



**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);

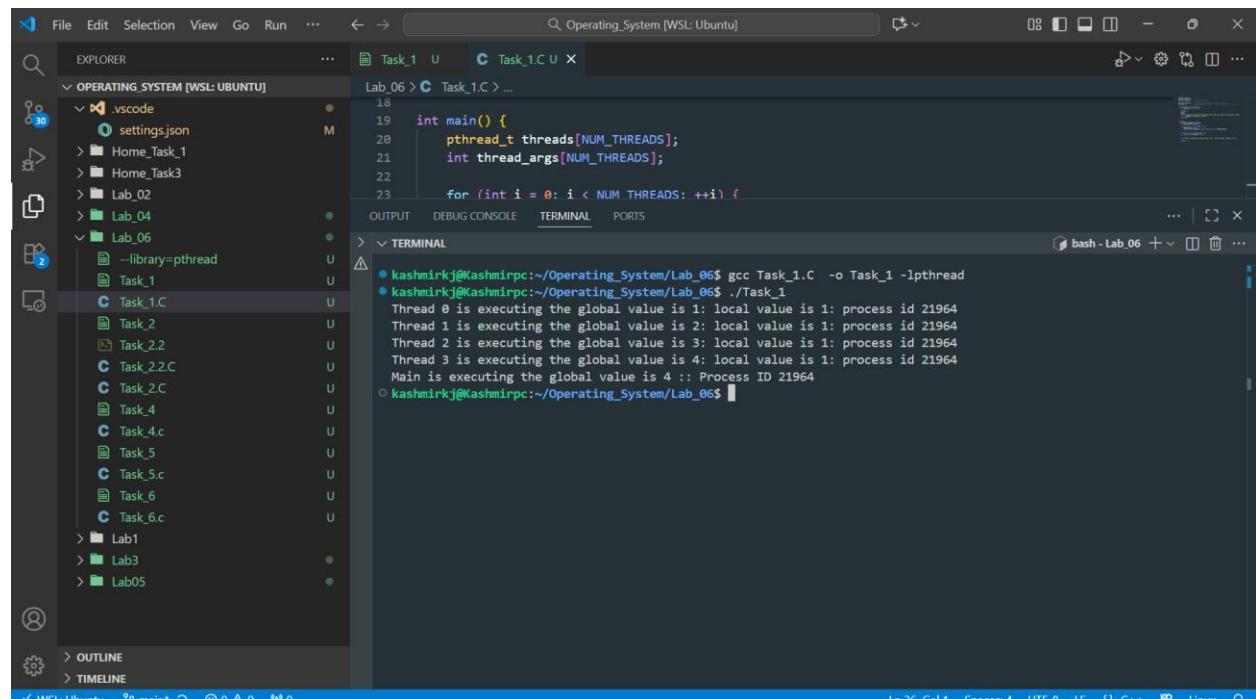
}

printf("Main is executing the global value is %d :: Process ID %d\n", varg, getpid());

return 0;
}

```

## Output:



```

Operating_System [WSL: UBUNTU]
File Edit Selection View Go Run ...
Task_1 Task_1.C
Task_1.C
Lab_06 > Task_1.C > ...
1.8
19 int main() {
20     pthread_t threads[NUM_THREADS];
21     int thread_args[NUM_THREADS];
22
23     for (int i = 0; i < NUM_THREADS; ++i) {
24
25         thread_args[i] = i;
26
27         pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
28
29     }
30
31     for (int i = 0; i < NUM_THREADS; ++i) {
32
33         pthread_join(threads[i], NULL);
34
35     }
36
37     printf("Main is executing the global value is %d :: Process ID %d\n", varg, getpid());
38
39     return 0;
40 }

