

Task1:

Code:

```
#include <stdio.h>

#include <pthread.h>

#include <unistd.h> // ☒ getpid() ke liye include karna zaroori hai


#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", var_g, getpid());
```

```
return 0;
```

}

The image shows a Windows 10 desktop with a Visual Studio Code editor open. The editor is running in a WSL2 environment, specifically Ubuntu 24.04. The Explorer sidebar on the left shows a project named 'LAB06 [WSL: UBUNTU-24.04]'. Inside this project, there are several files and folders: 'a.out', 'task1', 'task1.c', 'task2-out', 'task2.c', 'task3-out', 'task3.c', 'task4', 'task4.c', and 'task5.c'. The 'task5.c' file is currently selected and open in the main editor window. The code in 'task5.c' is a single line: '1 Generate code (Ctrl+I), or select a language (Ctrl+K M). Start typing to dismiss or don't show this again.' The bottom panel of the editor shows the 'TERMINAL' tab. The terminal output shows the following commands and results: 'shanza1209@DESKTOP-LI58TTJ:~/Lab06\$ gcc Task1.c -o task1 -lpthread', 'shanza1209@DESKTOP-LI58TTJ:~/Lab06\$./task1', 'Thread 0 is executing. Global value = 1 | Local value = 1 | Process ID = 26627', 'Thread 2 is executing. Global value = 3 | Local value = 1 | Process ID = 26627', 'Thread 3 is executing. Global value = 4 | Local value = 1 | Process ID = 26627', 'Thread 1 is executing. Global value = 2 | Local value = 1 | Process ID = 26627', 'Main is executing. Final global value = 4 | Process ID = 26627', and 'shanza1209@DESKTOP-LI58TTJ:~/Lab06\$'. The status bar at the bottom of the editor indicates 'WSL: Ubuntu-24.04' and 'main'. The Windows taskbar at the very bottom shows various application icons, including the Start button, File Explorer, Microsoft Edge, and several other background applications.

Task2:

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;
```

```
}
```

```
File Edit Selection View Go Run Terminal Help
Lab06 [WSL: Ubuntu-24.04]

EXPLORER
LAB06 [WSL: UBUNTU-24.04]
  a.out
  task1
  Task1.c
  task2
  task2.out
  Task2.c
  Task3.out
  Task3.c
  task4
  Task4.c
  Task5.c

C task2.c
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 100000
5 // Shared variables
6 int turn;
7 int flag[2];
8 int count=0;
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         for (int i = 0; i < NUM_ITERATIONS; i++)
16             count--;
17     }
18     else
19     {
20         for (int i = 0; i < NUM_ITERATIONS; i++)
21             count++;
22     }
23 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
shanza1209@DESKTOP-LIS8TT3:~/Lab06$ gcc task2.c -o task2 -lpthread
shanza1209@DESKTOP-LIS8TT3:~/Lab06$ ./task2
Final count: 0
shanza1209@DESKTOP-LIS8TT3:~/Lab06$
```

Task3:

Code:

