**Name:** B. Nageswar

**Reg-No**: 192372005

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

**Aim:**

The aim of the Reader-Writer problem is to manage access to a shared resource where multiple readers can access it simultaneously but writers need exclusive access. We use semaphores to synchronize the readers and writers.

**Algorithm:**

* Readers: Can read simultaneously, but if a writer is writing, they must wait.
* Writers: Must have exclusive access to the resource, meaning no readers or other writers can access it during writing.

**Procedure:**

1. Initialize semaphores:
   * mutex for mutual exclusion (to control access to shared data).
   * write\_lock to ensure exclusive access to the resource for writers.
   * read\_count\_lock for synchronization of the reader count.
2. Readers:
   * Increment the reader count.
   * If it's the first reader, wait for writers.
   * After reading, decrement the reader count.
   * If it's the last reader, signal the writers to proceed.
3. Writers:
   * Wait for the write\_lock to get exclusive access.
   * Perform writing.
   * Signal after writing is done.

### Code:

### #include <stdio.h>

### #include <pthread.h>

### #include <semaphore.h>

### sem\_t mutex, write\_lock, read\_count\_lock;

### int read\_count = 0;

### void\* reader(void\* arg) {

### sem\_wait(&read\_count\_lock);

### read\_count++;

### if (read\_count == 1)

### sem\_wait(&write\_lock);

### sem\_post(&read\_count\_lock);

### printf("Reader is reading\n");

### sem\_wait(&read\_count\_lock);

### read\_count--;

### if (read\_count == 0)

### sem\_post(&write\_lock);

### sem\_post(&read\_count\_lock);

### return NULL;

### }

### void\* writer(void\* arg) {

### sem\_wait(&write\_lock);

### printf("Writer is writing\n");

### sem\_post(&write\_lock);

### return NULL;

### }

### int main() {

### pthread\_t r[5], w[5];

### 

### sem\_init(&mutex, 0, 1);

### sem\_init(&write\_lock, 0, 1);

### sem\_init(&read\_count\_lock, 0, 1);

### for (int i = 0; i < 5; i++) {

### pthread\_create(&r[i], NULL, reader, NULL);

### pthread\_create(&w[i], NULL, writer, NULL);

### }

### for (int i = 0; i < 5; i++) {

### pthread\_join(r[i], NULL);

### pthread\_join(w[i], NULL);

### }

### sem\_destroy(&mutex);

### sem\_destroy(&write\_lock);

### sem\_destroy(&read\_count\_lock);

### return 0;

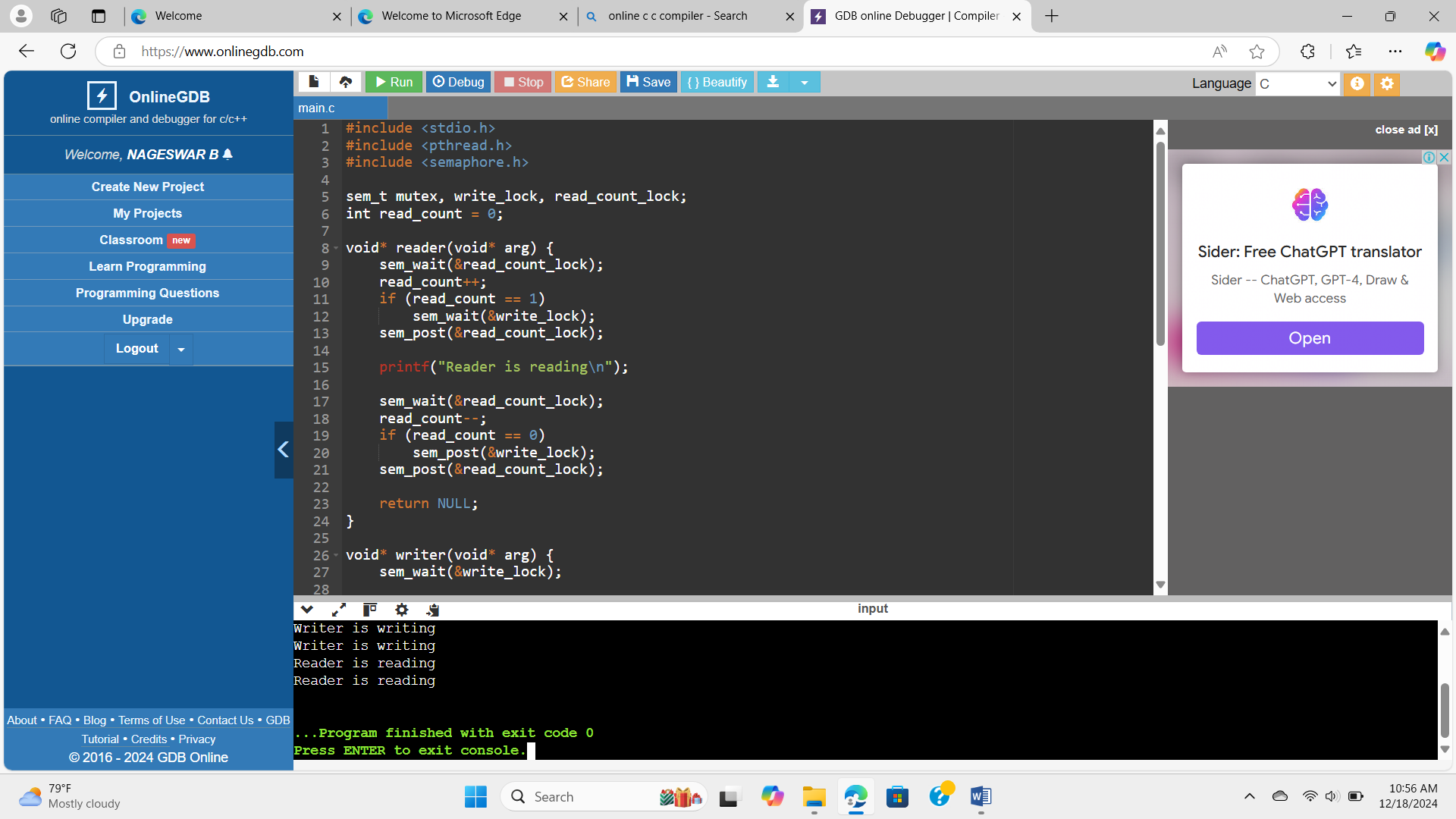
### }

**Result:**

The program simulates multiple readers and writers. It ensures that:

* Multiple readers can access the resource simultaneously.
* A writer has exclusive access, blocking readers when writing.
* Once writing is finished, readers can resume.

**Output:**

****