

Name: K.SaiKrishna

Reg-No: 192311106

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;

}

```

Output:

The screenshot shows a code editor with a sidebar on the left containing navigation links: 'Welcome, K Sai Krishna', 'Create New Project', 'My Projects', 'Classroom' (with a 'new' badge), 'Learn Programming', 'Programming Questions', 'Upgrade', and 'Logout'. The main editor area displays C code for a readers-writers problem. The code includes thread creation for 5 readers and 3 writers, joining them, and destroying semaphores. The terminal output at the bottom shows the execution results: 5 readers reading shared data (0) and 3 writers updating shared data to 10, 20, and 30 respectively. The program finishes with exit code 0.

```

62     writer_ids[i] = i + 1;
63     pthread_create(&writers[i], NULL, writer, &writer_ids[i]);
64 }
65
66 // Wait for all threads to finish
67 for (int i = 0; i < 5; i++) {
68     pthread_join(readers[i], NULL);
69 }
70 for (int i = 0; i < 3; i++) {
71     pthread_join(writers[i], NULL);
72 }
73
74 // Destroy semaphores
75 sem_destroy(&rw_mutex);
76 sem_destroy(&mutex);
77
78 return 0;
79 }
80

```

```

Reader 1: Reading shared data = 0
Reader 3: Reading shared data = 0
Reader 2: Reading shared data = 0
Reader 4: Reading shared data = 0
Reader 5: Reading shared data = 0
Writer 1: Updated shared data to 10
Writer 2: Updated shared data to 20
Writer 3: Updated shared data to 30

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

```

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.