National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Naseer

Submitted by:

Ayema

Reg number:

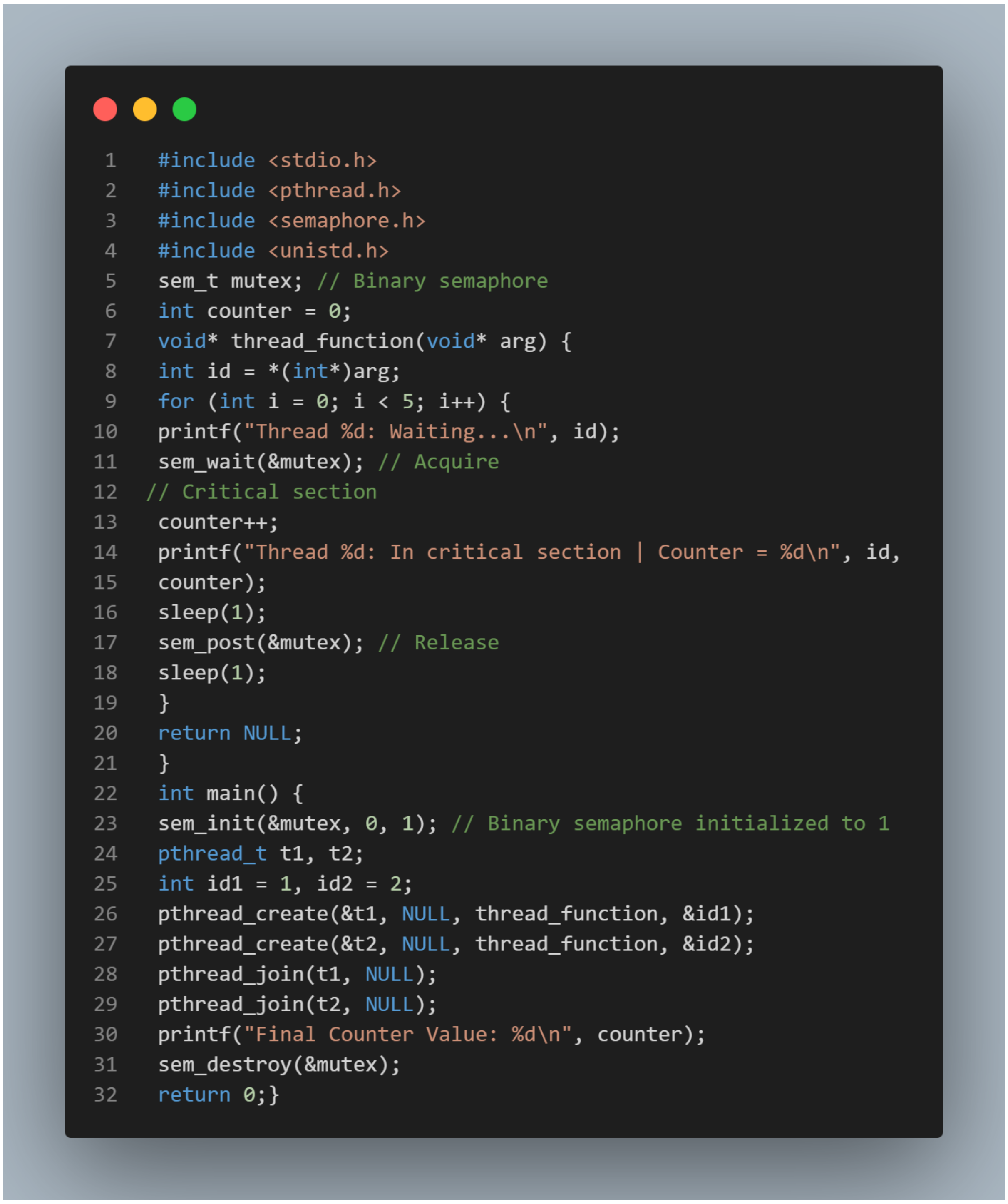
23-NTU-CS-1142

Assignment no. :

