Lab Exercise – 13

AIM :: Implement `Reader-Writer` problem by using semaphores in Shell Scripting.

Theory ::

Reader-Writer Problem

1. **Definition**: The Reader-Writer problem is a classic synchronization problem that deals with the situation where multiple threads (readers and writers) need to access a shared resource, such as a database or file, without causing inconsistencies.
2. **Reader Preference**: In many implementations, readers can access the shared resource concurrently, as long as no writers are currently writing. If a writer is active, new readers must wait.
3. **Writer Exclusivity**: Writers need exclusive access to the resource, meaning that if a writer is writing, no other reader or writer can access the resource.
4. **Synchronization Mechanisms**: Semaphores are commonly used to synchronize access to shared resources. Two semaphores are typically implemented: one for managing read access and another for managing the count of active readers.
5. **Concurrency Control**: The implementation ensures that if a writer is writing, no readers can read, and when readers are reading, writers cannot write. This helps prevent race conditions and ensures data integrity.

Source_Code ::

```
echo "Amit Singhal - 11614802722 (5C6)"

gcc -o reader_writer reader_writer.c -lpthread -lrt

cat << EOF > reader_writer.c

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

#define MAX_READERS 5
#define MAX_WRITERS 3
```

```
// Semaphore for controlling access to the shared resource
                                sem_t rw_mutex;
// Semaphore for controlling access to the read_count variable
sem_t read_count_mutex;

int read_count = 0; // Counter for active readers

// Reader function
void* reader(void* id) {
    int reader_id = *((int*)id);
    while(1) {
        // Wait for access to read_count
        sem_wait(&read_count_mutex);
        read_count++; // Increment the number of readers
        if(read_count == 1) // If this is the first reader
            sem_wait(&rw_mutex); // Wait for the writer
        sem_post(&read_count_mutex); // Release access to read_count

        // Reading section
        printf("Reader %d is reading\n", reader_id);

        // Wait for access to read_count
        sem_wait(&read_count_mutex);
        read_count--; // Decrement the number of readers
        if(read_count == 0) // If this was the last reader
            sem_post(&rw_mutex); // Release the writer
        sem_post(&read_count_mutex); // Release access to read_count
    }
}

// Writer function
void* writer(void* id) {
    int writer_id = *((int*)id);
    while(1) {
        sem_wait(&rw_mutex); // Wait for exclusive access
        // Writing section
```

```
        printf("Writer %d is writing\n", writer_id);
        sleep(2); // Simulating write time
        sem_post(&rw_mutex); // Release exclusive access
        sleep(1); // Sleep before next write
    }
}

int main() {
    pthread_t read_threads[MAX_READERS], write_threads[MAX_WRITERS];
    int read_ids[MAX_READERS], write_ids[MAX_WRITERS];

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

    // Create reader threads
    for(int i = 0; i < MAX_READERS; i++) {
        read_ids[i] = i + 1;
        pthread_create(&read_threads[i], NULL, reader, (void*)&read_ids[i]);
    }

    // Create writer threads
    for(int i = 0; i < MAX_WRITERS; i++) {
        write_ids[i] = i + 1;
        pthread_create(&write_threads[i], NULL, writer, (void*)&write_ids[i]);
    }

    // Join reader threads
    for(int i = 0; i < MAX_READERS; i++)
        pthread_join(read_threads[i], NULL);

    // Join writer threads
    for(int i = 0; i < MAX_WRITERS; i++)
        pthread_join(write_threads[i], NULL);

    // Destroy semaphores
    sem_destroy(&rw_mutex);
```

```
    sem_destroy(&read_count_mutex);
    return 0;
}

EOF
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./amit.sh

Amit Singhal - 11614802722 (5C6)

Reader 1 is reading
Reader 2 is reading
Reader 3 is reading
Writer 1 is writing
Reader 4 is reading
Reader 5 is reading
Writer 2 is writing
Reader 1 is reading
Reader 2 is reading
Reader 3 is reading
Writer 3 is writing
Writer 1 is writing
Reader 4 is reading
Reader 5 is reading
Reader 1 is reading
Writer 2 is writing
Reader 2 is reading
Reader 3 is reading
Reader 4 is reading
Reader 5 is reading
Writer 1 is writing
Writer 3 is writing
Reader 1 is reading
Reader 2 is reading
```