OS Assignment 4B - Reader-writer problem

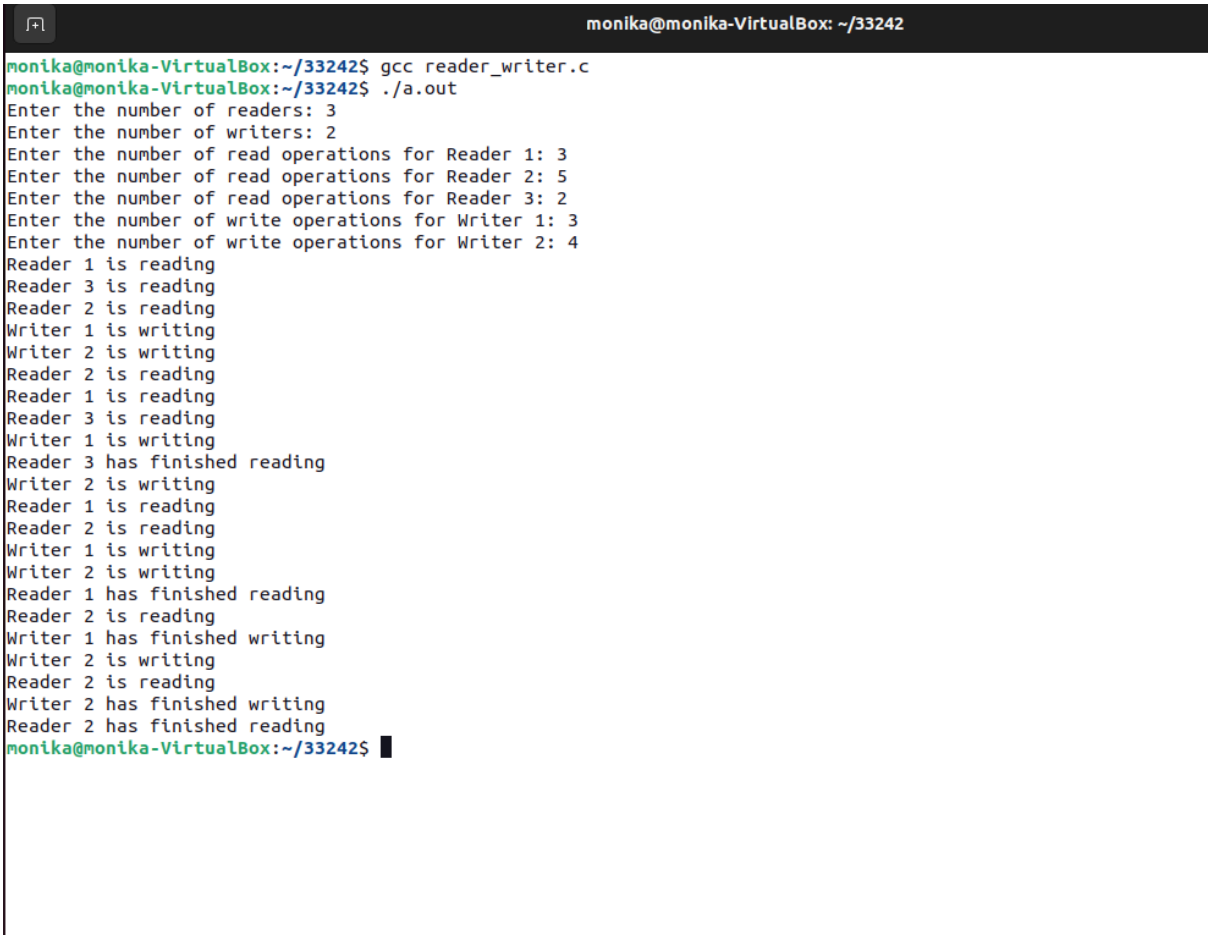
Code -

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

sem_t rw_mutex;  // Controls access to the shared resource
sem_t mutex;     // Controls access to the reader count
int read_count = 0;  // Number of active readers

void* reader(void* arg) {
    int reader_id = *((int*)arg);
    int iterations = *((int*)(arg + sizeof(int)));  // Get the number
of iterations from the passed arguments

    for (int i = 0; i < iterations; i++) {
        sem_wait(&mutex);  // Request access to modify read_count
        read_count++;
        if (read_count == 1) {
            sem_wait(&rw_mutex);  // First reader locks the shared
resource
        }
        sem_post(&mutex);  // Release access to read_count