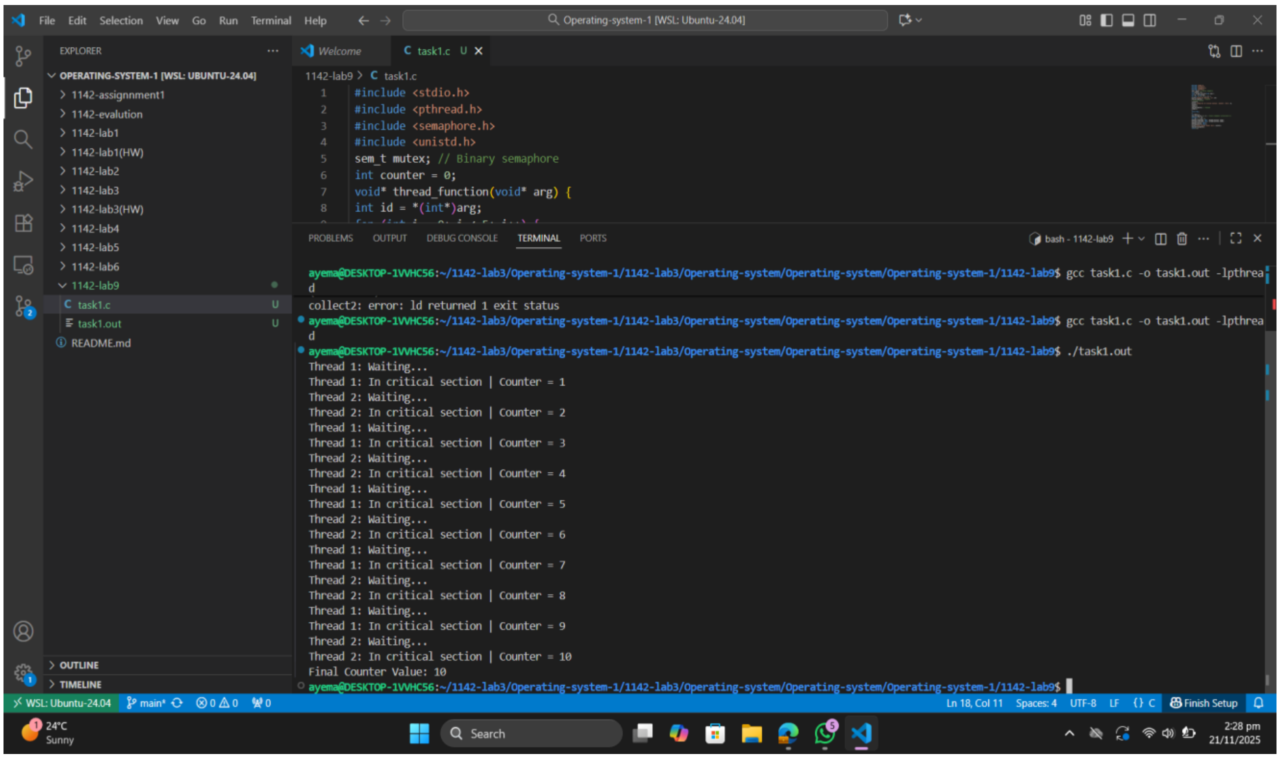
04

Semester: 5Th

Task 1: (Semaphore):



```c
#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;}
```

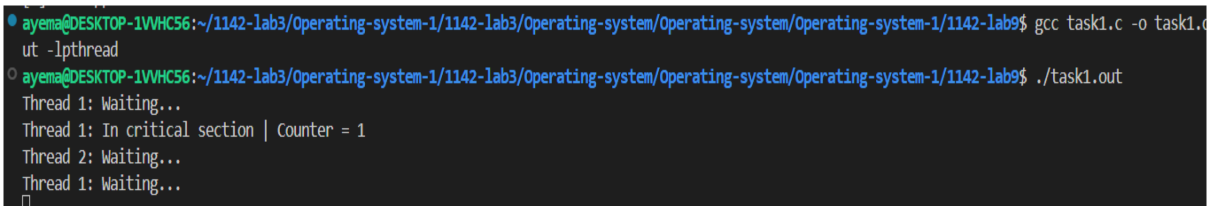


```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void* thread_function(void* arg) {
int id = *(int*)arg;
```

```
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ gcc task1.c -o task1.out -lpthrea
d
collect2: error: ld returned 1 exit status
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ gcc task1.c -o task1.out -lpthrea
d
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ ./task1.out
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
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$
```

