Experiment No. 6

NAME: Omkar Khanvilkar ROLL NO: 07

CLASS: TY_IT-B BATCH: 2

DATE OF PERFORMANCE: 26/09/2024 PRN No: 12210313

---

1) Implementation of Classical problems Reader Writer using Threads and Mutex

Code in c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

pthread_mutex_t mutex, write_mutex;
int read_count = 0;
int shared_data = 0;

void* reader(void* arg) {
    int reader_id = *((int*)arg);
    pthread_mutex_lock(&mutex);
    read_count++;
    if (read_count == 1) {
        pthread_mutex_lock(&write_mutex);
    }
    pthread_mutex_unlock(&mutex);

    // Reading the shared data
    printf("Reader %d: reads shared data as %d\n", reader_id, shared_data);

    pthread_mutex_lock(&mutex);
    read_count--;
    if (read_count == 0) {
        pthread_mutex_unlock(&write_mutex); .
    }
    pthread_mutex_unlock(&mutex);

    return NULL;
}

void* writer(void* arg) {
    int writer_id = *((int*)arg);
    pthread_mutex_lock(&write_mutex);

    // Writing to the shared data
    shared_data += 10;
```

```c
    printf("Writer %d: writes shared data as %d\n", writer_id, shared_data);

    pthread_mutex_unlock(&write_mutex);
    return NULL;
}

int main() {
    int num_readers, num_writers;

    printf("Enter the number of readers: ");
    scanf("%d", &num_readers);
    printf("Enter the number of writers: ");
    scanf("%d", &num_writers);

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

    pthread_mutex_init(&mutex, NULL);
    pthread_mutex_init(&write_mutex, NULL);

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

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

    // Joining reader threads
    for (int i = 0; i < num_readers; i++) {
        pthread_join(readers[i], NULL);
    }

    // Joining writer threads
    for (int i = 0; i < num_writers; i++) {
        pthread_join(writers[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    pthread_mutex_destroy(&write_mutex);

    return 0;
}
```

Output :

```
Enter the number of readers: 3
Enter the number of writers: 2
Reader 1: reads shared data as 0
Writer 2: writes shared data as 10
Reader 3: reads shared data as 10
Reader 2: reads shared data as 10
Writer 1: writes shared data as 20


...Program finished with exit code 0
Press ENTER to exit console.
```

2) Implementation of Classical problems Reader Writer using Threads and Samaphore

Code in C :

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

sem_t rw_mutex, mutex;
int read_count = 0;
int shared_data = 0;

void* reader(void* arg) {
    int reader_id = *((int*)arg);

    sem_wait(&mutex);
    read_count++;
    if (read_count == 1) {
        sem_wait(&rw_mutex); // First reader locks the writer.
    }
    sem_post(&mutex);

    // Reading the shared data
    printf("Reader %d: reads shared data as %d\n", reader_id, shared_data);

    sem_wait(&mutex);
    read_count--;
    if (read_count == 0) {
        sem_post(&rw_mutex); // Last reader unlocks the writer.
    }
```

```c
    sem_post(&mutex);

    return NULL;
}

void* writer(void* arg) {
    int writer_id = *((int*)arg);
    sem_wait(&rw_mutex);

    // Writing to the shared data
    shared_data += 10;
    printf("Writer %d: writes shared data as %d\n", writer_id, shared_data);

    sem_post(&rw_mutex);
    return NULL;
}

int main() {
    int num_readers, num_writers;

    printf("Enter the number of readers: ");
    scanf("%d", &num_readers);
    printf("Enter the number of writers: ");
    scanf("%d", &num_writers);

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

    sem_init(&rw_mutex, 0, 1);
    sem_init(&mutex, 0, 1);

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

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

    // Joining reader threads
    for (int i = 0; i < num_readers; i++) {
        pthread_join(readers[i], NULL);
    }
```

```
    // Joining writer threads
    for (int i = 0; i < num_writers; i++) {
        pthread_join(writers[i], NULL);
    }

    sem_destroy(&rw_mutex);
    sem_destroy(&mutex);

    return 0;
}
```

Output :

```
Enter the number of readers: 3
Enter the number of writers: 2
Reader 1: reads shared data as 0
Reader 2: reads shared data as 0
Reader 3: reads shared data as 0
Writer 1: writes shared data as 10
Writer 2: writes shared data as 20


...Program finished with exit code 0
Press ENTER to exit console.
```