        // Reading section
        printf("Reader %d is reading\n", reader_id);
        fflush(stdout);
        sleep(1);  // Simulate reading time

        sem_wait(&mutex);  // Request access to modify read_count
        read_count--;
        if (read_count == 0) {
            sem_post(&rw_mutex);  // Last reader unlocks the shared
resource
        }
        sem_post(&mutex);  // Release access to read_count

        sleep(1);  // Simulate time between reading attempts
    }
    printf("Reader %d has finished reading\n", reader_id);
    fflush(stdout);
    return NULL;
}

void* writer(void* arg) {
    int writer_id = *((int*)arg);
```

```
    int iterations = *((int*)(arg + sizeof(int)));  // Get the number
of iterations from the passed arguments

    for (int i = 0; i < iterations; i++) {
        sem_wait(&rw_mutex);  // Request exclusive access to the shared
resource

        // Writing section
        printf("Writer %d is writing\n", writer_id);
        fflush(stdout);
        sleep(1);  // Simulate writing time

        sem_post(&rw_mutex);  // Release exclusive access to the shared
resource

        sleep(2);  // Simulate time between writing attempts
    }
    printf("Writer %d has finished writing\n", writer_id);
    fflush(stdout);
    return NULL;
}

int main() {
    int num_readers, num_writers;

    // Ask the user for the number of readers and writers
    printf("Enter the number of readers: ");
    fflush(stdout);
    scanf("%d", &num_readers);
    printf("Enter the number of writers: ");
    fflush(stdout);
    scanf("%d", &num_writers);

    pthread_t readers[num_readers], writers[num_writers];
    int reader_ids[num_readers], writer_ids[num_writers];
    int read_iterations[num_readers], write_iterations[num_writers];

    // Collect reader and writer iterations
    for (int i = 0; i < num_readers; i++) {
        reader_ids[i] = i + 1;
        printf("Enter the number of read operations for Reader %d: ",
reader_ids[i]);
        fflush(stdout);
        scanf("%d", &read_iterations[i]);
    }
    for (int i = 0; i < num_writers; i++) {
        writer_ids[i] = i + 1;
        printf("Enter the number of write operations for Writer %d: ",
writer_ids[i]);
        fflush(stdout);
        scanf("%d", &write_iterations[i]);
```

```
    }

    // Initialize semaphores
    sem_init(&mutex, 0, 1);
    sem_init(&rw_mutex, 0, 1);

    // Create reader threads
    for (int i = 0; i < num_readers; i++) {
        // Pass both ID and iterations to the thread
        int* args = malloc(2 * sizeof(int));
        args[0] = reader_ids[i];
        args[1] = read_iterations[i];
        pthread_create(&readers[i], NULL, reader, args);
    }

    // Create writer threads
    for (int i = 0; i < num_writers; i++) {
        // Pass both ID and iterations to the thread
        int* args = malloc(2 * sizeof(int));
        args[0] = writer_ids[i];
        args[1] = write_iterations[i];
        pthread_create(&writers[i], NULL, writer, args);
    }

    // Wait for all threads to finish
    for (int i = 0; i < num_readers; i++) {
        pthread_join(readers[i], NULL);
    }
    for (int i = 0; i < num_writers; i++) {
        pthread_join(writers[i], NULL);
    }

    // Destroy the semaphores
    sem_destroy(&mutex);
    sem_destroy(&rw_mutex);

    return 0;
}
```

OUTPUT-


monika@monika-VirtualBox: ~/33242
monika@monika-VirtualBox:~/33242$ gcc reader_writer.c
monika@monika-VirtualBox:~/33242$ ./a.out
Enter the number of readers: 3
Enter the number of writers: 2
Enter the number of read operations for Reader 1: 3
Enter the number of read operations for Reader 2: 5
Enter the number of read operations for Reader 3: 2
Enter the number of write operations for Writer 1: 3
Enter the number of write operations for Writer 2: 4
Reader 1 is reading
Reader 3 is reading
Reader 2 is reading
Writer 1 is writing
Writer 2 is writing
Reader 2 is reading
Reader 1 is reading
Reader 3 is reading
Writer 1 is writing
Reader 3 has finished reading
Writer 2 is writing
Reader 1 is reading
Reader 2 is reading
Writer 1 is writing
Writer 2 is writing
Reader 1 has finished reading
Reader 2 is reading
Writer 1 has finished writing
Writer 2 is writing
Reader 2 is reading
Writer 2 has finished writing
Reader 2 has finished reading
monika@monika-VirtualBox:~/33242$