
# National Textile University

*Department of Computer Science*

**Subject:**

Operating system

**Submitted To:**

Sir Nasir

**Submitted By:**

Noor Ul Ain

**Registration No:**

23-NTU-CS-1221

**Lab No:**

9

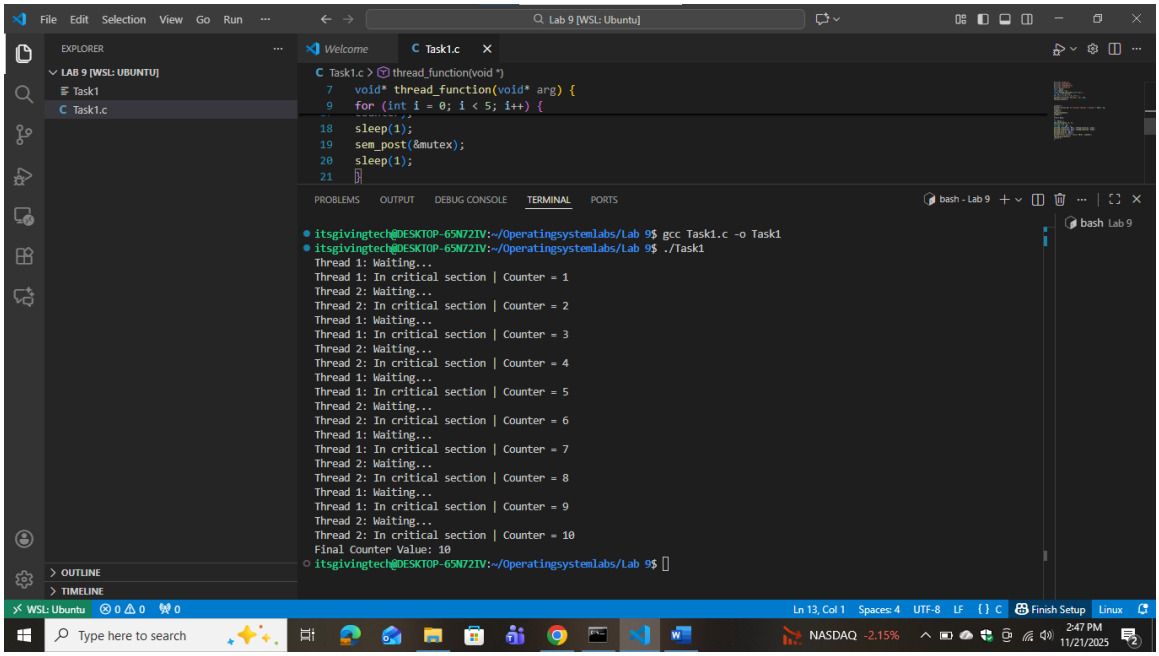
**Semester:**

5

## Task 1:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex;
int counter = 0;
void* thread_function(void* arg) {
int id = *(int*)arg;
for (int i = 0; i < 5; i++) {
printf("Thread %d: Waiting...\n", id);
sem_wait(&mutex);