bash - Lab_06
kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ gcc Task_1.C -o Task_1 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ ./Task_1
Thread 0 is executing the global value is 1: local value is 1: process id 21964
Thread 1 is executing the global value is 2: local value is 1: process id 21964
Thread 2 is executing the global value is 3: local value is 1: process id 21964
Thread 3 is executing the global value is 4: local value is 1: process id 21964
Main is executing the global value is 4 :: Process ID 21964
kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ 

```

## Task\_2:

### Code:

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

int count=10;

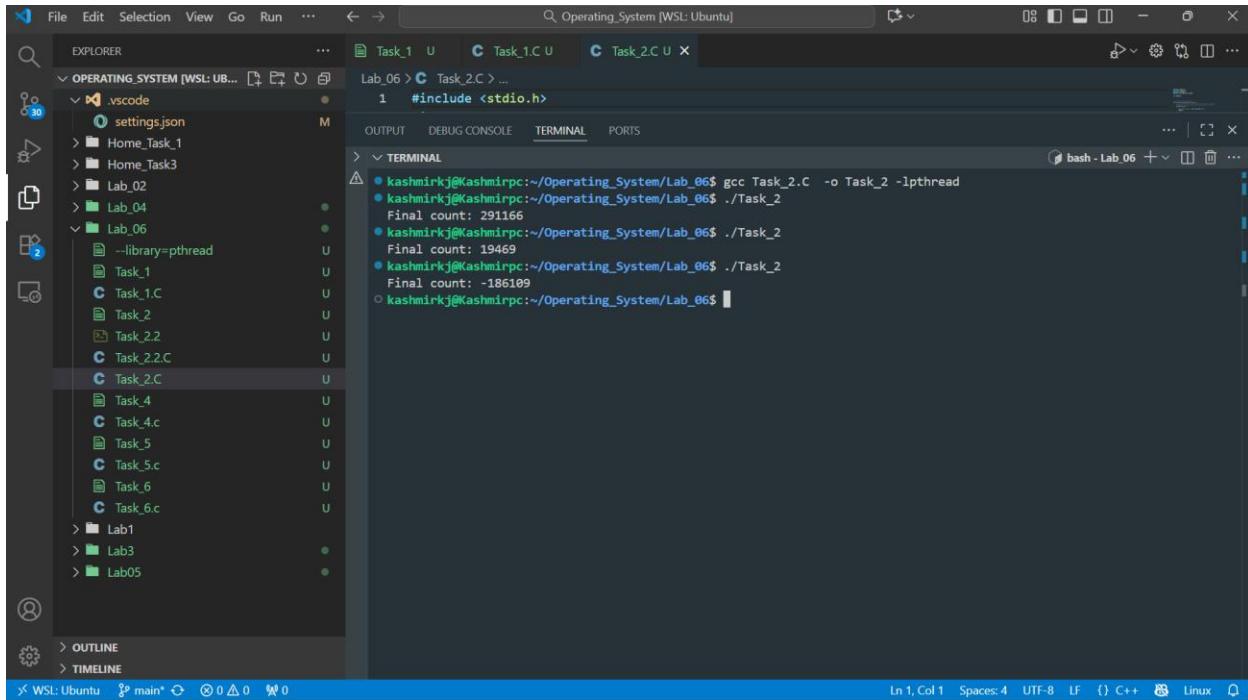
// Critical section function
void critical_section(int process){
    //printf("Process %d is in the critical section\n", process);
    //sleep(1); // Simulate some work in the critical section
    if(process==0){

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

```
void *process0(void *arg){\n\n    // Critical section\n    critical_section(0);\n\n    // Exit section\n\n    return NULL;\n}\n\nvoid *process1(void *arg){\n\n    // Critical section\n    critical_section(1);\n\n    // Exit section\n\n    return NULL;\n}\n\nint main()\n{\n    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 (Peterson 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:

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "OPERATING\_SYSTEM [WSL: UBUNTU]". It includes a ".vscode" folder with "settings.json", and several subfolders like "Home\_Task1", "Lab\_02", "Lab\_04", and "Lab\_06". Inside "Lab\_06", there are files for Task\_1 through Task\_6, and sub-folders for Lab1, Lab3, and Lab05.
- EDITOR:** The main editor area displays the code for "Task\_3.c" with the following content:

```

30 void *process0(void *arg) {
41
42
43

```
- TERMINAL:** The terminal window shows the following session:

```

$ gcc Task_3.c -o Task_3 -lpthread
$ ./Task_3
Final count: 0
$ ./Task_3
Final count: 0
$ ./Task_3
Final count: 0
$ 

```
- STATUS BAR:** At the bottom, it shows "WSL: Ubuntu" and "main.c" as the active file, along with other status indicators like line and column numbers.

## 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:

The screenshot shows the VS Code interface running in a WSL Ubuntu environment. The Explorer sidebar on the left lists files and folders, including 'OPERATING\_SYSTEM [WSL: UBUNTU]', 'vscode', 'settings.json', 'Home\_Task\_1', 'Home\_Task3', 'Lab\_02', 'Lab\_04', and 'Lab\_06'. Inside 'Lab\_06', there are sub-folders like '--library(pthread)' and files such as Task\_1.c, Task\_2.c, Task\_3.c, Task\_4.c, Task\_5.c, Task\_6.c, and Task\_6.c. The terminal tab is active, showing three separate instances of Task\_4.c running. Each instance prints 'Final count: 10' to the console, demonstrating the use of pthreads in a multi-threaded environment.

## 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); // 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:

The screenshot shows a VS Code interface running in WSL (Ubuntu). The Explorer sidebar shows a project folder named 'OPERATING\_SYSTEM [WSL: UBUNTU]' containing files like 'Task\_1.c' through 'Task\_6.c'. The Code Editor shows a C file with code related to threads and mutexes. The Terminal tab is active, displaying three separate runs of the 'Task\_5.c' program. Each run shows the output: 'Final count: 10'. The status bar at the bottom indicates the environment is 'WSL: Ubuntu'.

```
Lab_06 > C Task_5.c > main()
100 int main() {
114     // Wait for threads to finish
115     pthread_join(thread0, NULL);
116     pthread_mutex_destroy(&mutex); // destroy mutex

OUTPUT DEBUG CONSOLE TERMINAL PORTS
> > TERMINAL
@ kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ gcc Task_5.c -o Task_5 -lpthread
@ kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ ./Task_5
Final count: 10
@ kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ ./Task_5
Final count: 10
@ kashmirkj@Kashmirpc:~/Operating_System/Lab_06$ ./Task_5
Final count: 10
@ kashmirkj@Kashmirpc:~/Operating_System/Lab_06$
```

# 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:

The screenshot shows a VS Code interface for a C program named `Task_6.c`. The code includes a function `main` that creates three threads and uses `pthread_join` to wait for them to finish. The terminal window shows the execution of the program, which prints a final count of 107137.

```
Lab_06 > C Task_6.c > ...
100 int main() {
117     ... pthread_join(thread2, NULL);
118     ... pthread_join(thread3, NULL);
```

TERMINAL

```
kashmirkj@KashmirIpc:~/Operating_System/Lab_06$ gcc Task_6.c -o Task_6 -lpthread
kashmirkj@KashmirIpc:~/Operating_System/Lab_06$ ./Task_6
Final count: 107137
kashmirkj@KashmirIpc:~/Operating_System/Lab_06$ ./Task_6
Final count: -781875
kashmirkj@KashmirIpc:~/Operating_System/Lab_06$ ./Task_6
Final count: -268306
kashmirkj@KashmirIpc:~/Operating_System/Lab_06$
```

## Peterson's Algorithm:

## Mutex Based Code

|   |  |
|---|--|
| <ul style="list-style-type: none"><li>➤ Uses two shared variables: flag[] (interest) and turn (whose turn it is) to enforce mutual exclusion.</li></ul> | <ul style="list-style-type: none"><li>➤ Uses pthread_mutex_lock() and pthread_mutex_unlock() — system calls that block or wake threads safely.</li></ul> |
| <ul style="list-style-type: none"><li>➤ Only supports 2 processes (threads). Hard to extend to 3+ safely.</li></ul>                                     | <ul style="list-style-type: none"><li>➤ Supports any number of threads/processes with minimal code change.</li></ul>                                     |
| <ul style="list-style-type: none"><li>➤ Controlled by flag and turn variables manually coded.</li></ul>   | <ul style="list-style-type: none"><li>➤ Controlled by pthread_mutex_lock() internally in the OS.</li></ul>   |
| <ul style="list-style-type: none"><li>➤ Must manually reset flags and turn.</li></ul>   | <ul style="list-style-type: none"><li>➤ Simply call pthread_mutex_unlock().</li></ul>  |

- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>➤ Works only with strict memory ordering<br/>may fail on modern multi-core systems.</li></ul> | <ul style="list-style-type: none"><li>➤ Uses kernel-level atomic operations and<br/>hardware-supported locking.</li></ul> |
|---|---|