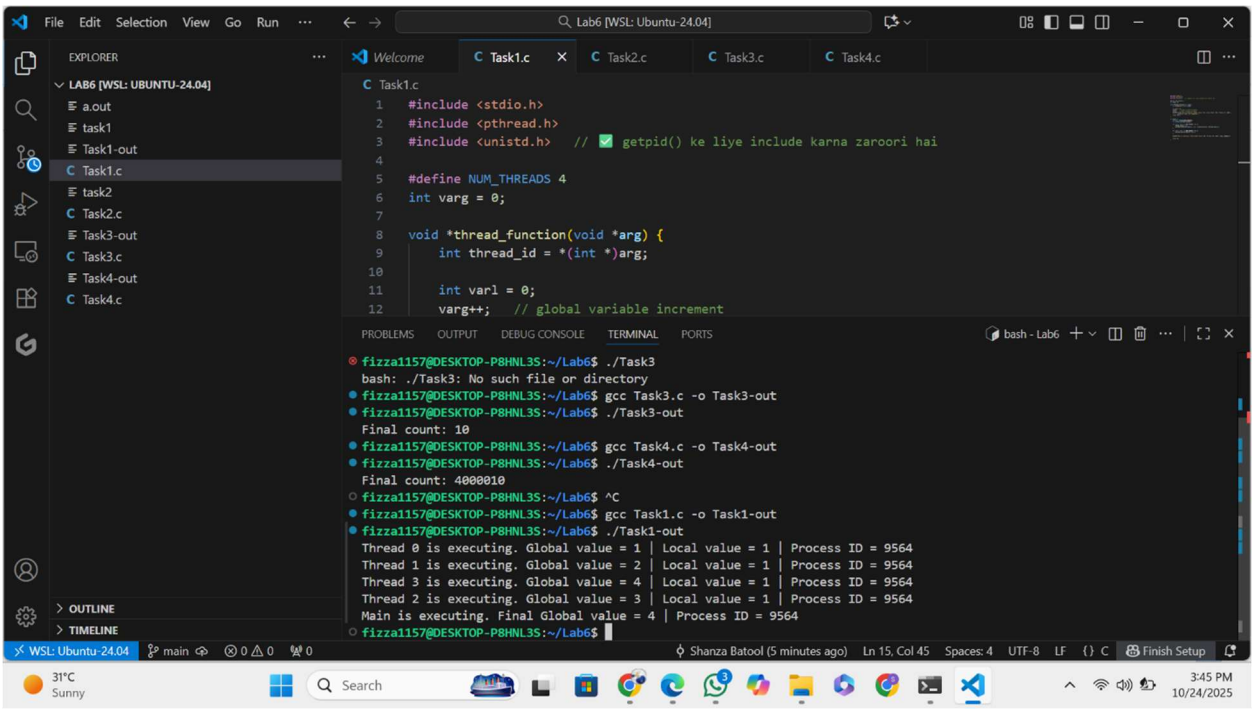
## Task1:

### 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", varg, getpid());

  return 0;
}
```




## Task2:

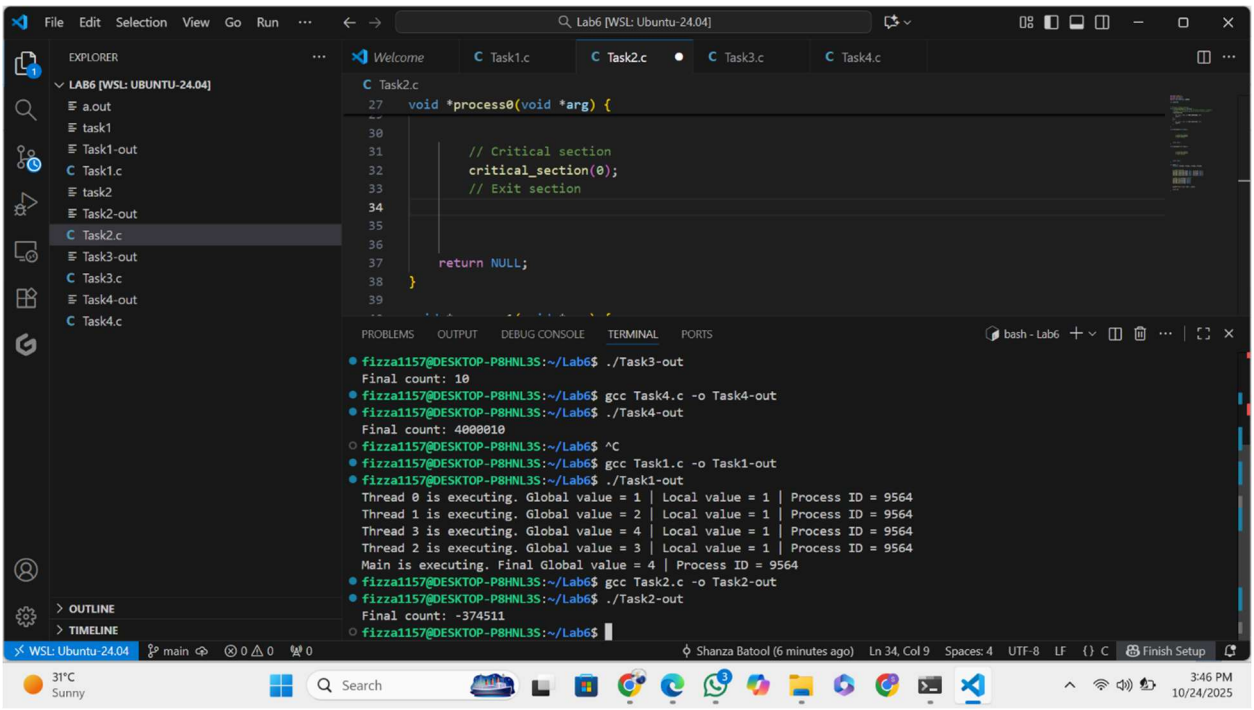
## Code:

```
#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;
}
```



## 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;
}
```

## Task2:

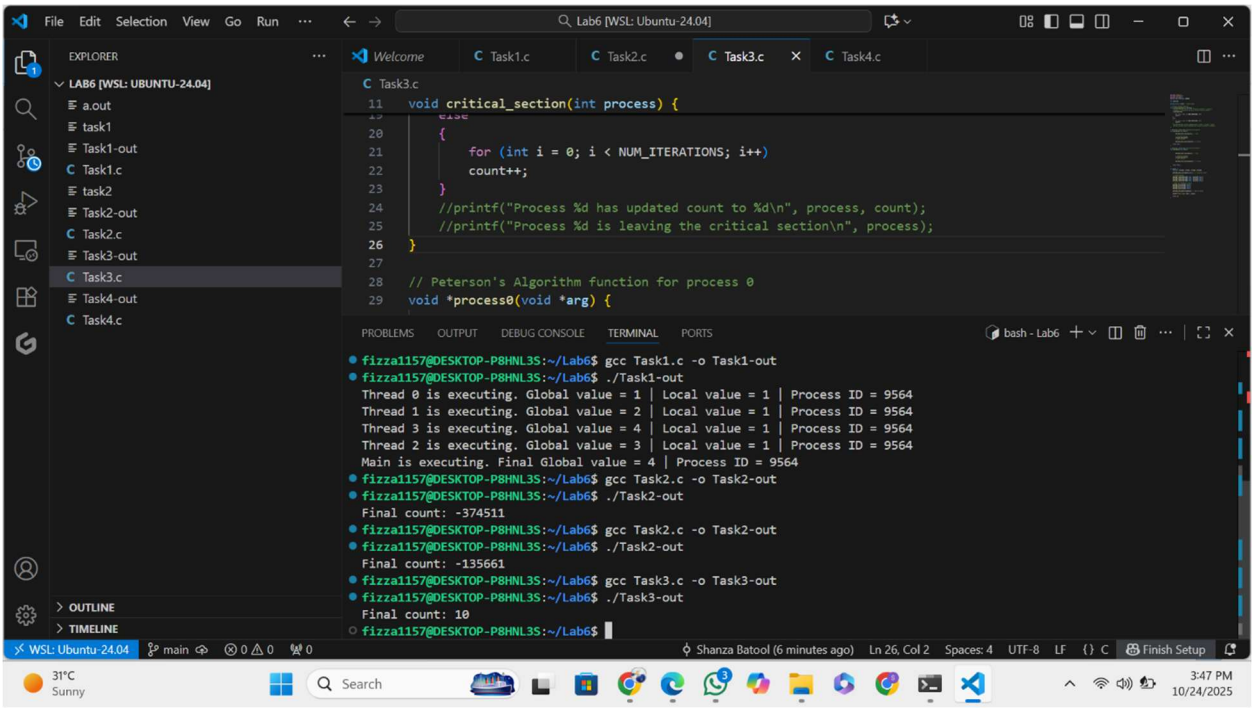
### Code:

```
#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;
}
```




## 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;
}
```