NATIONAL TEXTILE

UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

SUBMITTED BY:

Eman Faisal 23-NTU-CS-1149

SECTION SE: 5th (A)

Operating System-Lab plan 9

SUBMITTED TO:

Sir Nasir Mahmood

SUBMISSION DATE: 21/11/25

Task1:
Default 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);
 // 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;
 }
```

Output:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.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
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9#
```

Change:

Initialize s with 0:
Code:

```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);
 // 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;
}
```

Output:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
Thread 1: Waiting...
Thread 2: Waiting...
```

Description:

Now when we initialize s with 0**,both threads are waiting as the wait is blocked** ,it is not decrementing S variable because the value is 0.The value has to be 1 for the wait to decrement it.

Commenting wait:

```
#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);
// 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;
}
```

Output:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 2
Thread 2: Waiting...
Thread 2: In critical section | Counter = 3
Thread 1: Waiting...
Thread 1: 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...
```

Now,in this we can see that thread2 has the critical section twice in a row here.which means there is no wait check here.it will let any thread access critical section anytime.

Commenting post:

```
#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);
 // 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;
 }
```

Output:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
```

In this,The critical section is accessed just once.This is because when the thread leaves critical section,the value of s=0.As post is commented,the s is not incremented.Now the wait is also blocked as s is not equal to 1.

Task2:

```c
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function1(void* arg) {         //incrementng function
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
    printf("Thread %d: Waiting...\n", id);
    sem_wait(&mutex);
    // 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) {        //decrementing function
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
    printf("Thread %d: Waiting...\n", id);
    sem_wait(&mutex);
    // Critical section
    counter--;
```

```c
    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_function1, &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;
}
//here the final counter value is 0.whatever is incremented is also decremented
// by other thread on the other side.
```

Output:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
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
```

Changing initialization to 0:

```c
#include <stdio.h>

 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function1(void* arg) {         //incrementng function
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
    printf("Thread %d: Waiting...\n", id);
    sem_wait(&mutex);
    // 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) {        //decrementing function
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
    printf("Thread %d: Waiting...\n", id);
    sem_wait(&mutex);
    // 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_function1, &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;
}
//here the final counter value is 0.whatever is incremented is also decremented
// by other thread on the other side.
```

Output:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# gcc task2.c
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
Thread 1: Waiting...
Thread 2: Waiting...
```

Description:

Now when we initialize s with 0**,both threads are waiting as the wait is blocked** ,it is not decrementing S variable because the value is 0.The value has to be 1 for the wait to decrement it.

Commenting wait:

```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
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 2: Waiting...
Thread 2: In critical section | Counter = -1
Thread 1: Waiting...
Thread 1: In critical section | Counter = 0
Thread 2: Waiting...
Thread 2: In critical section | Counter = -1
Thread 1: Waiting...
Thread 1: 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
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/Lab9# ./a.out
```

Description:

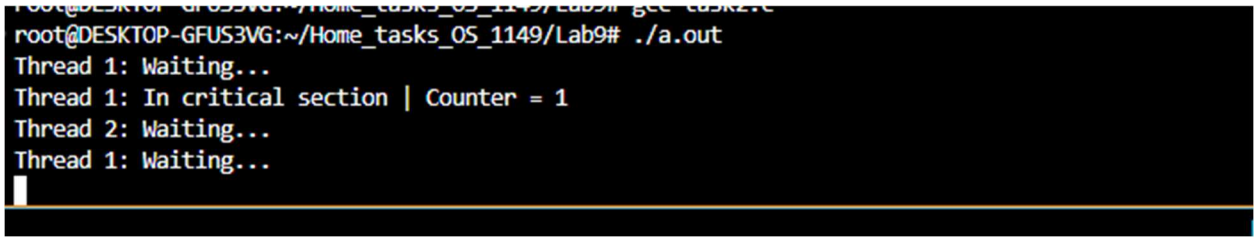
here the count value is even going to -1,because any thread can access the critical section anytime and even constantly.like thread2 does access it two times in a row and so the counter value goes to -1.

Commenting post:

```
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function1(void* arg) {         //incrementng function
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
    printf("Thread %d: Waiting...\n", id);
    sem_wait(&mutex);
    // 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) {        //decrementing function
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
    printf("Thread %d: Waiting...\n", id);
    sem_wait(&mutex);
    // 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
```

```c
    pthread_t t1, t2;
    int id1 = 1, id2 = 2;
    pthread_create(&t1, NULL, thread_function1, &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;
}
//here the final counter value is 0.whatever is incremented is also decremented
// by other thread on the other side.
```

Output:



Description:

In this,The critical section is accessed just **once**.This is because when the thread leaves critical section,the value of s=0.As post is commented,the s is not incremented.Now the wait is also blocked as s is not equal to 1.

Task3:

Difference between Mutex and Binary Semaphore

MUTEX	BINARY SEMAPHORE
It uses lock and unlock	It uses wait and post
Owned by the thread/process that locks it. Only the owner can unlock it.	No ownership. Any thread/process can signal (release) it.
Ensures only one thread must access resource at a time.	Control access to resources or send signals between threads.
Use case: Shared variables/critical section	Usecase: producer consumer