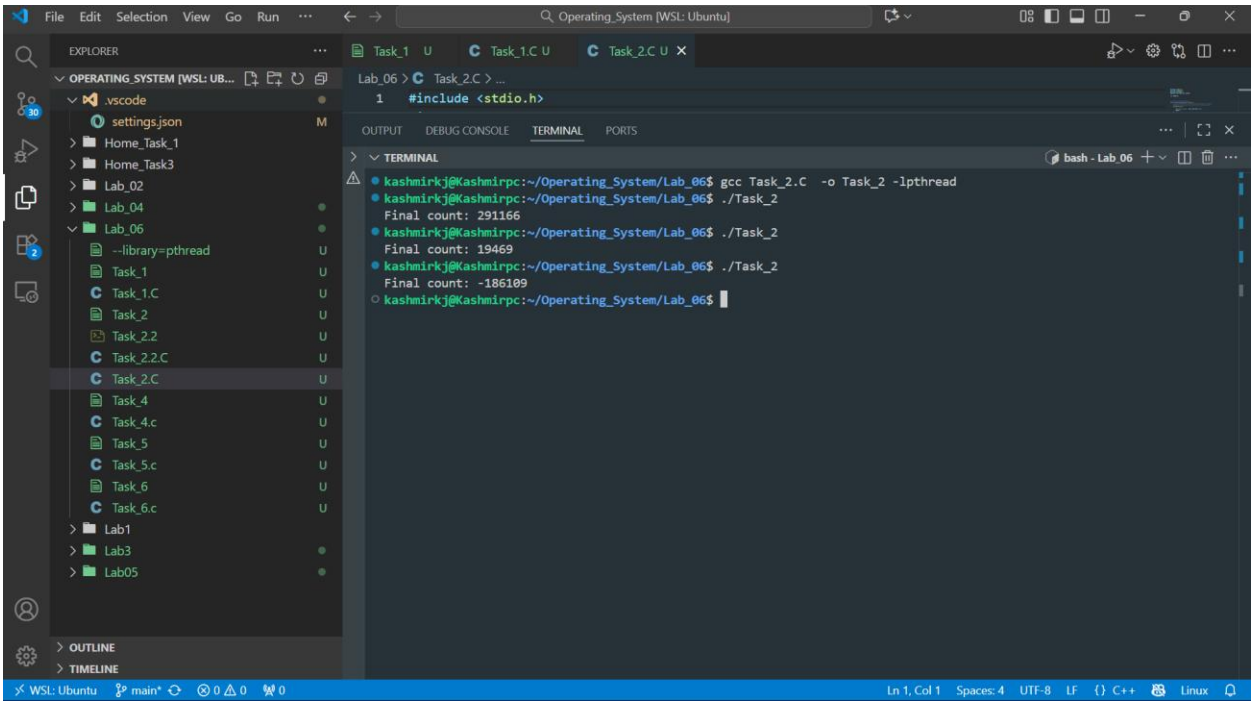
Code:

```
#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;
}
```

Output:



Task_3 (Petrson Problem)

Input:

Code:

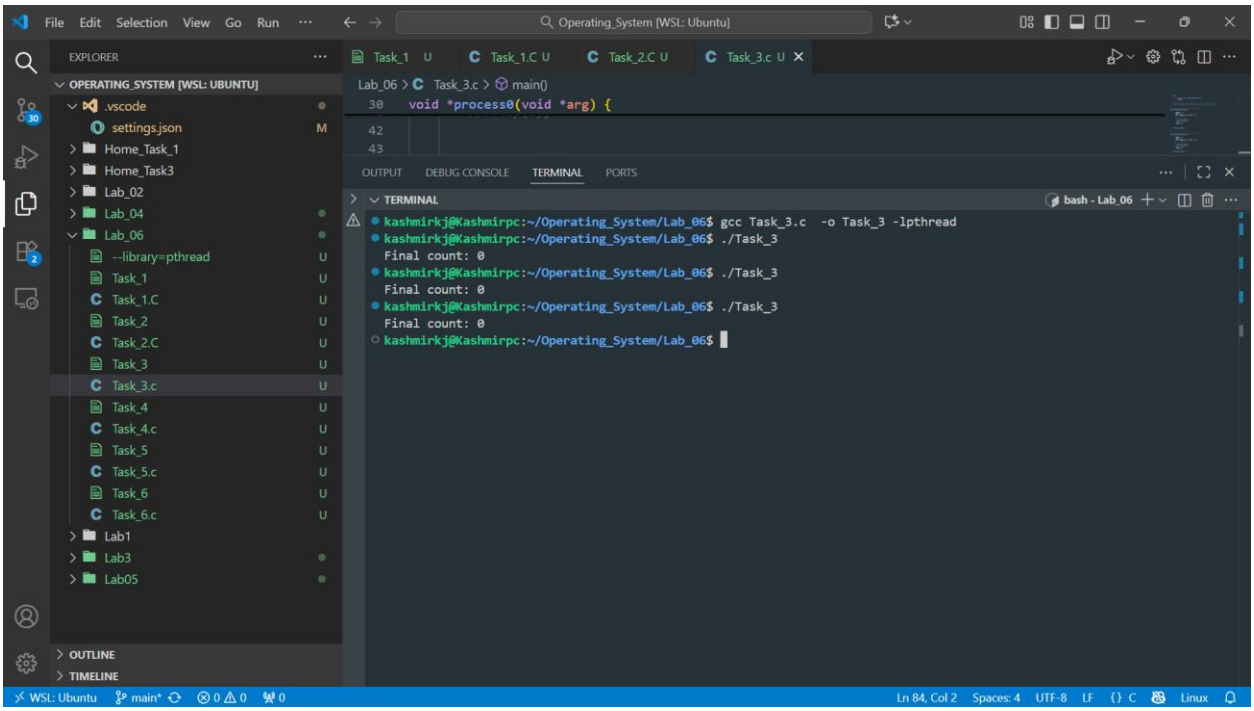
```
#include <stdio.h>

#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;
}
```

