National Textile University

Department of Computer Science

Subject: Operating System

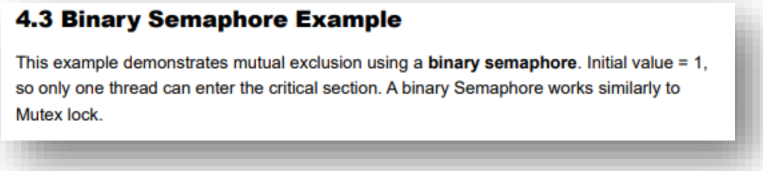
Submitted to: Sir Nasir

Submitted by: Amina

Reg. number: 23-NTU-CS-1136

Lab no.: lab9

Semester:5th



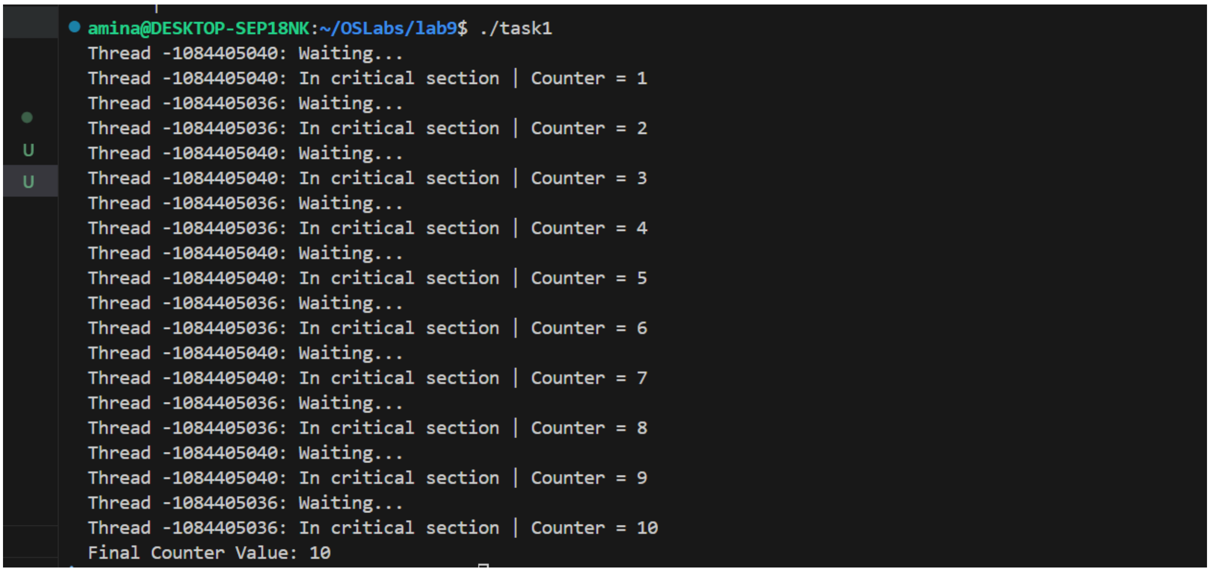
4.3 Binary Semaphore Example

This example demonstrates mutual exclusion using a **binary semaphore**. Initial value = 1, so only one thread can enter the critical section. A binary Semaphore works similarly to Mutex lock.

```
#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;
 }
```

Terminal:



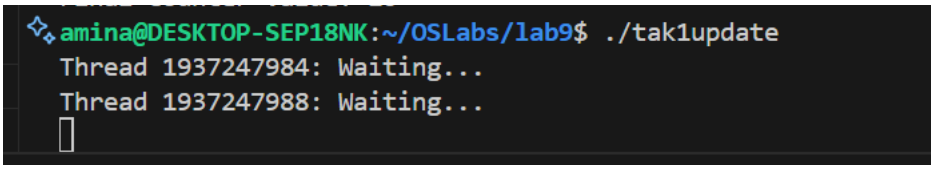
```
amina@DESKTOP-SEP18NK:~/OSLabs/lab9$ ./task1
Thread -1084405040: Waiting...
Thread -1084405040: In critical section | Counter = 1
Thread -1084405036: Waiting...
Thread -1084405036: In critical section | Counter = 2
Thread -1084405040: Waiting...
Thread -1084405040: In critical section | Counter = 3
Thread -1084405036: Waiting...
Thread -1084405036: In critical section | Counter = 4
Thread -1084405040: Waiting...
Thread -1084405040: In critical section | Counter = 5
Thread -1084405036: Waiting...
Thread -1084405036: In critical section | Counter = 6
Thread -1084405040: Waiting...
Thread -1084405040: In critical section | Counter = 7
Thread -1084405036: Waiting...
Thread -1084405036: In critical section | Counter = 8
Thread -1084405040: Waiting...
Thread -1084405040: In critical section | Counter = 9
Thread -1084405036: Waiting...
Thread -1084405036: In critical section | Counter = 10
Final Counter Value: 10
```

When both zero:

```
#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, 0); // 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;
 }
```

Terminal



```
amina@DESKTOP-SEP18NK:~/OSLabs/lab9$ ./tak1update
Thread 1937247984: Waiting...
Thread 1937247988: Waiting...
```

1. When sem_post commented:

- sem_wait() **locks** the semaphore.
- But since sem_post() is removed, the thread **never unlocks** it.
- So the semaphore stays **0 forever**.
- The **first thread** enters the critical section.
- The **second thread** will wait forever (blocked).

