`

National Textile University

Department of Computer Science

Subject

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Kashmir Jamshaid

Registration Number

23-NTU-CS-1167

Lab No.

07

Semester

5th

Task_01 :

Input

```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;
}
```

Out put :

```
TERMINAL                                                    bash - Lab_07
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task1.c -o Task1 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./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
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$
```

Code:

```
1   //sem_init(&mutex, 0, 1); // Binary semaphore initia
```

Output:

```
Final Counter Value: 10
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task1.c -o Task1 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./Task1
Thread 1: Waiting...
Thread 2: Waiting...

^Z
[1]+  Stopped                 ./Task1
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$
```

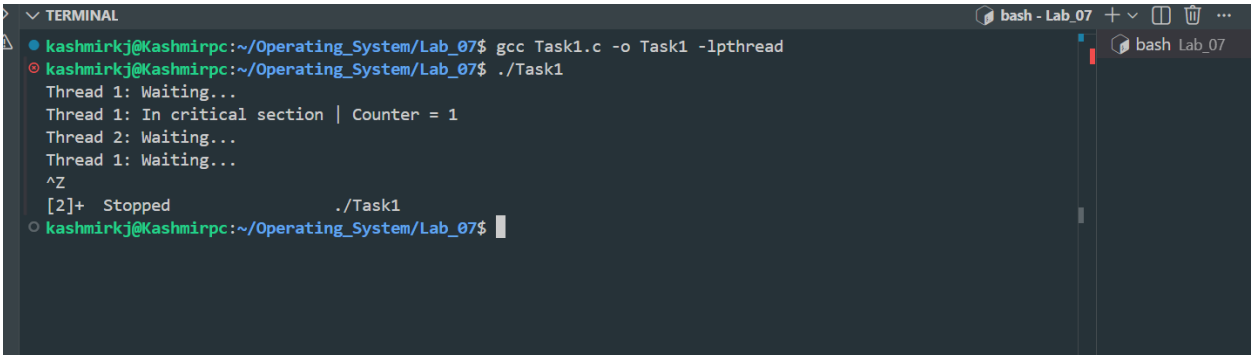
Description:

- **Both threads will block immediately**
- **DEADLOCK occurs** - Neither thread can ever enter the critical section
- **Program hangs forever**

Code :

```
1   //sem_post(&mutex); // Release
```

Output:

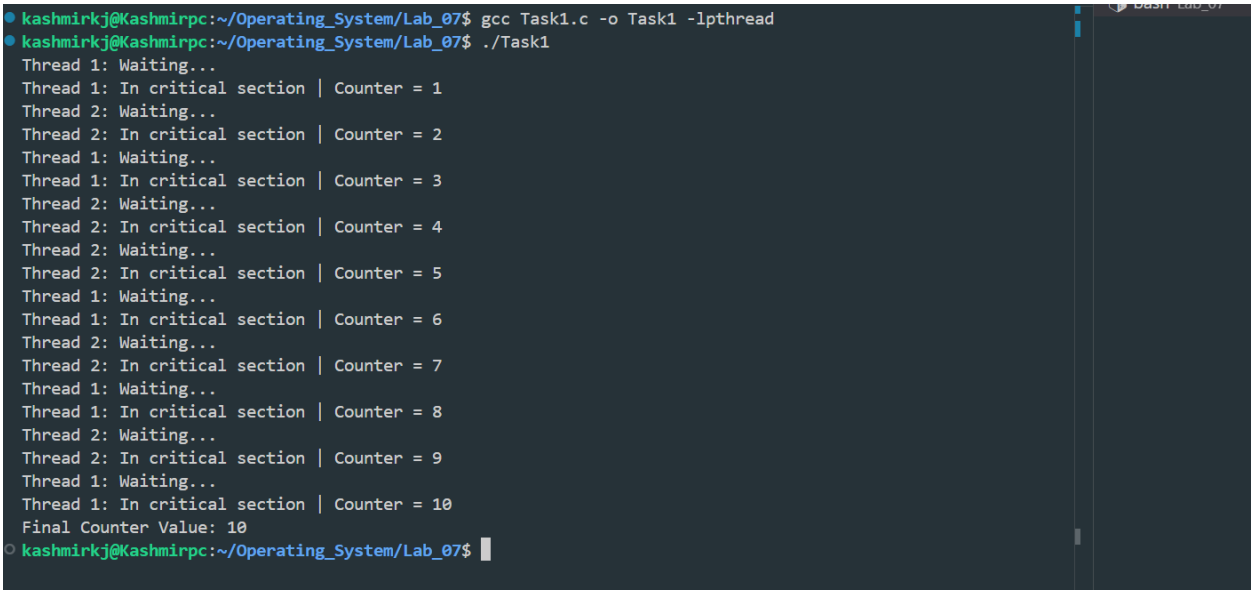


```
TERMINAL                                                    bash - Lab_07
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task1.c -o Task1 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./Task1
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
^Z
[2]+  Stopped                 ./Task1
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$
```

