

Name: P.MALREDDY

Reg-No: 192372015

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;

}

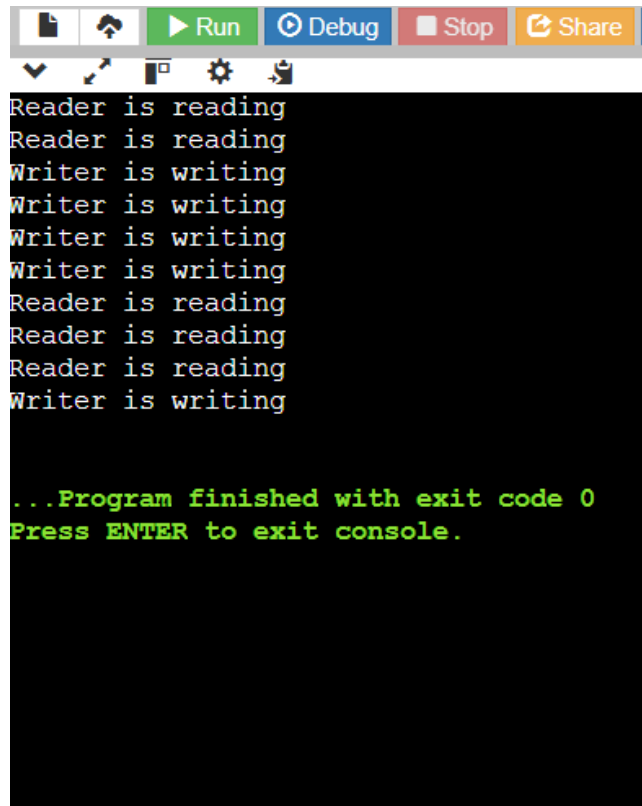
```

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:



The screenshot shows a code editor's interface with a toolbar at the top containing icons for file operations, a 'Run' button (green), a 'Debug' button (blue), a 'Stop' button (red), and a 'Share' button (orange). Below the toolbar is a dark-themed console window displaying the following text:

```
Reader is reading
Reader is reading
Writer is writing
Writer is writing
Writer is writing
Writer is writing
Reader is reading
Reader is reading
Reader is reading
Writer is writing

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