National Textile University

Department of Computer Science

Subject:

Operating System

Submitted To:

Sir Nasir Mehmood

Submitted By:

Hafsa Tayyab

Registration No:

23-NTU-CS-1163

Lab No:

9

Semester:

5th

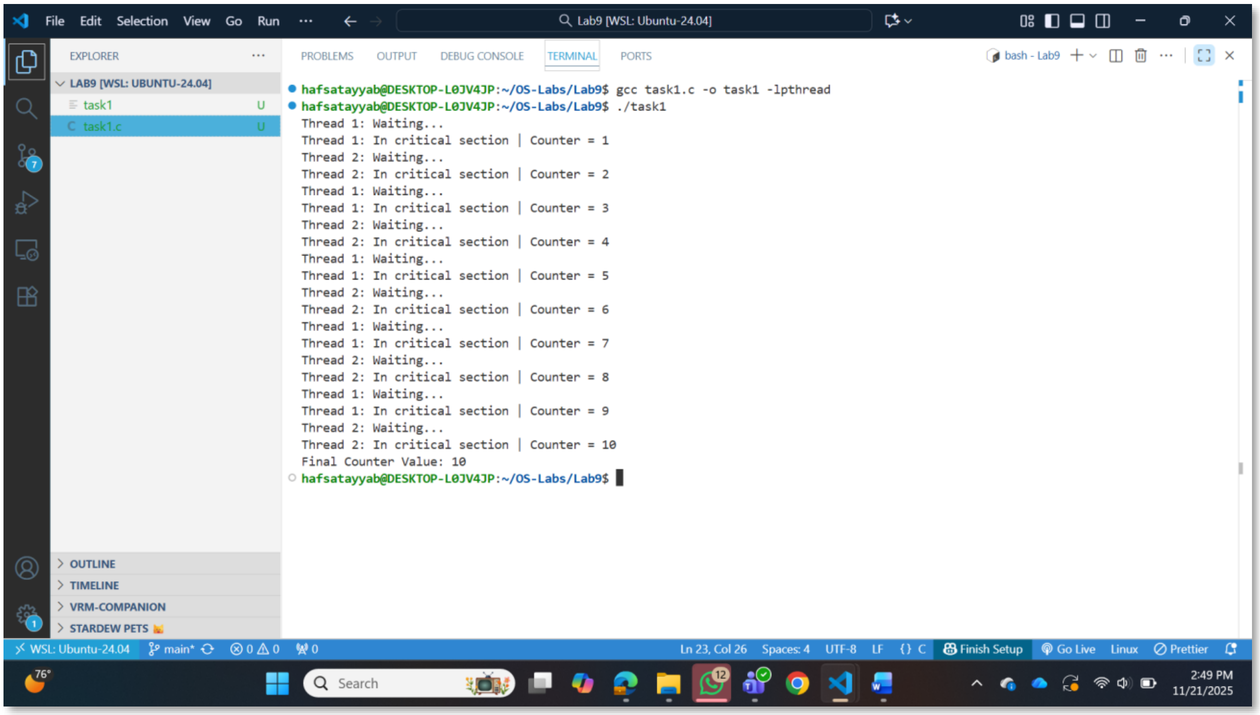
Lab 9: Synchronization

Introduction to Semaphores

Task 1: Binary Semaphore Example

Source Code:

