



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Eman Marium Tariq Rao

Reg number:

23-NTU-CS-1150

Semester:

05

Semaphore Operations

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  sem_t mutex; // Binary semaphore
6  int counter = 0;
7  void* thread_function(void* arg) {
8      int id = *(int*)arg;
9      for (int i = 0; i < 5; i++) {
10         printf("Thread %d: Waiting...\n", id);
11         sem_wait(&mutex); // Acquire
12         // Critical section
13         counter++;
14         printf("Thread %d: In critical section | Counter = %d\n", id,
15         counter);
16         sleep(1);
17         sem_post(&mutex); // Release
18         sleep(1);
19     }
20     return NULL;
21 }
22 int main() {
23     sem_init(&mutex, 0, 1); // Binary semaphore initialized to 1
24     pthread_t t1, t2;
25     int id1 = 1, id2 = 2;
26     pthread_create(&t1, NULL, thread_function, &id1);
27     pthread_create(&t2, NULL, thread_function, &id2);
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30     printf("Final Counter Value: %d\n", counter);
31     sem_destroy(&mutex);
32     return 0;
33 }
```

Terminal:

The screenshot shows a terminal window with the following interface elements at the top:

- PROBLEMS
- OUTPUT
- DEBUG CONSOLE
- TERMINAL** (underlined)
- PORTS

On the right side of the terminal window, there is a small icon and the text "bash - lab9".

The terminal output is as follows:

```
● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ gcc program1.c -o program1 -lpthread
● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ ./program1
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
◆ eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ []
```

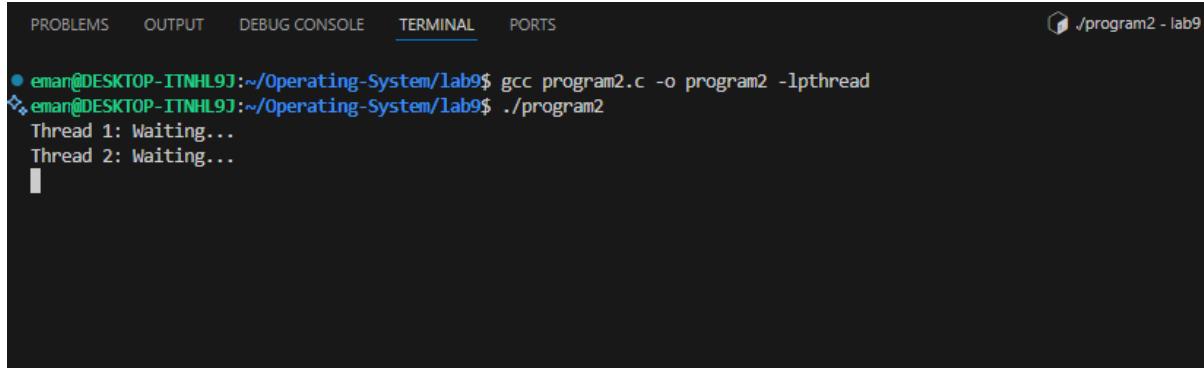
Initializing Semaphore with 0:

Explanation: As we have initialized with 0 the threads will be blocked ie both will keep waiting

```
● ● ●

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 sem_t mutex; // Binary semaphore
6 int counter = 0;
7 void* thread_function(void* arg) {
8     int id = *(int*)arg;
9     for (int i = 0; i < 5; i++) {
10        printf("Thread %d: Waiting...\n", id);
11        sem_wait(&mutex); // Acquire
12        // Critical section
13        counter++;
14        printf("Thread %d: In critical section | Counter = %d\n", id,
15               counter);
16        sleep(1);
17        sem_post(&mutex); // Release
18        sleep(1);
19    }
20    return NULL;
21 }
22 int main() {
23     sem_init(&mutex, 0, 0); // Binary semaphore initialized to 1
24     pthread_t t1, t2;
25     int id1 = 1, id2 = 2;
26     pthread_create(&t1, NULL, thread_function, &id1);
27     pthread_create(&t2, NULL, thread_function, &id2);
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30     printf("Final Counter Value: %d\n", counter);
31     sem_destroy(&mutex);
32     return 0;
33 }
34
```

Terminal:



The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ gcc program2.c -o program2 -lpthread
✧ eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ ./program2
Thread 1: Waiting...
Thread 2: Waiting...
```

Commenting Post:

If you comment out `sem_post()`, the thread **never releases the semaphore**, so the second thread can **never enter the critical section**.

This causes the program to **freeze (deadlock)** forever.