2. If sem_wait() is commented :

- Threads **do not wait** for the semaphore.
- Both threads enter the critical section **at the same time**.
- No control is there on shared counter.

Task2:

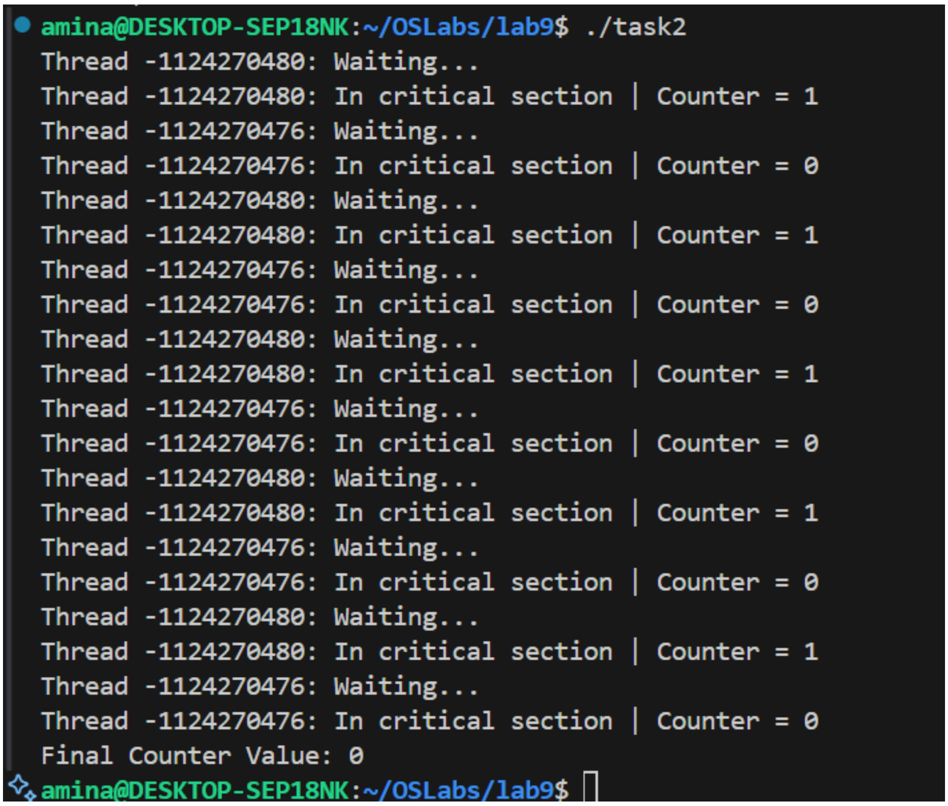
```
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;

void* thread_increament(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_decrement(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_increament, &id1);
pthread_create(&t2, NULL, thread_decrement, &id2);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final Counter Value: %d\n", counter);
sem_destroy(&mutex);
return 0;
}
```

Terminal:



```
amina@DESKTOP-SEP18NK:~/OSLabs/lab9$ ./task2
Thread -1124270480: Waiting...
Thread -1124270480: In critical section | Counter = 1
Thread -1124270476: Waiting...
Thread -1124270476: In critical section | Counter = 0
Thread -1124270480: Waiting...
Thread -1124270480: In critical section | Counter = 1
Thread -1124270476: Waiting...
Thread -1124270476: In critical section | Counter = 0
Thread -1124270480: Waiting...
Thread -1124270480: In critical section | Counter = 1
Thread -1124270476: Waiting...
Thread -1124270476: In critical section | Counter = 0
Thread -1124270480: Waiting...
Thread -1124270480: In critical section | Counter = 1
Thread -1124270476: Waiting...
Thread -1124270476: In critical section | Counter = 0
Thread -1124270480: Waiting...
Thread -1124270480: In critical section | Counter = 1
Thread -1124270476: Waiting...
Thread -1124270476: In critical section | Counter = 0
Final Counter Value: 0
amina@DESKTOP-SEP18NK:~/OSLabs/lab9$
```

Difference:

Feature	Semaphore	Mutex
Primary Purpose	Control access to resources or send signals between threads	Ensure only one thread accesses a recourse at a time
When to Use	When multiple threads may share a resource, or you need signaling	When only ONE thread must access a resource at a time
Resource Type	shared buffers	strict protection
Ownership	can be released by any thread	Owned by the thread that locks it
Allows Multiple Threads	Yes (counting semaphore allows N threads)	No, only 1 thread at a time
Typical Use Cases	Producer–consumer	Shared/Protection variables, file writing, critical sections
Deadlock Risk	Lower	Higher if a thread forgets to unlock

Signaling Between Threads	Yes, used for notifications	Not used for signaling, only locking
Initialization Value	Can be > 1	Always starts at 1