B. Siva Shirish-192324016

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

AIM

To construct a C program to simulate the Reader-Writer problem using semaphores, ensuring synchronization between readers and writers.

ALGORITHM

1. Start.
2. Initialize semaphores for mutual exclusion and resource access.
3. Initialize variables for counting readers.
4. For each reader:
    - Wait for mutual exclusion.
    - Increment reader count.
    - If it's the first reader, wait for the resource semaphore.
    - Signal mutual exclusion.
    - Perform reading.
    - Wait for mutual exclusion.
    - Decrement reader count.
    - If it's the last reader, signal the resource semaphore.
    - Signal mutual exclusion.
5. For each writer:
    - Wait for the resource semaphore.
    - Perform writing.
    - Signal the resource semaphore.
6. Synchronize reader and writer threads.
7. End.

AIM

To construct a C program to simulate the Reader-Writer problem using semaphores, ensuring synchronization between readers and writers.

ALGORITHM

1. Start.
2. Initialize semaphores for mutual exclusion and resource access.
3. Initialize variables for counting readers.
4. For each reader:
    - Wait for mutual exclusion.
    - Increment reader count.
    - If it's the first reader, wait for the resource semaphore.
    - Signal mutual exclusion.
    - Perform reading.
    - Wait for mutual exclusion.
    - Decrement reader count.
    - If it's the last reader, signal the resource semaphore.
    - Signal mutual exclusion.
5. For each writer:
    - Wait for the resource semaphore.
    - Perform writing.
    - Signal the resource semaphore.
6. Synchronize reader and writer threads.
7. End.

PROCEDURE

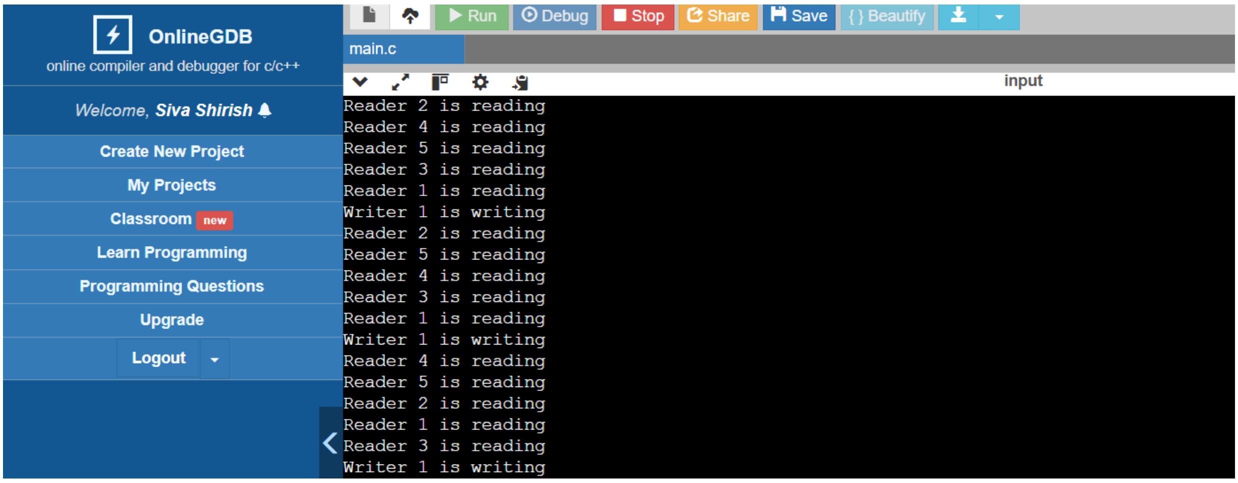
1. Declare and initialize semaphores and shared variables.
2. Create reader and writer threads.
3. Use semaphores to handle critical sections, ensuring no conflicts between readers and writers.

4. Synchronize thread execution.
5. Clean up and terminate.

CODE:

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

#define MAX_READERS 5

sem_t mutex;
sem_t wrt;
int read_count = 0;

void* reader(void* arg) {
    int f = *((int*)arg);
    while (1) {
        sem_wait(&mutex);
        read_count++;
        if (read_count == 1) {
            sem_wait(&wrt);
        }
        sem_post(&mutex);

        printf("Reader %d is reading\n", f);
        sleep(1);

        sem_wait(&mutex);
        read_count--;
        if (read_count == 0) {
            sem_post(&wrt);
        }
        sem_post(&mutex);
        sleep(1);
    }
}

void* writer(void* arg) {
    int f = *((int*)arg);
    while (1) {
        sem_wait(&wrt);
        printf("Writer %d is writing\n", f);
        sleep(1);
        sem_post(&wrt);
        sleep(1);
    }
}

int main() {
    pthread_t read[MAX_READERS], write;
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);

    int reader_ids[MAX_READERS];
    for (int i = 0; i < MAX_READERS; i++) {
        reader_ids[i] = i + 1;
        pthread_create(&read[i], NULL, reader, &reader_ids[i]);
    }
    int writer_id = 1;
    pthread_create(&write, NULL, writer, &writer_id);

    for (int i = 0; i < MAX_READERS; i++) {
        pthread_join(read[i], NULL);
    }
    pthread_join(write, NULL);

    sem_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;
}
```

OUTPUT:



```
Reader 2 is reading
Reader 4 is reading
Reader 5 is reading
Reader 3 is reading
Reader 1 is reading
Writer 1 is writing
Reader 2 is reading
Reader 5 is reading
Reader 4 is reading
Reader 3 is reading
Reader 1 is reading
Writer 1 is writing
Reader 4 is reading
Reader 5 is reading
Reader 2 is reading
Reader 1 is reading
Reader 3 is reading
Writer 1 is writing
```

RESULT

The program successfully simulates the Reader-Writer problem using semaphores, ensuring proper synchronization and mutual exclusion.