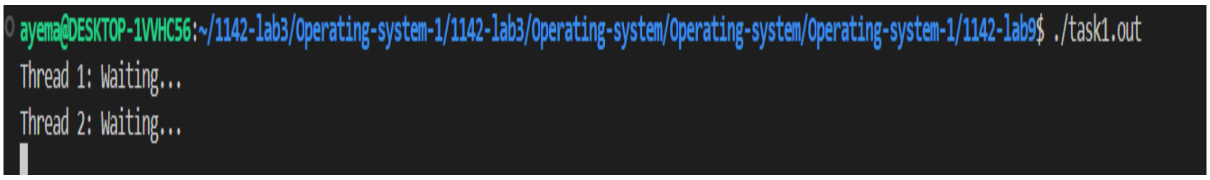
Demonstration:

Removing Sem post→ Woh critical meh hi rahay ga baheer Nahi ai ga sirf ek counter Chalay ga, threads stop forever.



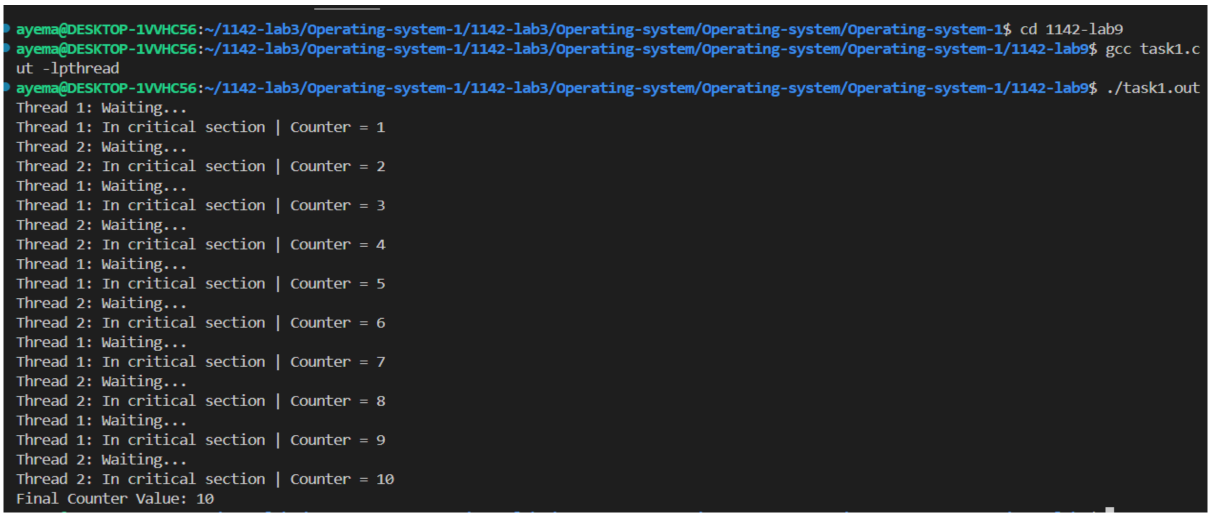
```
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ gcc task1.c -o task1.o
ut -lpthread
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ ./task1.out
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
```

Changing semaphore value from 1 to 0→ both threads enter, race condition. They blocked and waiting for each other.