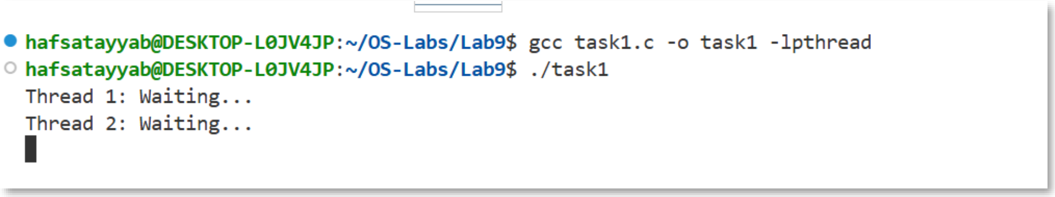
```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem_t mutex; // Binary semaphore

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); // Acquire

        // Critical section

        counter++;

        printf("Thread %d: In critical section | Counter = %d\n", id,

        counter);

        sleep(1);

        sem_post(&mutex); // Release

        sleep(1);

    }

    return NULL;
}

int main() {

    sem_init(&mutex, 0, 1); // Binary semaphore initialized to 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;

}
```



```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task1.c -o task1 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task1
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 2
Thread 1: Waiting...
Thread 1: In critical section | Counter = 3
Thread 2: Waiting...
Thread 2: In critical section | Counter = 4
Thread 1: Waiting...
Thread 1: In critical section | Counter = 5
Thread 2: Waiting...
Thread 2: In critical section | Counter = 6
Thread 1: Waiting...
Thread 1: In critical section | Counter = 7
Thread 2: Waiting...
Thread 2: In critical section | Counter = 8
Thread 1: Waiting...
Thread 1: In critical section | Counter = 9
Thread 2: Waiting...
Thread 2: In critical section | Counter = 10
Final Counter Value: 10
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$
```

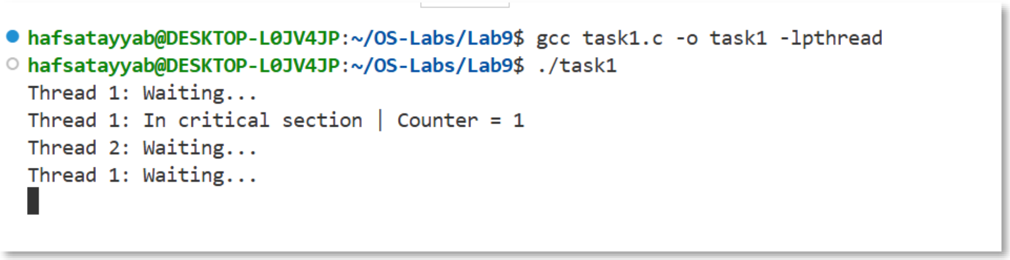
Remarks:

- With;   sem_init(&mutex, 0, 1); // Binary semaphore initialized to 0



```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task1.c -o task1 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task1
Thread 1: Waiting...
Thread 2: Waiting...
█
```

