## Assignment 4B

Name: Kumar Deshmukh

Roll: 33143

## Problem Statement:

Implement C program to demonstrate Reader-Writer problem with readers having priority using counting semaphores and mutex

## Theory:

### Semaphores:

An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic: initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a counting semaphore or a general semaphore.

Semaphores are the OS tools for synchronization. Two types:

1. Binary Semaphore.
2. Counting Semaphore.

### Counting semaphore

The counting semaphores are free of the limitations of the binary semaphores. A counting semaphore comprises:

An integer variable, initialized to a value K (K>=0). During operation it can

assume any value <= K, a pointer to a process queue. The queue will hold the PCBs of all those

processes, waiting to enter their critical sections. The queue is implemented as a

FCFS, so that the waiting processes are served in a FCFS order.

### Operation of a counting semaphore:

1. Let the initial value of the semaphore count be 1.

2. When semaphore count = 1, it implies that no process is executing in its critical section and no process is waiting in the semaphore queue.

3. When semaphore count = 0, it implies that one process is executing in its critical section but no process is waiting in the semaphore queue.

4. When semaphore count = N, it implies that one process is executing in its critical section and N processes are waiting in the semaphore queue.

5. When a process is waiting in semaphore queue, it is not performing any busy waiting. It is rather in a "waiting" or "blocked" state.

6. When a waiting process is selected for entry into its critical section, it is transferred from "Blocked" state to "ready" state.

### *Reader-Writer problem with readers priority*

The readers/writer's problem is defined as follows: There is a data area shared among a number of processes. The data area could be a file, a block of main memory, or even a bank of processor registers. There are a number of 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.

A more general case, which includes this case, is to allow any of the processes to read or write the data area. In that case, we can declare any portion of a process that accesses the data area to be a critical section and impose the general mutual exclusion solution. The reason for being concerned with the more restricted case is that more efficient solutions are possible for this case and that the less efficient solutions to the general problem are unacceptably slow. For example, suppose that the shared area is a library catalog. Ordinary users of the library read the catalog to locate a book. One or more librarians are able to update the catalog.

In the general solution, every access to the catalog would be treated as a critical section, and users would be forced to read the catalog one at a time. This would clearly impose intolerable delays. At the same time, it is important to prevent writers from interfering with each other and it is also required to prevent reading while writing is in progress to prevent the access of inconsistent information.

This is not a special case of producer-consumer. The producer is not just a writer.

It must read queue pointers to determine where to write the next item, and it must determine if the buffer is full. Similarly, the consumer is not just a reader, because it must adjust the queue pointers to show that it has removed a unit from the buffer.

## **Conclusion:**

We implemented reader-writer problem with reader-priority using counting semaphores and mutex

**CODE:**

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
using namespace std;

pthread_mutex_t mutex;
pthread_mutex_t wrt;

int read_count = 0;

void *Reader(void *args)
{
    pthread_mutex_lock(&mutex);
    read_count++;
    if (read_count == 1)
    {
        pthread_mutex_lock(&wrt);
        cout << "Reader is Reading" << endl;
    }

    pthread_mutex_unlock(&mutex);
    cout << "Reader count = " << read_count << endl;
    sleep(10);
    pthread_mutex_lock(&mutex);
    read_count--;
    if (read_count == 0)
    {
        cout << "Reader is Leaving" << endl;
        pthread_mutex_unlock(&wrt);
    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
void *Writer(void *args)
{
    pthread_mutex_lock(&wrt);
    cout << "Writer is Writing " << endl;
    cout << "Writer is Leaving " << endl;
    pthread_mutex_unlock(&wrt);
    return NULL;
}
int main()
{
    int reader_thread = 0, writer_thread = 0;
    pthread_mutex_init(&mutex, NULL);
    pthread_mutex_init(&wrt, NULL);
    pthread_t WriterThreads[10], ReaderThreads[10];
    int i = 0;
    int ch = 0;
    do
    {
        cout << "\n1. Reader\n2. Writer\n3. Exit" << endl;
        cout<<"Choice: ";
        cin >> ch;
        switch (ch)
        {
        case 1:
            pthread_create(&ReaderThreads[i], NULL, Reader, NULL);
            i++;
            sleep(1);
```

```
            break;
        case 2:
            pthread_create(&WriterThreads[i], NULL, Writer, NULL);
            i++;
            sleep(1);
            break;
        case 3:
            exit(1);
            break;
        }
    } while (ch != 3 && i != 10);
    for (int i = 0; i < reader_thread; i++)
        (void)pthread_join(ReaderThreads[i], NULL);
    for (int i = 0; i < writer_thread; i++)
        (void)pthread_join(WriterThreads[i], NULL);
    pthread_mutex_destroy(&wrt);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

## OUTPUT:

```
(base) kumar@pop-os:~/Desktop/OS/Assignments$ g++ -pthread Assig4b.cpp
(base) kumar@pop-os:~/Desktop/OS/Assignments$ ./a.out

1. Reader
2. Writer
3. Exit
Choice: 1
Reader is Reading
Reader count = 1

1. Reader
2. Writer
3. Exit
Choice: 1
Reader count = 2

1. Reader
2. Writer
3. Exit
Choice: 1
Reader count = 3

1. Reader
2. Writer
3. Exit
Choice: 2

1. Reader
2. Writer
3. Exit
Choice: 2

1. Reader
2. Writer
3. Exit
Choice: 1
Reader count = 2

1. Reader
2. Writer
3. Exit
Choice: 2

1. Reader
2. Writer
3. Exit
Choice: 3
(base) kumar@pop-os:~/Desktop/OS/Assignments$
```