Assignment 4 (B)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>  // For the sleep function on Unix-based systems

pthread_mutex_t mutex;          // Mutex to protect the read_count
pthread_mutex_t rw_mutex;       // Mutex to control access to the
shared resource
int read_count = 0;             // Count of readers currently reading
int shared_data = 0;            // Shared resource

// Function executed by each reader thread
void *reader(void *param) {
    int reader_id = *((int *)param);

    pthread_mutex_lock(&mutex);  // Lock mutex to modify read_count
    read_count++;
    if (read_count == 1)
        pthread_mutex_lock(&rw_mutex);  // First reader locks the
shared resource
    pthread_mutex_unlock(&mutex);  // Unlock mutex

    // Readers are now reading the shared resource
    printf("Reader %d started reading. Readers currently reading:
%d\n", reader_id, read_count);
    sleep(1);  // Simulate reading time with sleep for 1 second

    printf("Reader %d is still reading... (Shared Data = %d)\n",
reader_id, shared_data);  // Additional message to show concurrent
access
    sleep(1);  // Simulate reading time again to show overlapping
readers
```

```
    printf("Reader %d finished reading.\n", reader_id);

    pthread_mutex_lock(&mutex);  // Lock mutex to modify read_count
    read_count--;
    if (read_count == 0)
        pthread_mutex_unlock(&rw_mutex);  // Last reader unlocks the shared resource
    pthread_mutex_unlock(&mutex);  // Unlock mutex

    pthread_exit(0);
}

// Function executed by each writer thread
void *writer(void *param) {
    int writer_id = *((int *)param);

    pthread_mutex_lock(&rw_mutex);  // Writer locks the shared resource

    // Writer is now writing to the shared resource
    printf("Writer %d started writing.\n", writer_id);
    shared_data++;  // Increment the shared resource value
    sleep(1);  // Simulate writing time
    printf("Writer %d finished writing. Shared Data = %d\n", writer_id, shared_data);

    pthread_mutex_unlock(&rw_mutex);  // Writer unlocks the shared resource

    pthread_exit(0);
}

int main() {
    pthread_t r_tid[5], w_tid[5];
```

```
    int reader_ids[5], writer_ids[5];

    // Initialize the mutexes
    pthread_mutex_init(&mutex, NULL);
    pthread_mutex_init(&rw_mutex, NULL);

    // Create reader threads
    for (int i = 0; i < 5; i++) {
        reader_ids[i] = i + 1;
        pthread_create(&r_tid[i], NULL, reader, &reader_ids[i]);
    }

    // Create writer threads
    for (int i = 0; i < 5; i++) {
        writer_ids[i] = i + 1;
        pthread_create(&w_tid[i], NULL, writer, &writer_ids[i]);
    }

    // Wait for all reader and writer threads to complete
    for (int i = 0; i < 5; i++) {
        pthread_join(r_tid[i], NULL);
        pthread_join(w_tid[i], NULL);
    }

    // Destroy the mutexes
    pthread_mutex_destroy(&mutex);
    pthread_mutex_destroy(&rw_mutex);

    return 0;
}
```

```
Output :
Reader 1 started reading. Readers currently reading: 1
Reader 2 started reading. Readers currently reading: 2
Reader 3 started reading. Readers currently reading: 3
Reader 4 started reading. Readers currently reading: 4
Reader 5 started reading. Readers currently reading: 5
Reader 1 is still reading... (Shared Data = 0)
Reader 3 is still reading... (Shared Data = 0)
Reader 2 is still reading... (Shared Data = 0)
Reader 4 is still reading... (Shared Data = 0)
Reader 5 is still reading... (Shared Data = 0)
Reader 1 finished reading.
Reader 2 finished reading.
Reader 5 finished reading.
Reader 3 finished reading.
Reader 4 finished reading.
Writer 1 started writing.
Writer 1 finished writing. Shared Data = 1
Writer 2 started writing.
Writer 2 finished writing. Shared Data = 2
Writer 3 started writing.
Writer 3 finished writing. Shared Data = 3
Writer 4 started writing.
Writer 4 finished writing. Shared Data = 4
Writer 5 started writing.
Writer 5 finished writing. Shared Data = 5
```