D.GUNA VARDHAN - 192372041

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 <pthread.h>
#include <semaphore.h>

sem_t resource, rmutex;
int read_count = 0;

void *reader(void *arg) {
    sem_wait(&rmutex);
    read_count++;
    if (read_count == 1) {
        sem_wait(&resource);
    }
    sem_post(&rmutex);

    printf("Reader %ld is reading.\n", pthread_self());

    sem_wait(&rmutex);
    read_count--;
    if (read_count == 0) {
        sem_post(&resource);
    }
    sem_post(&rmutex);
    return NULL;
}
```

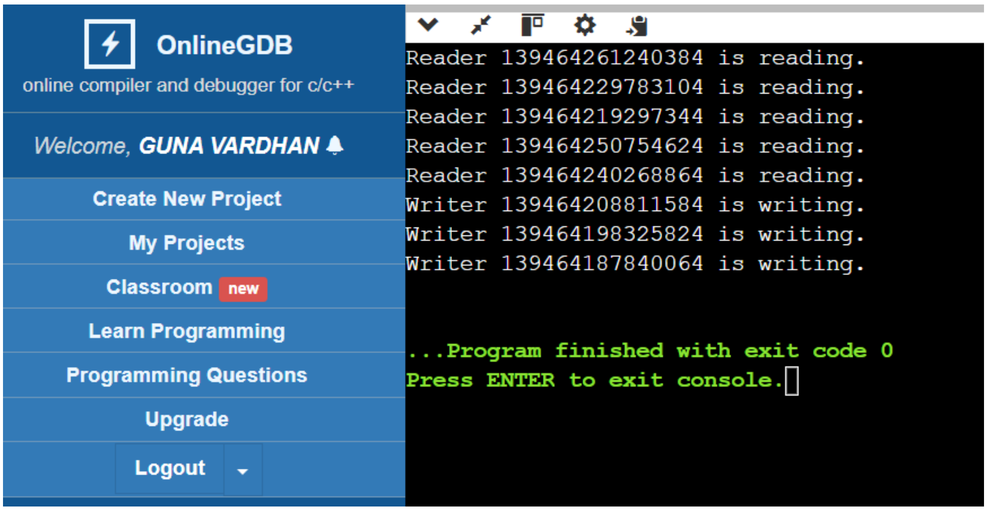
```
void *writer(void *arg) {
    sem_wait(&resource);
    printf("Writer %ld is writing.\n", pthread_self());
    sem_post(&resource);
    return NULL;
}

int main() {
    pthread_t readers[5], writers[3];
    sem_init(&resource, 0, 1);
    sem_init(&rmutex, 0, 1);

    for (int i = 0; i < 5; i++) {
        pthread_create(&readers[i], NULL, reader, NULL);
    }
    for (int i = 0; i < 3; i++) {
        pthread_create(&writers[i], NULL, writer, NULL);
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(readers[i], NULL);
    }
    for (int i = 0; i < 3; i++) {
        pthread_join(writers[i], NULL);
    }

    sem_destroy(&resource);
    sem_destroy(&rmutex);
    return 0;
}
```

OUTPUT:



```
Reader 139464261240384 is reading.
Reader 139464229783104 is reading.
Reader 139464219297344 is reading.
Reader 139464250754624 is reading.
Reader 139464240268864 is reading.
Writer 139464208811584 is writing.
Writer 139464198325824 is writing.
Writer 139464187840064 is writing.


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT

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