

Ex. No:	5	IMPLEMENTATION OF MUTUAL EXCLUSION BY SEMAPHORE
Date:		

AIM:

To implement mutual exclusion using a binary semaphore in a multithreaded program to ensure that only one thread accesses the critical section at a time.

ALGORITHM:

Step 1: Start the program and include the required header files:

stdio.h for input/output functions.

- pthread.h for thread operations.
- semaphore.h for semaphore functions.
- unistd.h for the sleep() function.

Step 2: Declare a binary semaphore (e.g., sem_t mutex) globally to control access to the critical section.

Step 3: Define the thread function which performs the following:

Print a message indicating the thread is waiting.

Use sem_wait(&mutex) to request access to the critical section.

Once allowed, enter the critical section, print messages, and simulate work using sleep().

Exit the critical section and use sem_post(&mutex) to release the semaphore.

Step 4: In the main() function:

Initialize the semaphore using sem_init(&mutex, 0, 1) where the initial value is 1 (binary).

Create two (or more) threads using pthread_create() and pass the thread function.

Step 5: Wait for all threads to finish using pthread_join().

Step 6: Destroy the semaphore using sem_destroy(&mutex) to clean up resources.

Step 7: End the program.

CODE:

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

sem_t mutex;
```

```
void* thread_function(void* arg)
{
    int thread_num = *(int*)arg;
    printf("Thread %d is waiting to enter the critical section...\n", thread_num);
    // Wait (P operation) on semaphore
    sem_wait(&mutex);
    // Start of Critical Section
    printf("Thread %d has entered the critical section.\n", thread_num);
    sleep(2); // Simulate work in critical section
    printf("Thread %d is leaving the critical section.\n", thread_num);
    // End of Critical Section
    // Signal (V operation) on semaphore
    sem_post(&mutex);
    return NULL;
}

int main()
{
    pthread_t t1, t2;
    int thread_id1 = 1, thread_id2 = 2;
    // Initialize the semaphore with value 1 (binary semaphore)
    sem_init(&mutex, 0, 1);
    // Create two threads
    pthread_create(&t1, NULL, thread_function, &thread_id1);
    pthread_create(&t2, NULL, thread_function, &thread_id2);
    // Wait for both threads to complete
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    // Destroy the semaphore
}
```

```
sem_destroy(&mutex);

printf("Both threads have finished execution.\n");

return 0;

}
```

OUTPUT:

```
> nano mm.c
> gcc -o mm mm.c
> ./mm
Thread 1 is waiting to enter the critical section ...
Thread 1 has entered the critical section.
Thread 2 is waiting to enter the critical section ...
Thread 1 is leaving the critical section.
Thread 2 has entered the critical section.
Thread 2 is leaving the critical section.
Both threads have finished execution.
```

RUBRICS FOR EVALUATION	MAXIMUM	AWARDED
Fundamental Knowledge	2	
Design of Experiment	2	
Implementation	4	
Viva	2	
Total	10	
Signature		

RESULT:

Thus, the program was executed successfully. Mutual exclusion was implemented using a semaphore, ensuring that only one thread accessed the **critical section** at a time, and the shared counter was updated correctly without race conditions.