Output:



Task_4 (Mutex):

Code:

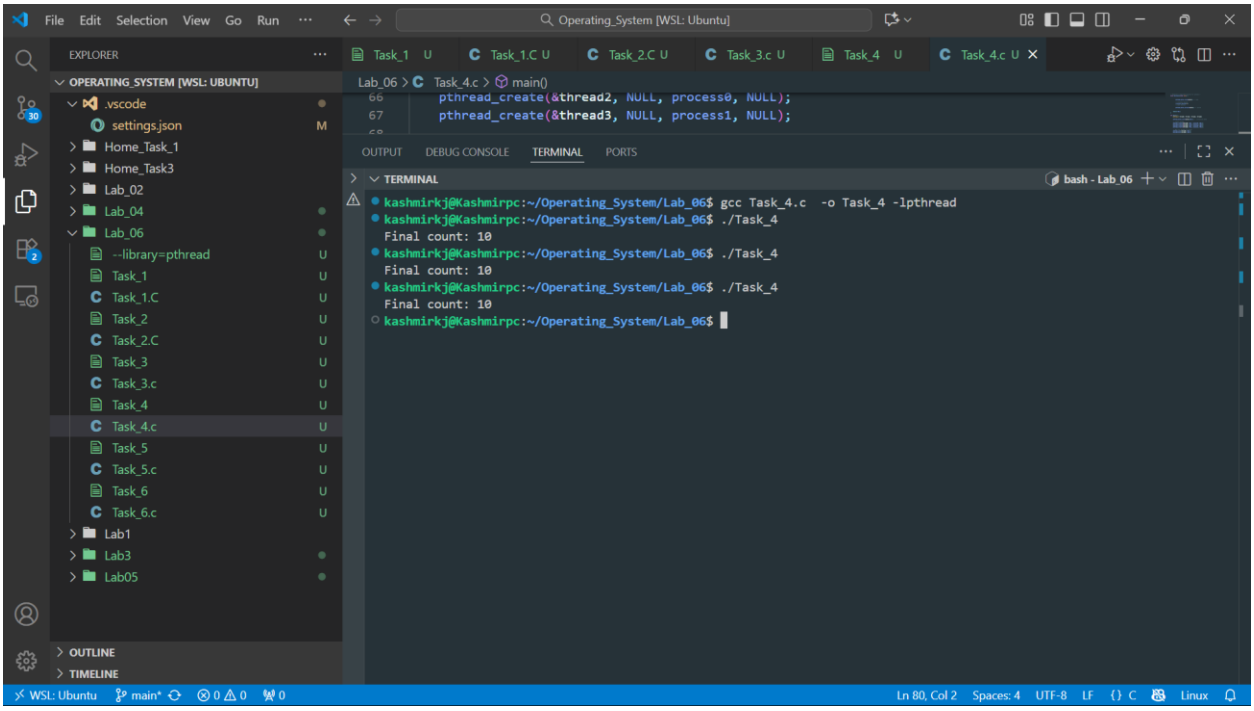
Input:

```
#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++)
    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) {

      pthread_mutex_lock(&mutex); // lock

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

      pthread_mutex_unlock(&mutex); // unlock

   return NULL;
}

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


      pthread_mutex_lock(&mutex); // lock

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

      pthread_mutex_unlock(&mutex); // unlock
```

```
    return NULL;
}

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

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

    // 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);

    pthread_mutex_destroy(&mutex); // destroy mutex

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

   return 0;
}
```