```
#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);
```

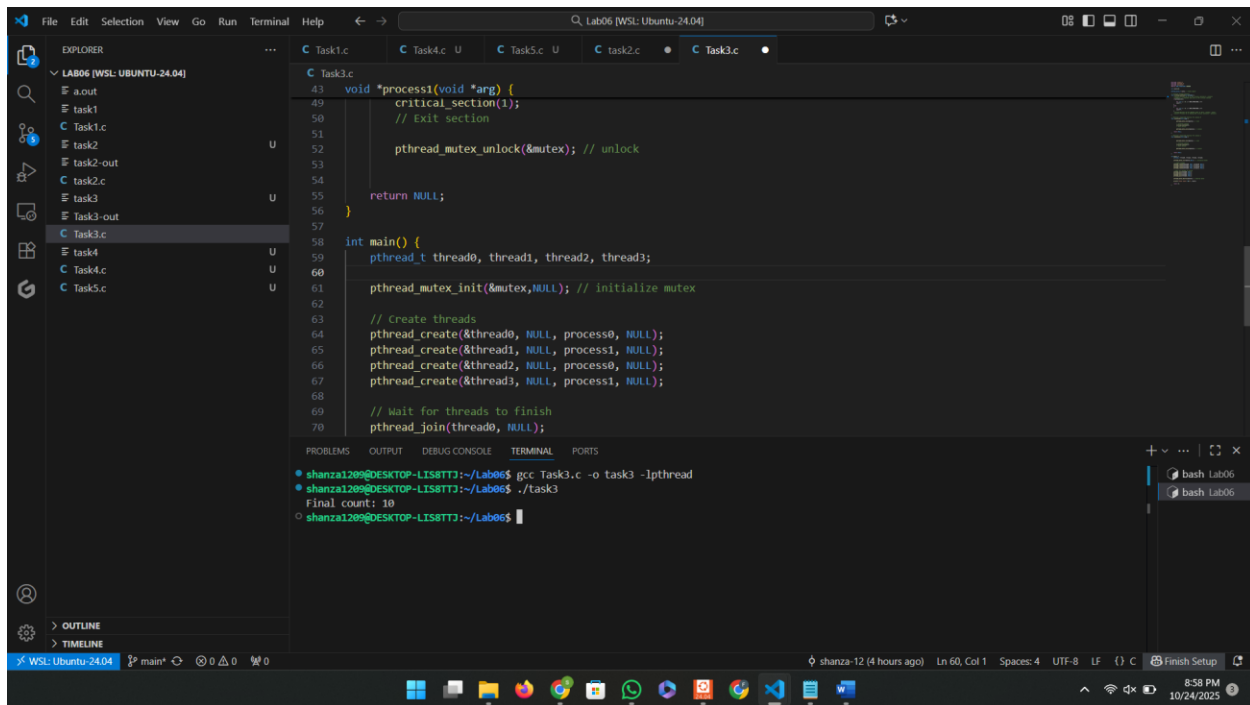
```
pthread_join(thread3, NULL);
```

```
pthread_mutex_destroy(&mutex); // destroy mutex
```

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

```
return 0;
```

```
}
```



Task4:

Code:

```
#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) {

```

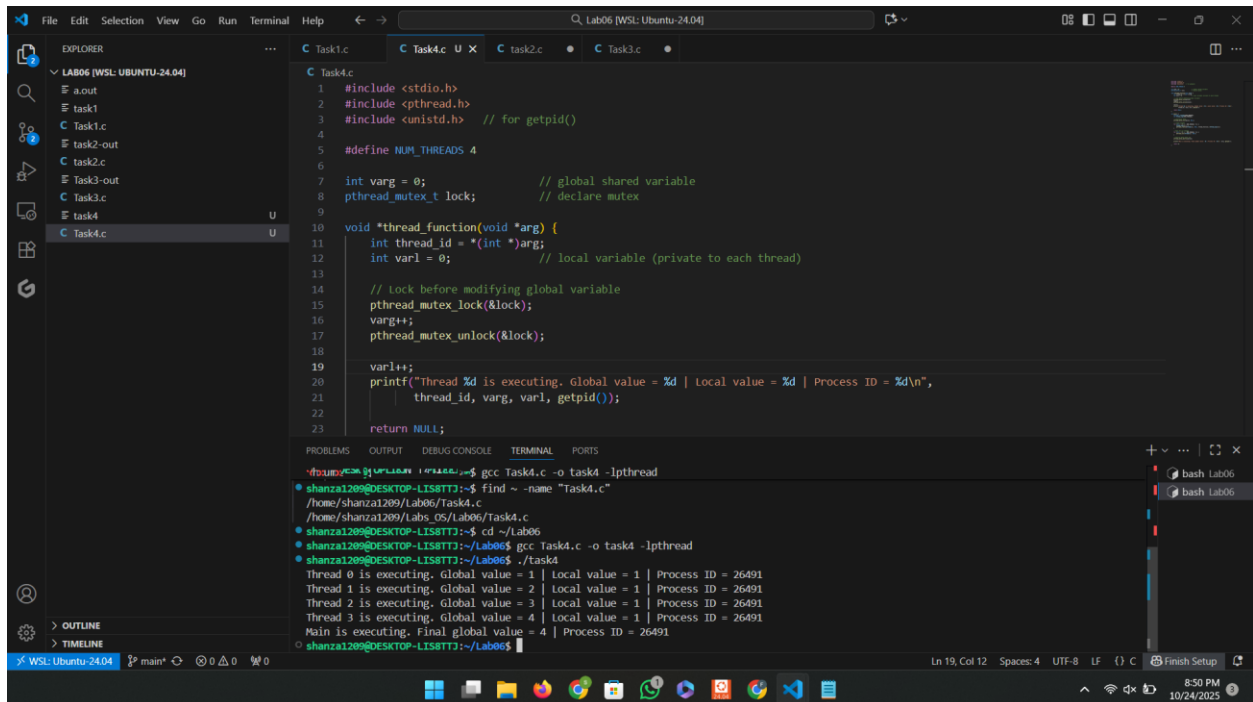
```
pthread_mutex_lock(&mutex);  
critical_section(2);  
pthread_mutex_unlock(&mutex);  
return NULL;  
}
```