- **First thread enters once, then deadlocks**: Whichever thread gets the semaphore first will enter the critical section, increment counter to 1, but never release the semaphore.
- **Second thread and all subsequent attempts block forever**: The first thread's second iteration and the other thread will all get stuck waiting at sem_wait() since the semaphore is never released.
- **Program hangs**

Code:

Output:



Description:

- **No mutual exclusion**: Both threads can enter the critical section simultaneously, causing potential race conditions on the counter variable.
- **Unpredictable counter increments**: Due to race conditions, the final counter value may be less than 10 (could be anywhere from 2-10, though likely close to 10 in practice).

Task-2

Input:

```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;
}
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); // 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_function2, &id2);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final Counter Value: %d\n", counter);
sem_destroy(&mutex);
return 0;
}

```

Output:

```
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task2.c -o Task2 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./Task2
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Final Counter Value: 0
```

Code:

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

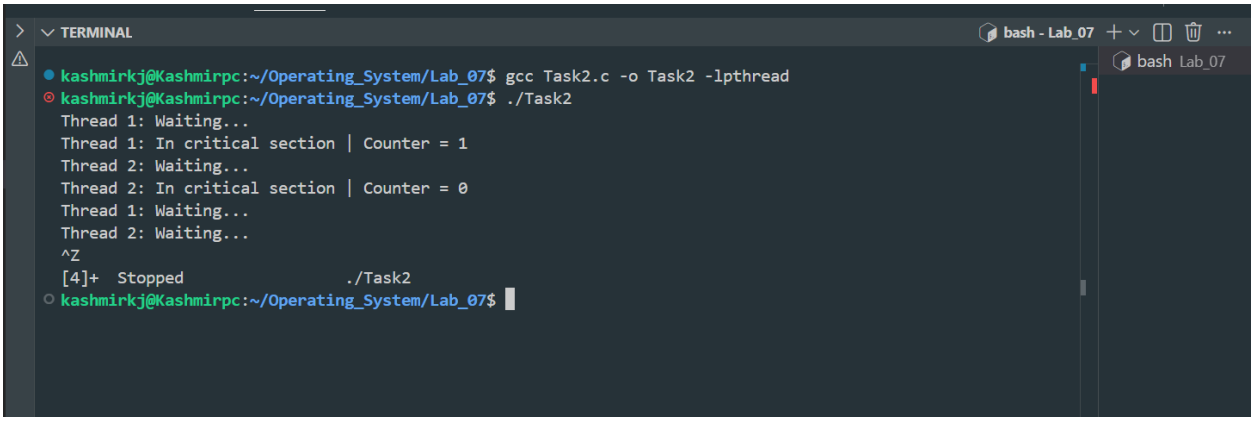
Output:

```
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task2.c -o Task2 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./Task2
Thread 1: Waiting...
Thread 2: Waiting...
^Z
[3]+  Stopped                 ./Task2
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$
```

2. Commit the *//sem_post(&mutex); in 2nd function i.e is in Counter –*

Code:

Output



Commit the *//sem_post(&mutex); in First function i.e is in Counter ++*

Code:

Output:

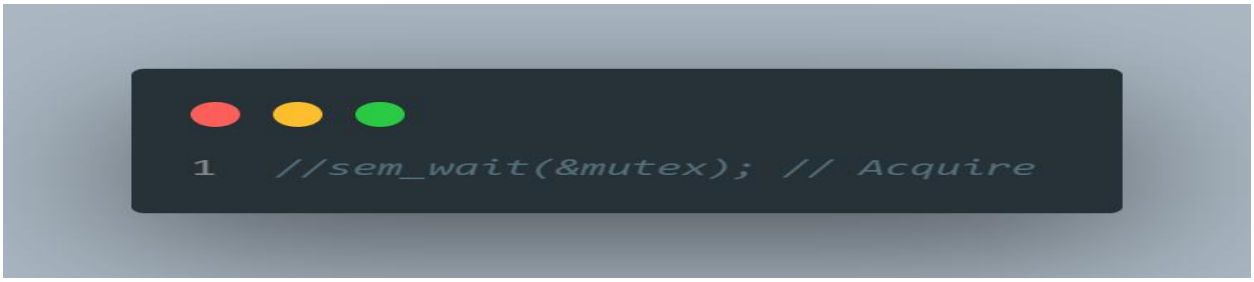
```
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task2.c -o Task2 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./Task2
Thread 2: Waiting...
Thread 2: In critical section | Counter = -1
Thread 1: Waiting...
Thread 1: In critical section | Counter = 0
Thread 2: Waiting...
Thread 1: Waiting...
^Z
[5]+  Stopped                 ./Task2
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$
```

Description:

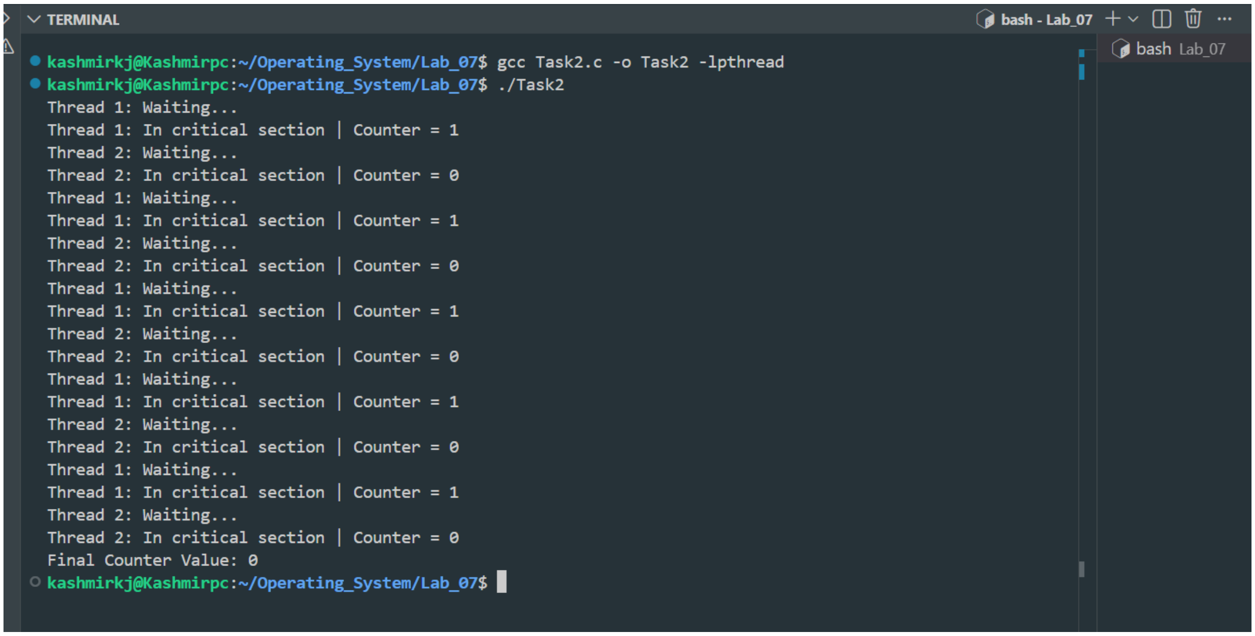
- Thread 1 acquires semaphore on first iteration but never releases it
- Thread 2 gets stuck waiting after Thread 1's first entry, deadlock occurs
- Final counter = 1 (only Thread 1's first increment completes), program hangs

Commit the *//sem_wait(&mutex); Second function i.e is in Counter – -*

Code:



Output:



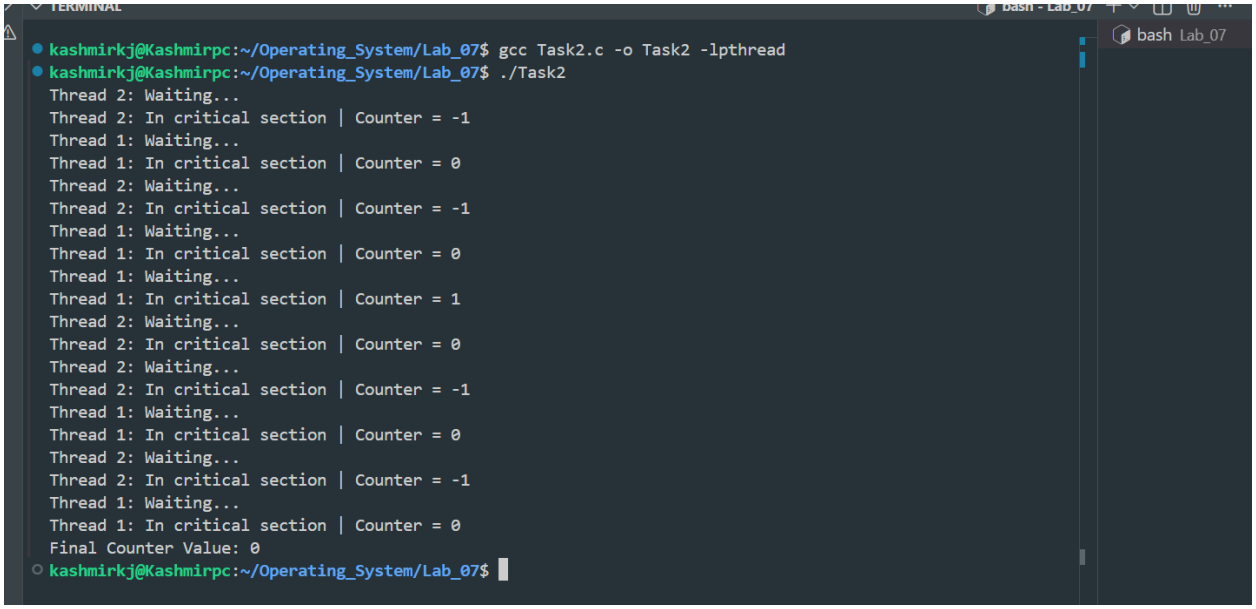
```
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ gcc Task2.c -o Task2 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$ ./Task2
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 0
Final Counter Value: 0
kashmirkj@Kashmirpc:~/Operating_System/Lab_07$
```

Commit the *//sem_wait(&mutex); First function i.e is in Counter ++*

Code:

```
//sem_wait(&mutex); // Acquire
```

Output:



Description:

- Thread 1 enters critical section without acquiring semaphore, while Thread 2 properly waits
- Race condition: Thread 1 can increment while Thread 2 is decrementing (both in critical section simultaneously)
- Final counter unpredictable (likely close to 0 but not guaranteed), Thread 1's increments may be lost due to concurrent access

Task _3 Comparison between Binary Semaphore Mutex?

#	Difference	Binary Semaphore	Mutex	Why it matters in practice
1	Who can release it?	Any thread or process can call sem_post()	Only the thread that locked it can unlock	If the wrong thread unlocks a mutex → crash or deadlock. This is the biggest difference.
2	Ownership	No owner at all	Has a strict owner (thread ID stored)	Mutex prevents accidental unlocks by other threads → safer for protecting data.

#	Difference	Binary Semaphore	Mutex	Why it matters in practice
3	Priority Inversion Protection	Usually none (high-priority threads can be blocked)	Automatic priority inheritance (Linux futex does it)	With semaphores, high-priority threads can starve; mutexes reduce this problem.
4	Intended Use	Signaling / notifications (e.g., “work is done”)	Mutual exclusion (protect counters, lists, shared memory)	Using a semaphore for shared data protection can cause bugs.
5	Behavior if owner thread dies	Stays locked forever → program hangs	Can be robust →	