praveenkumar.R
192324012

Experiment-20:Construct a C program to simulate Reader-Writer problem using Semaphores.

Aim:

To simulate the **Reader-Writer problem** using **semaphores**. The problem involves multiple readers and writers accessing a shared resource. The solution uses semaphores to ensure that multiple readers can read the resource concurrently, but writers have exclusive access when writing, preventing race conditions.

Procedure:

1. Use two semaphores: mutex for controlling access to the read_count, and write_mutex for ensuring exclusive access for writers.
2. Readers increment and decrement a read_count. If it's the first reader, it acquires write_mutex, and if it's the last reader, it releases write_mutex.
3. Writers acquire write_mutex to gain exclusive access to the resource and release it when done.
4. Readers and writers run concurrently, with synchronization ensuring mutual exclusion where necessary.

Code Implementation:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <unistd.h>


sem_t mutex, write_mutex;

int read_count = 0;


void* reader(void* param) {

   while(1) {

      sem_wait(&mutex);

      read_count++;

      if(read_count == 1) {
```

```
            sem_wait(&write_mutex);
        }
        sem_post(&mutex);

        printf("Reader is reading the resource\n");
        sleep(1);

        sem_wait(&mutex);
        read_count--;
        if(read_count == 0) {
            sem_post(&write_mutex);
        }
        sem_post(&mutex);

        sleep(1);
    }
}

void* writer(void* param) {
    while(1) {
        sem_wait(&write_mutex);
        printf("Writer is writing to the resource\n");
        sleep(2);
        sem_post(&write_mutex);

        sleep(1);
    }
}

int main() {
    pthread_t reader_threads[5], writer_threads[2];
```

```
    sem_init(&mutex, 0, 1);
    sem_init(&write_mutex, 0, 1);

    for(int i = 0; i < 5; i++) {
        pthread_create(&reader_threads[i], NULL, reader, NULL);
    }
    for(int i = 0; i < 2; i++) {
        pthread_create(&writer_threads[i], NULL, writer, NULL);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(reader_threads[i], NULL);
    }
    for(int i = 0; i < 2; i++) {
        pthread_join(writer_threads[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&write_mutex);

    return 0;
}
```

Output:

Output

```
Reader is reading the resource
Reader is reading the resource
Reader is reading the resource
Writer is writing to the resource
Reader is reading the resource
Writer is writing to the resource
Reader is reading the resource
```