counter++;
printf("Thread %d: In critical section | Counter = %d\n", id,
counter);
sleep(1);
sem_post(&mutex);
sleep(1);
}
return NULL;
}
int main() {
sem_init(&mutex, 0, 1);
pthread_t t1, t2;
int id1 = 1, id2 = 2;
pthread_create(&t1, NULL, thread_function, &id1);
pthread_create(&t2, NULL, thread_function, &id2);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final Counter Value: %d\n", counter);
sem_destroy(&mutex);
return 0;
}
```

## Output:




## Remarks:

In this code, two threads are trying to increment the shared counter in the same function, here semaphore value is one means one thread can enter the lock at a time, when one thread enters and increment the code the other one waits and vice versa.

- **Different cases:**
  **//sem_post(&mutex):**

Because of this  one thread gets locked in the semaphore forever and other thread gets to wait forever and never prints the final counter value.

**Sem_init(&mutex, 0,0):**

We initialized the semaphore with zero, this means that both threads will be blocked and never gets to enter the &sem_wait(&mutex) and the program freezes here.

**//sem_wait(&mutex)**

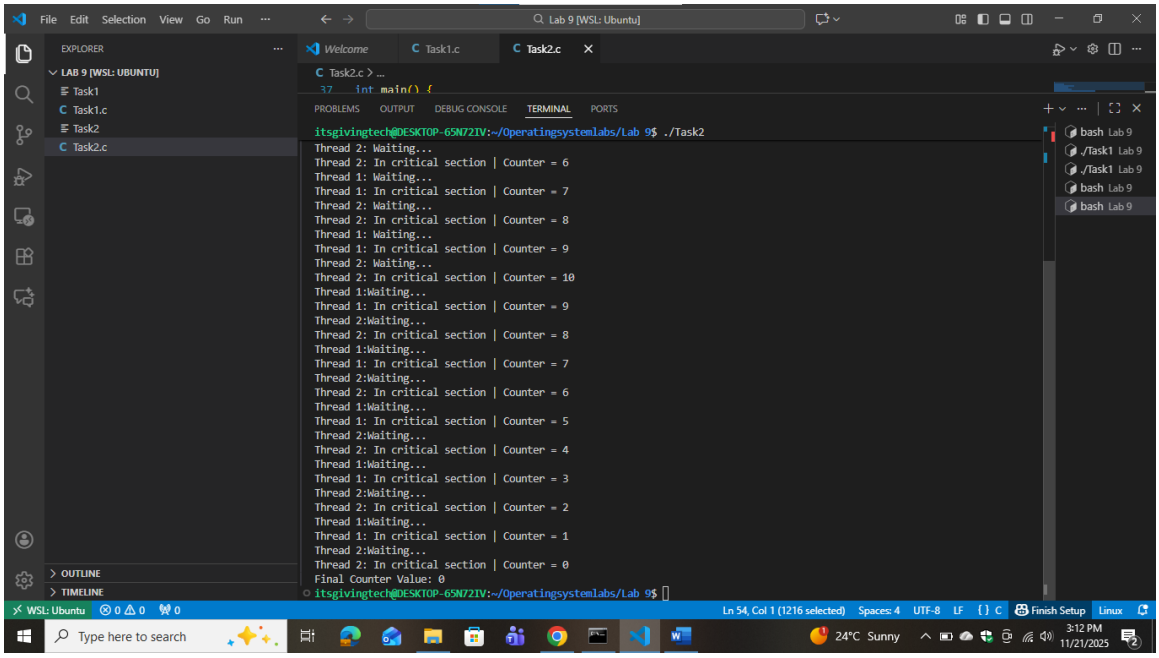
Thread 1 and 2 are blocked  forever here.

## Task2

## Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex;
int counter = 0;
void* thread_function(void* arg) {
int id = *(int*)arg;
for (int i = 0; i < 5; i++) {
printf("Thread %d: Waiting...\n", id);
sem_wait(&mutex);



counter++;
printf("Thread %d: In critical section | Counter = %d\n", id,
counter);
sleep(1);
sem_post(&mutex);
sleep(1);
}
return NULL;
}
void* thread_function2(void* arg){
    int id=*(int*)arg;
    for(int i=0;i<5;i++){
        printf("Thread %d:Waiting...\n",id);
        sem_wait(&mutex);
        counter--;
        printf("Thread %d: In critical section | Counter = %d\n", id,counter);
            sleep(1);
            sem_post(&mutex);
            sleep(1);
    }
            return NULL;
}
int main() {
sem_init(&mutex, 0, 1);
pthread_t t1, t2;
int id1 = 1, id2 = 2;
pthread_create(&t1, NULL, thread_function, &id1);
pthread_create(&t2, NULL, thread_function2, &id2);
pthread_join(t1, NULL);
pthread_join(t2, NULL);

printf("Final Counter Value: %d\n", counter);
sem_destroy(&mutex);
return 0;
}

```

## Output:




## Remarks:

Here I've created two functions for each thread where one function increments the counter and the other one decrements the counter.

- **Cases:**

  **//sem_post(&mutex):**

  Because of this the one thread gets locked in the semaphore forever and other thread gets to wait forever and never prints the final counter value

  **Sem_init(&mutex,0,0):**

  Both threads are blocked to enter the sem_wait(&mutex)

  **//sem_wait(&mutex)**

  Thread 1 and 2 waits forever here

# Task3:

## Difference between mutex lock and binary semaphore:

| Feature | Binary Semaphore (your code) | Mutex Lock (if you used pthread_mutex_t) |
|---|---|---|
| **Type** | `sem_t mutex` initialized to 1 | `pthread_mutex_t mutex` |
| **Ownership** | No ownership for semaphore any thread can call `sem_post()` even if it didn't call `sem_wait()` | Ownership is enforced which means only the thread that locked it can unlock it |
| **Use in above codes** | `sem_wait(&mutex)` locks the counter and `sem_post(&mutex)` unlock the counter | We can use `pthread_mutex_lock(&mutex)` and `pthread_mutex_unlock(&mutex)` unlocks the counter |
| **Can be used for signaling?** | Yes, semaphores can signal other threads/processes | No, mutex is strictly for mutual exclusion |
| **Error checking** | No error if another thread calls `sem_post()` incorrectly | It shows error if a thread that doesn't own the mutex tries to unlock it. |
| **Blocking behavior** | The threads wait if the semaphore value is 0 | Threads wait if the mutex is locked by another thread |
| **Initialization** | `sem_init(&mutex, 0, 1)` | `pthread_mutex_init(&mutex, NULL)` |