Assignment No. 4B

Name: **Sahil Jagadale** Division: TE-11 Batch: L-11 Roll No.: **33327**

Title: Thread synchronization and mutual exclusion using mutex.

Aim: Application to demonstrate Reader-Writer problem with reader having priority

OBJECTIVE: Implement C program to demonstrate Reader-Writer problem with readers

having priority using counting semaphores and mutex.

THEORY:

Reader-Writer problem with readers priority the readers/writer's problem is defined as follows: There is a data area shared among several processes. The data area could be a file, a block of main memory, or even a bank of processor registers. There are several processes that only read the data area (readers) and a number that only write to the data area (writers).

The conditions that must be satisfied are as follows:

1. Any number of readers may simultaneously read the file.

2. Only one writer at a time may write to the file.

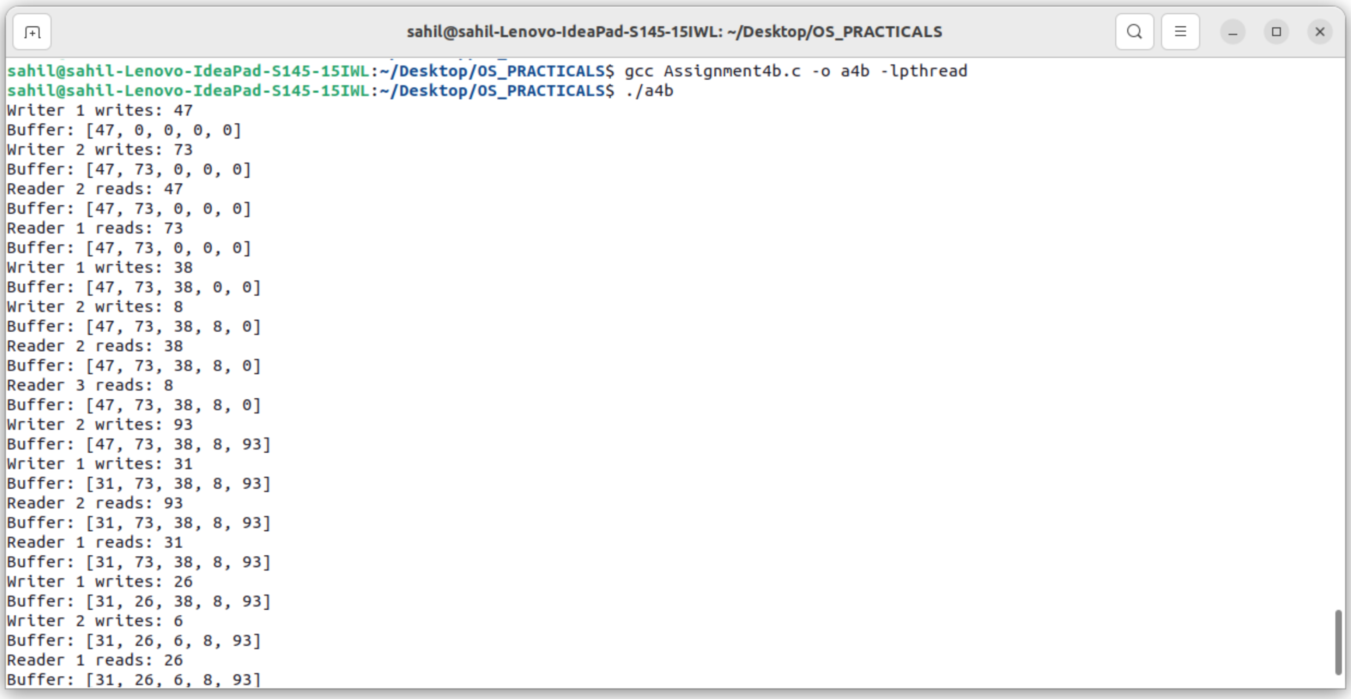
3. If a writer is writing to the file, no reader may read it.

Thus, readers are processes that are not required to exclude one another, and writers are processes that are required to exclude all other processes, readers, and writers alike. Before proceeding, let us distinguish this problem from two others: the general mutual exclusion problem and the producer/consumer problem. In the readers/writer's problem readers do not also write to the data area, nor do writers read the data area while writing.

Code:

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

#define BUFFER_SIZE 5
#define NUM_READERS 3
#define NUM_WRITERS 2

int buffer[BUFFER_SIZE];
sem_t empty, full;
pthread_mutex_t mutex;
int in = 0, out = 0;

void print_buffer() {
    printf("Buffer: [");
    for (int i = 0; i < BUFFER_SIZE; i++) {
        printf("%d", buffer[i]);
        if (i < BUFFER_SIZE - 1) {
            printf(", ");
        }
    }
    printf("]\n");
}

void *reader(void *arg) {
    int reader_id = *((int *)arg);
    int data;
    while (1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        data = buffer[out];
        printf("Reader %d reads: %d\n", reader_id, data);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        print_buffer();

        sleep(2);
    }
    pthread_exit(NULL);
}

void *writer(void *arg) {
    int writer_id = *((int *)arg);
    int data;
    while (1) {
        data = rand() % 100; // Generate a random integer

        sem_wait(&empty);
        pthread_mutex_lock(&mutex);

        buffer[in] = data;
        printf("Writer %d writes: %d\n", writer_id, data);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);

        print_buffer();

        sleep(5);
    }
    pthread_exit(NULL);
}

int main() {
    srand(time(NULL)); // Seed the random number generator
    pthread_t readers[NUM_READERS];
    pthread_t writers[NUM_WRITERS];
    int reader_ids[NUM_READERS];
    int writer_ids[NUM_WRITERS];

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    for (int i = 0; i < NUM_READERS; i++) {
        reader_ids[i] = i + 1;
        pthread_create(&readers[i], NULL, reader, &reader_ids[i]);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
        writer_ids[i] = i + 1;
        pthread_create(&writers[i], NULL, writer, &writer_ids[i]);
    }

    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);
    }

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



sahil@sahil-Lenovo-IdeaPad-S145-15IWL: ~/Desktop/OS_PRACTICALS

```
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ gcc Assignment4b.c -o a4b -lpthread
sahil@sahil-Lenovo-IdeaPad-S145-15IWL:~/Desktop/OS_PRACTICALS$ ./a4b
Writer 1 writes: 47
Buffer: [47, 0, 0, 0, 0]
Writer 2 writes: 73
Buffer: [47, 73, 0, 0, 0]
Reader 2 reads: 47
Buffer: [47, 73, 0, 0, 0]
Reader 1 reads: 73
Buffer: [47, 73, 0, 0, 0]
Writer 1 writes: 38
Buffer: [47, 73, 38, 0, 0]
Writer 2 writes: 8
Buffer: [47, 73, 38, 8, 0]
Reader 2 reads: 38
Buffer: [47, 73, 38, 8, 0]
Reader 3 reads: 8
Buffer: [47, 73, 38, 8, 0]
Writer 2 writes: 93
Buffer: [47, 73, 38, 8, 93]
Writer 1 writes: 31
Buffer: [31, 73, 38, 8, 93]
Reader 2 reads: 93
Buffer: [31, 73, 38, 8, 93]
Reader 1 reads: 31
Buffer: [31, 73, 38, 8, 93]
Writer 1 writes: 26
Buffer: [31, 26, 38, 8, 93]
Writer 2 writes: 6
Buffer: [31, 26, 6, 8, 93]
Reader 1 reads: 26
Buffer: [31, 26, 6, 8, 93]
```