- With; // sem_post(&mutex); // Release



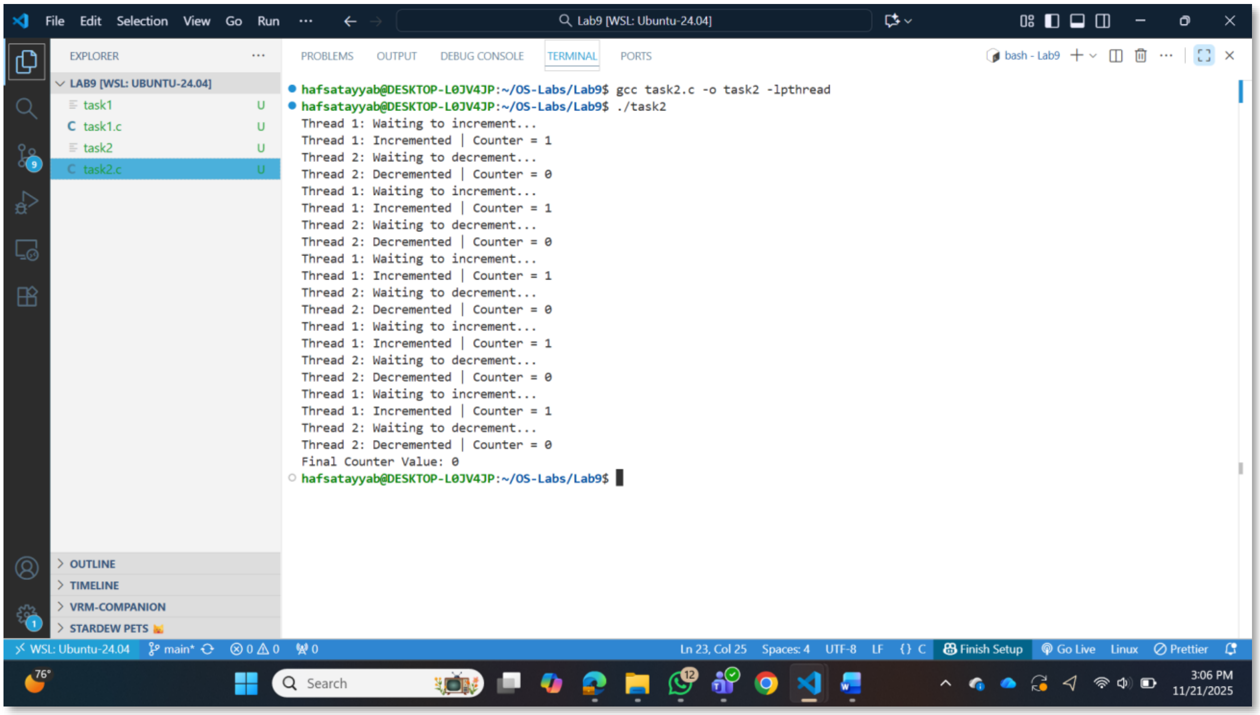
```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task1.c -o task1 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task1
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
█
```

Task2: Binary Semaphore Example

Source Code:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem_t mutex; // Binary semaphore

int counter = 0;

// Thread that increments counter

void* increment_thread(void* arg) {

    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
        printf("Thread %d: Waiting to increment...\n", id);
        sem_wait(&mutex); // acquire
        counter++;
        printf("Thread %d: Incremented | Counter = %d\n", id, counter);
        sleep(1);
        sem_post(&mutex); // release
        sleep(1);
    }
    return NULL;
}
// Thread that decrements counter
void* decrement_thread(void* arg) {
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
        printf("Thread %d: Waiting to decrement...\n", id);
        sem_wait(&mutex); // acquire
        counter--;
        printf("Thread %d: Decremented | Counter = %d\n", id, counter);
        sleep(1);
        sem_post(&mutex); // release
        sleep(1);
    }
    return NULL;
}
```

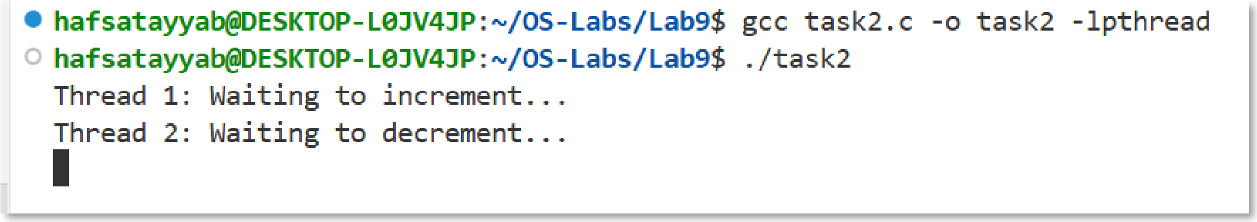
```
int main() {
    sem_init(&mutex, 0, 1);   // semaphore = 1
    pthread_t t1, t2;
    int id1 = 1, id2 = 2;
    pthread_create(&t1, NULL, increment_thread, &id1);
    pthread_create(&t2, NULL, decrement_thread, &id2);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Final Counter Value: %d\n", counter);
    sem_destroy(&mutex);
    return 0;
}
```