```
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;
```

```
}
```



```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h> // for getpid()
4
5 #define NUM_THREADS 4
6
7 int varg = 0; // global shared variable
8 pthread_mutex_t lock; // declare mutex
9
10 void *thread_function(void *arg) {
11     int thread_id = *(int *)arg;
12     int varl = 0; // local variable (private to each thread)
13
14     // lock before modifying global variable
15     pthread_mutex_lock(&lock);
16     varg++;
17     pthread_mutex_unlock(&lock);
18
19     varl++;
20     printf("Thread %d is executing. Global value = %d | Local value = %d | Process ID = %d\n",
21           thread_id, varg, varl, getpid());
22
23     return NULL;
24 }
```

```
shanza1209@DESKTOP-LIS8TT3:~$ gcc Task4.c -o task4 -lpthread
shanza1209@DESKTOP-LIS8TT3:~$ find ~ -name "Task4.c"
/home/shanza1209/Lab06/Lab06/Task4.c
shanza1209@DESKTOP-LIS8TT3:~$ cd ~/Lab06
shanza1209@DESKTOP-LIS8TT3:~/Lab06$ gcc Task4.c -o task4 -lpthread
shanza1209@DESKTOP-LIS8TT3:~/Lab06$ ./task4
Thread 0 is executing. Global value = 1 | Local value = 1 | Process ID = 26491
Thread 1 is executing. Global value = 2 | Local value = 1 | Process ID = 26491
Thread 2 is executing. Global value = 3 | Local value = 1 | Process ID = 26491
Thread 3 is executing. Global value = 4 | Local value = 1 | Process ID = 26491
Main is executing. Final global value = 4 | Process ID = 26491
```

Comparison of mutex and peterson's algorithm:

A mutex is a synchronization mechanism provided by the operating system that allows only one process or thread to access a shared resource at a time. When a process wants to enter its critical section, it locks the mutex, ensuring that no other process can enter until it is unlocked. Once the process finishes its task, it releases the mutex, allowing other processes to proceed. Mutexes are hardware- or OS-supported, making them efficient, reliable, and widely used in real-world systems for handling concurrency and preventing race conditions. A mutex is a hardware or OS-based synchronization tool that ensures only one thread enters the critical section at a time, preventing data inconsistency.

Peterson's Algorithm:

In contrast, Peterson's algorithm is a software-based solution for achieving mutual exclusion between two processes. It uses two shared variables—flag[] and turn—to coordinate which process can enter the critical section. Each process sets its flag to indicate interest in entering and assigns the turn to the other process. A process only enters its critical section when it is its turn and the other process is not interested. Peterson's algorithm is a software-based synchronization

technique that uses shared variables to ensure mutual exclusion between two processes without hardware support.