Output:



Task_5 (Mutex with 3 processes)

Code:

Input:

```
#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#define NUM_ITERATIONS 1000000


int count=10;
```

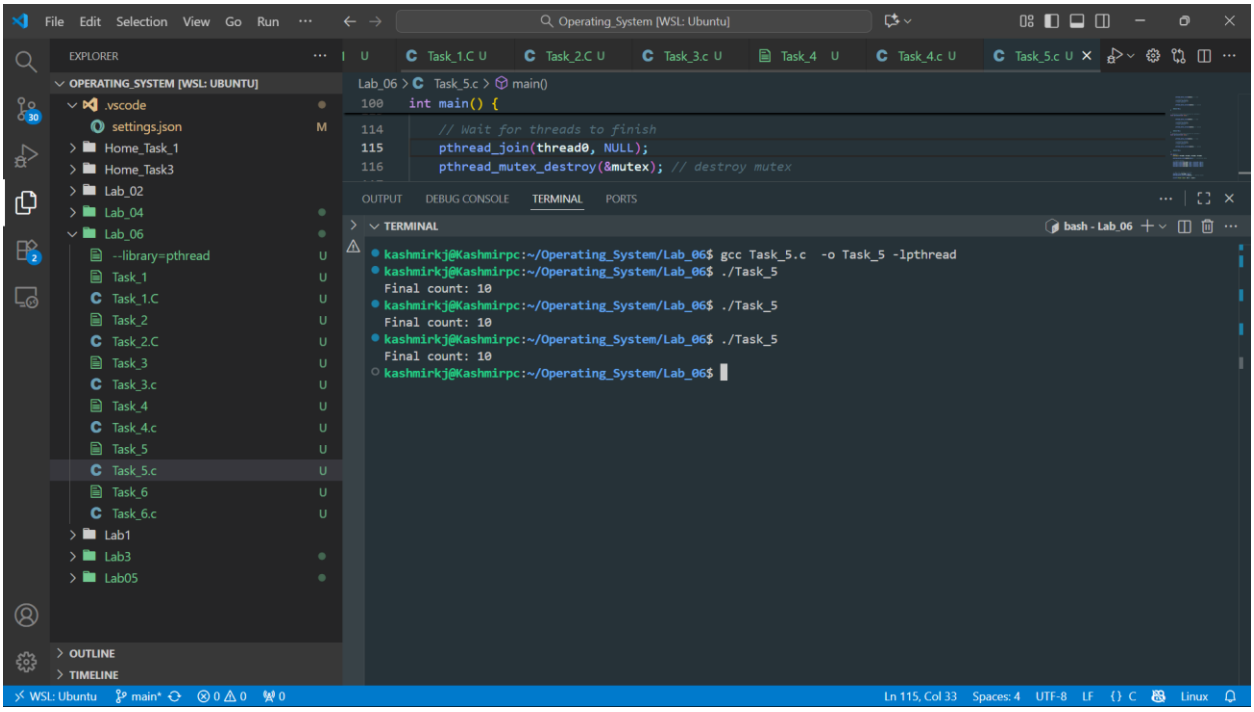
```
pthread_mutex_t mutex; // mutex object

// 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 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--;
  }
  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) {

    pthread_mutex_lock(&mutex); // lock

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

    pthread_mutex_unlock(&mutex); // unlock

  return NULL;
}

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


    pthread_mutex_lock(&mutex); // lock

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

    pthread_mutex_unlock(&mutex); // unlock


  return NULL;
```

```
}

// Peterson's Algorithm function for process 1
void *process2(void *arg) {


    pthread_mutex_lock(&mutex); // lock

    // Critical section
    critical_section(2);
    // Exit section

    pthread_mutex_unlock(&mutex); // unlock


  return NULL;
}
// Peterson's Algorithm function for process 1
void *process3(void *arg) {


    pthread_mutex_lock(&mutex); // lock

    // Critical section
    critical_section(3);
    // Exit section

    pthread_mutex_unlock(&mutex); // unlock
```

```
    return NULL;
}

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

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

    // Create threads
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process2, NULL);
    pthread_create(&thread3, NULL, process3, NULL);



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

    pthread_mutex_destroy(&mutex); // destroy mutex

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

    return 0;
}
```

Output:



Task_6 (// Lock && Unlock)

Code:

Input:

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

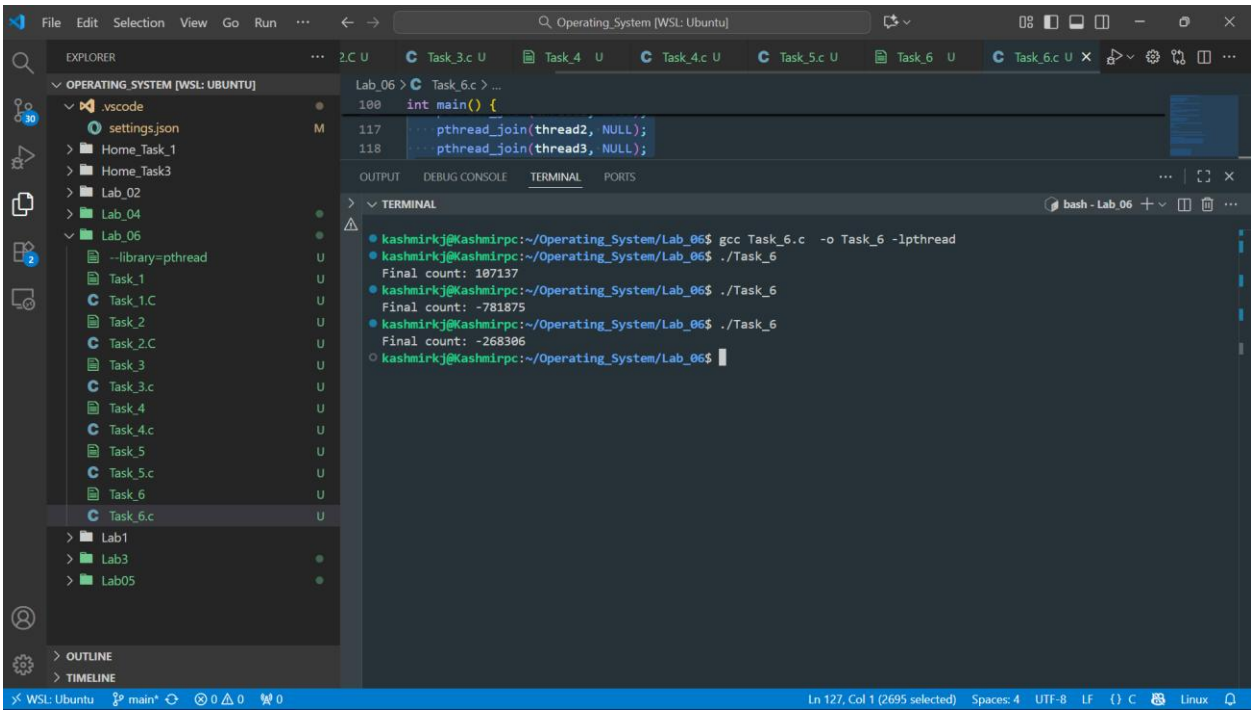
```
    //printf("Process %d has updated count to %d\n", process, count);
    //printf("Process %d is leaving the critical section\n", process);
}

// Peterson's Algorithm function for process 0
void *process0(void *arg) {

        pthread_mutex_lock(&mutex); // lock

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

        pthread_mutex_unlock(&mutex); // unlock

    return NULL;
}

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


        pthread_mutex_lock(&mutex); // lock

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

        pthread_mutex_unlock(&mutex); // unlock
```

```
    return NULL;
}

// Peterson's Algorithm function for process 1
void *process2(void *arg) {


        pthread_mutex_lock(&mutex); // lock

        // Critical section
        critical_section(2);
        // Exit section

        pthread_mutex_unlock(&mutex); // unlock


    return NULL;
}
// Peterson's Algorithm function for process 1
void *process3(void *arg) {


    //    pthread_mutex_lock(&mutex); // lock

        // Critical section
        critical_section(3);
        // Exit section
```

```
    //  pthread_mutex_unlock(&mutex); // unlock


    return NULL;
}

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

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

    // Create threads
    pthread_create(&thread0, NULL, process0, NULL);
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process2, NULL);
    pthread_create(&thread3, NULL, process3, NULL);



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

    pthread_mutex_destroy(&mutex); // destroy mutex

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

```
    return 0;
}
```

Output:



Peterson's Algorithm:	Mutex Based Code
➢ Uses two shared variables: flag[] (interest) and turn (whose turn it is) to enforce mutual exclusion.	➢ Uses pthread_mutex_lock() and pthread_mutex_unlock() — system calls that block or wake threads safely.
➢ Only supports 2 processes (threads). Hard to extend to 3+ safely.	➢ Supports any number of threads/processes with minimal code change.
➢ Controlled by flag and turn variables manually coded.	➢ Controlled by pthread_mutex_lock() internally in the OS.
➢ Must manually reset flags and turn.	➢ Simply call pthread_mutex_unlock().

➢ Works only with strict memory ordering may fail on modern multi-core systems.	➢ Uses kernel-level atomic operations and hardware-supported locking.