```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task2.c -o task2 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task2
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decremented | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decremented | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decremented | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decremented | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decremented | Counter = 0
Final Counter Value: 0
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ 
```

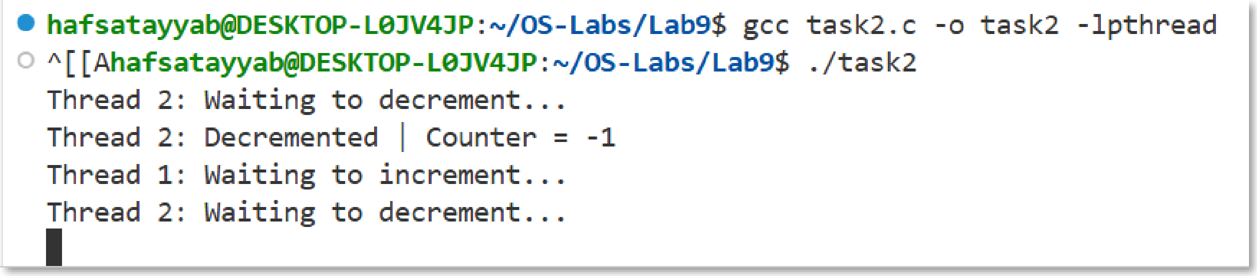
Remarks:

- With; *sem_init*(&*mutex*, 0, 0); *// Binary semaphore initialized to 0*



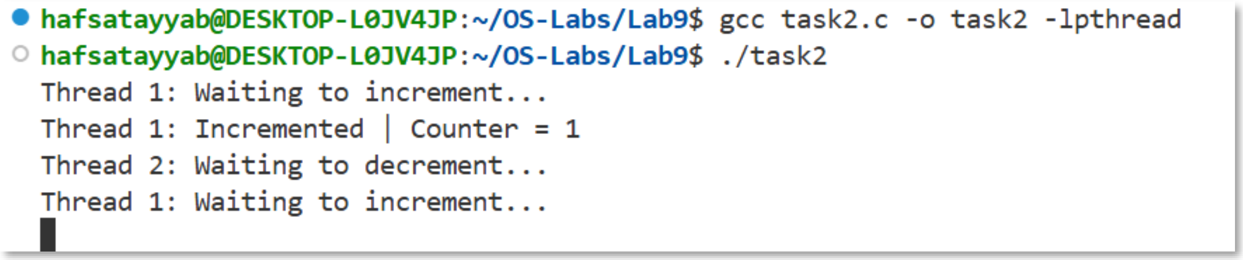
```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task2.c -o task2 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task2
Thread 1: Waiting to increment...
Thread 2: Waiting to decrement...
```

- With; *// sem_post(&mutex); // release* of decrementing function



```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task2.c -o task2 -lpthread
^[[Ahafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task2
Thread 2: Waiting to decrement...
Thread 2: Decremented | Counter = -1
Thread 1: Waiting to increment...
Thread 2: Waiting to decrement...
```

- With; *// sem_post(&mutex); // release* of incrementing function



```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ gcc task2.c -o task2 -lpthread
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab9$ ./task2
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 1: Waiting to increment...
```

Task 3: Comparison

Feature	Mutex (Mutual Exclusion)	Semaphore
Purpose	To enforce mutual exclusion—ensuring that only one thread is inside a critical section at a time.	To limit the number of threads/processes accessing a pool of identical resources simultaneously.
Internal State	Locked (1) or Unlocked (0).	An integer count.
Wait/Acquire	The thread attempts to Lock() the mutex. If locked, the thread is blocked until it's unlocked.	The thread attempts to Decrement() the count. If the count is 0, the thread is blocked.
Signal/Release	The thread attempts to Unlock() the mutex, setting the state to Unlocked.	The thread attempts to Increment() the count, setting one waiting thread free.
Ownership	Strict Ownership. Only the thread that successfully called Lock() can call Unlock().	No Ownership. Any thread (or process) can call Increment() (signal) to release a blocked thread.
Use Case	Protecting a single global data structure (like a linked list) from race conditions.	Implementing a producer-consumer buffer (limiting the number of items) or managing a pool of database connections.