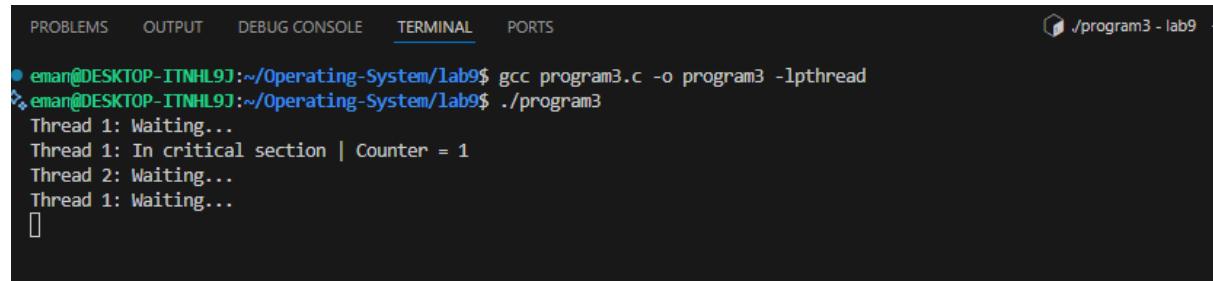
```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex; // Binary semaphore
int counter = 0;
void* thread_function(void* arg) {
    int id = *(int*)arg;
    for (int i = 0; i < 5; i++) {
        printf("Thread %d: Waiting...\n", id);
        sem_wait(&mutex);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



```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ gcc program3.c -o program3 -lpthread
eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ ./program3
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 1: Waiting...
[]


```

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); // 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:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash - lab9

● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ ./program4
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 2: Waiting...
Thread 2: In critical section | Counter = 9
Thread 1: Waiting...
Thread 1: In critical section | Counter = 10
Final Counter Value: 10
eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ []

```

Creating Two thread Functions:

```

#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: Decrement | 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);

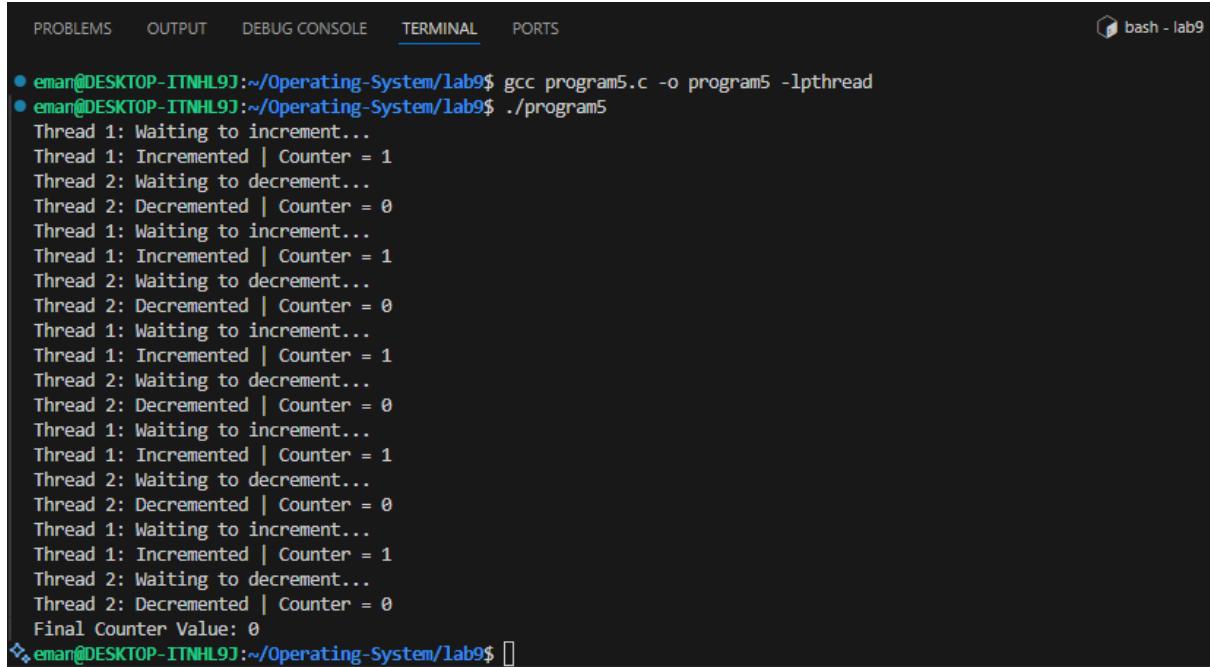
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Final Counter Value: %d\n", counter);

    sem_destroy(&mutex);
    return 0;
}

```

Terminal:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash - lab9

● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ gcc program5.c -o program5 -lpthread
● eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$ ./program5
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decrement | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decrement | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decrement | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decrement | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decrement | Counter = 0
Thread 1: Waiting to increment...
Thread 1: Incremented | Counter = 1
Thread 2: Waiting to decrement...
Thread 2: Decrement | Counter = 0
Final Counter Value: 0
eman@DESKTOP-ITNHL9J:~/Operating-System/lab9$
```

Difference between Semaphore and Mutex:

Feature	Mutex	Semaphore
Meaning	A lock	A signaling mechanism (counter)
Ownership	Owned by a thread – only the thread that locked it can unlock it	Not owned – any thread can signal (V) or wait (P)
Value	Only 0 or 1 (binary)	Can be 0, 1, 2, ... N (counting)
Purpose	Protect a shared resource	Control access to multiple resources or coordination
When used?	When only ONE thread should access a critical section	When multiple threads can access up to a limit
Deadlock risk	Higher (if same thread doesn't unlock)	Lower
Example use	Protect a shared variable	Limit access to database connections (e.g., only 10 allowed)