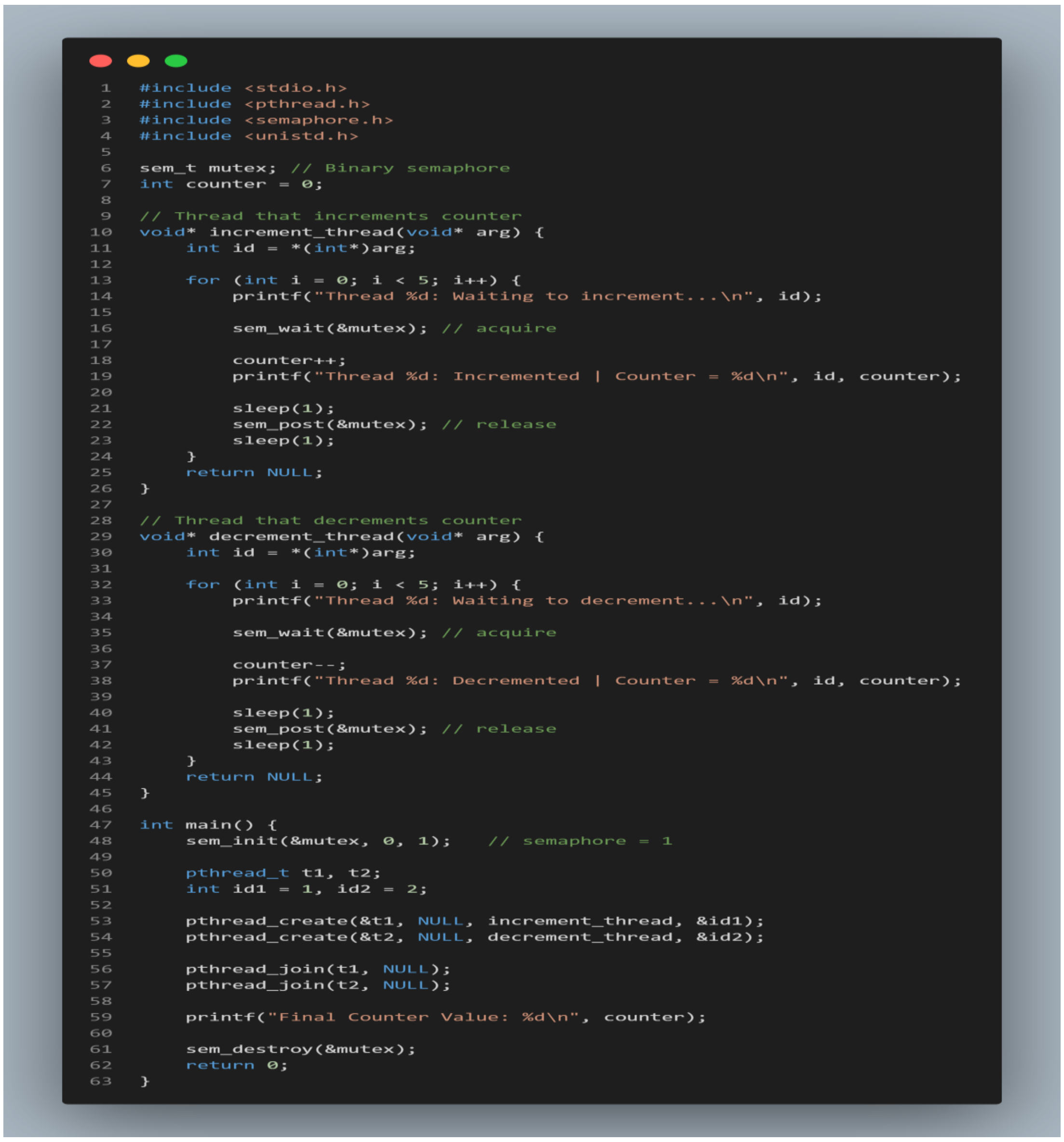
```
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ ./task1.out
Thread 1: Waiting...
Thread 2: Waiting...
```

Removing sem_wait:

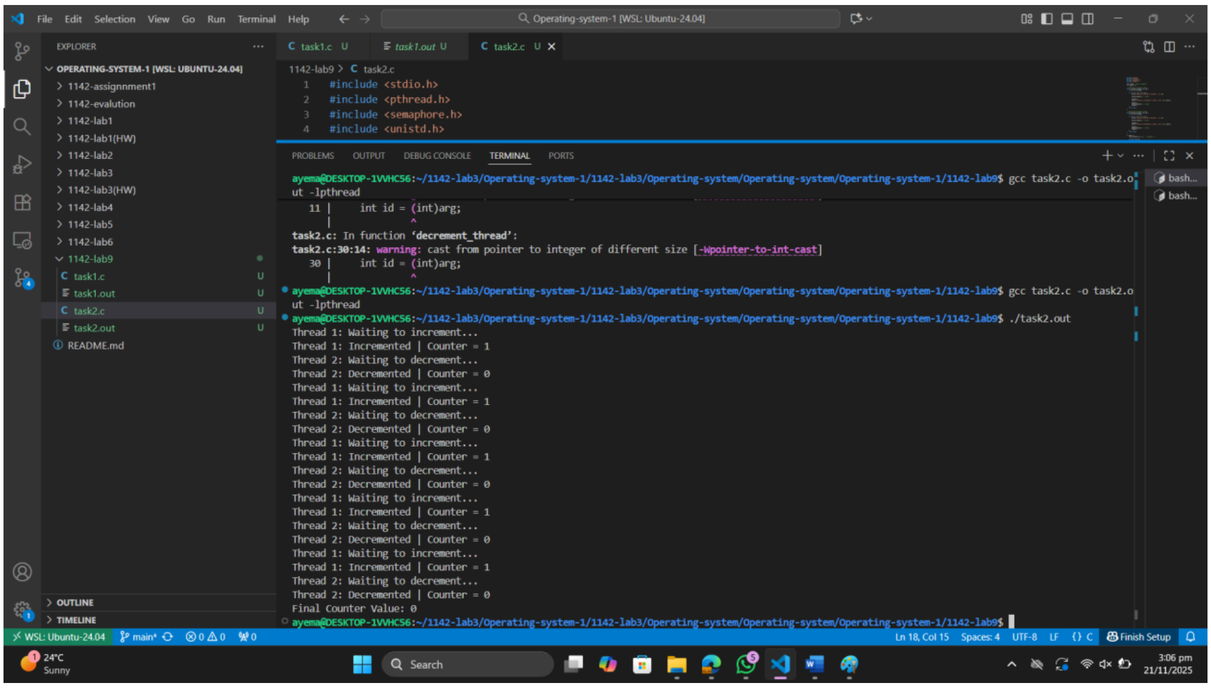


```
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1$ cd 1142-lab9
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ gcc task1.c
ut -lpthread
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ ./task1.out
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
```

Task2 : (Two Function)



```c
#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;
}
```

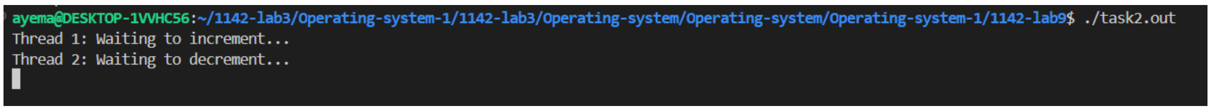


```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
```

```
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ gcc task2.c -o task2.out -lpthread
   11 |     int id = (int)arg;
      |              ^
task2.c: In function ‘decrement_thread’:
task2.c:30:14: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
   30 |     int id = (int)arg;
      |              ^
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ gcc task2.c -o task2.out -lpthread
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ ./task2.out
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
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$
```

Description:

Changing semaphore value from 1 to 0→ both threads enter, race condition. They blocked and waiting for each other.



```
ayema@DESKTOP-1VVHC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system/Operating-system-1/1142-lab9$ ./task2.out
Thread 1: Waiting to increment...
Thread 2: Waiting to decrement...
```

Task 3:

Comparison btw mutex and semaphore:

Feature	Mutex	Semaphore
Definition	Mutual exclusion object for one thread at a time	Signalling mechanism, can allow multiple threads
Count	Binary (locked/unlocked)	Can be counting (value > 1)
Ownership	Owned by the thread that locks it	Not owned by any thread; any thread can signal
Use case	Protect critical section for shared resources	Synchronize threads or processes, control access to N resources
Deadlock risk	Yes, if a thread never unlocks	Yes, if a semaphore is never posted
Performance	Lightweight, faster for thread-level locking	Slightly heavier, works across processes

Example	pthread_mutex_lock/unlock()	sem_wait/sem_post()