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

**Aim:**

To simulate the **Reader-Writer problem** using **semaphores**. The problem involves multiple readers and writers accessing a shared resource. The solution uses semaphores to ensure that multiple readers can read the resource concurrently, but writers have exclusive access when writing, preventing race conditions.

**Procedure:**

1. Use two semaphores: mutex for controlling access to the read\_count, and write\_mutex for ensuring exclusive access for writers.
2. Readers increment and decrement a read\_count. If it's the first reader, it acquires write\_mutex, and if it's the last reader, it releases write\_mutex.
3. Writers acquire write\_mutex to gain exclusive access to the resource and release it when done.
4. Readers and writers run concurrently, with synchronization ensuring mutual exclusion where necessary.

**Code Implementation:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <unistd.h>

sem\_t mutex, write\_mutex;

int read\_count = 0;

void\* reader(void\* param) {

while(1) {

sem\_wait(&mutex);

read\_count++;

if(read\_count == 1) {

sem\_wait(&write\_mutex);

}

sem\_post(&mutex);

printf("Reader is reading the resource\n");

sleep(1);

sem\_wait(&mutex);

read\_count--;

if(read\_count == 0) {

sem\_post(&write\_mutex);

}

sem\_post(&mutex);

sleep(1);

}

}

void\* writer(void\* param) {

while(1) {

sem\_wait(&write\_mutex);

printf("Writer is writing to the resource\n");

sleep(2);

sem\_post(&write\_mutex);

sleep(1);

}

}

int main() {

pthread\_t reader\_threads[5], writer\_threads[2];

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

sem\_init(&write\_mutex, 0, 1);

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

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

}

for(int i = 0; i < 2; i++) {

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

}

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

pthread\_join(reader\_threads[i], NULL);

}

for(int i = 0; i < 2; i++) {

pthread\_join(writer\_threads[i], NULL);

}

sem\_destroy(&mutex);

sem\_destroy(&write\_mutex);

return 0;

}

Output:

