REPORT OF ASSIGNMENT 3

What is Readers-Writers Problem: Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database. We distinguish between these two types of processes by referring to the former as readers and to the latter as writers. Obviously, if two readers access the shared data simultaneously, no adverse effects will result. However, if a writer and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue.
To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to the database. This synchronization problem is referred to as the readers–writers problem.

Design and Analysis of Readers-Writers(Normal): C++11 Semaphore.h and Thread libraries are used for maintaing threads and semaphores. Following is the pseudocode for Readers and Writers.

```
Initaialisations:   sem_t rw mutex = 1;
                    sem_t mutex = 1;
                    int read count = 0;
Writer Process:     do {
                            wait(rw mutex);
                            . . .
                            /* writing is performed */
                            . . .
                            signal(rw mutex);
                    }
Reader Process:     do {
                            wait(mutex);
                            read count++;
                            if (read count == 1)
                                    wait(rw mutex);
                            signal(mutex);
                            . . .
                            /* reading is performed */
                            . . .
                            wait(mutex);
                            read count--;
                            if (read count == 0)
                            signal(rw mutex);
                            signal(mutex);
                    } while (true);
```

Above mentioned code implements first readers-writers problem. Here no reader is kept waiting unless a writer has already obtained permission to use the shared object.

In other words, no reader should wait for other readers to finish simply because a writer is waiting.
The semaphores mutex and rw mutex are initialized to 1; read count is initialized to 0. The semaphore rw mutex is common to both reader and writer processes. The mutex semaphore is used to ensure mutual exclusion when the variable read count is updated. The read count variable keeps track of how many processes are currently reading the object. The semaphore rw mutex functions as a mutual exclusion semaphore for the writers. It is also used by the first or last reader that enters or exits the critical section. It is not used by readers who enter or exit while other readers are in their critical sections.
Note that, if a writer is in the critical section and n readers are waiting, then one reader is queued on rw mutex , and n − 1 readers are queued on mutex . Also observe that, when a writer executes signal(rw mutex) , we may resume the execution of either the waiting readers or a single waiting writer. The selection is made by the scheduler.

Design and Analysis of Fair Solution to Readers-Writers Problem: C++11 Semaphore.h and Thread libraries are used for maintaing threads and semaphores.

Why a Fair Solution to Readers-Writers Problem is requiered?

The readers–writers problem has several variations, all involving priorities. The simplest one, referred to as the first readers–writers problem, requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting. The second readers –writers problem requires that, once a writer is ready, that writer perform its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading.
A solution to either problem may result in starvation. In the first case, writers may starve; in the second case, readers may starve.
Therefore, a fair Solution to Readers-Writers Problem is required to avoid starvation.

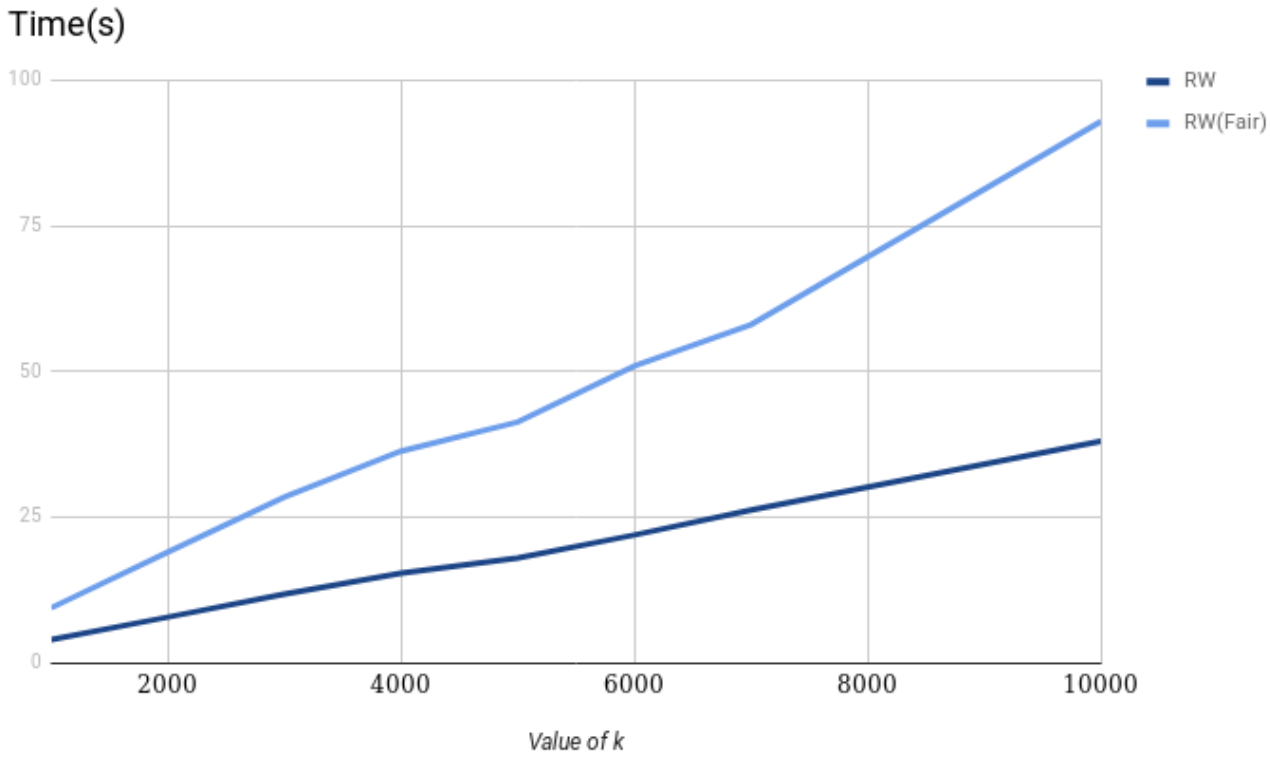
Following is the pseudocode for Readers and Writers.

```
Initialisations:        in = Semaphore(1)
                        out = Semaphore(1)
                        wrt = Semaphore(0)
                        ctrin = Integer(0)
                        ctrout = Integer(0)
                        wait = Boolean(0)

Readers:                Wait in
                        ctrin++
                        Signal in
```

```
                [Critical section]
                Wait out
                ctrout++
                if (wait==1 && ctrin==ctrout)
                then Signal wrt
                Signal out

Writers:        Wait in
                Wait out
                if (ctrin==ctrout)
                then Signal out
                else
                        wait=1
                        Signal out
                        Wait wrt
                        wait=0
                [Critical section]
                Signal in
```

The main idea here is that a Writer indicates to Readers its necessity to access the working area. At the same time no new Readers can start working. Every Reader leaving the working area checks if there is a Writer waiting and the last leaving Reader signals Writer that it is safe to proceed now. Upon completing access to the working area Writer signals waiting Readers that it finished allowing them to access the working area again.
The solution requires only one mutex locking for a Reader both entering the critical section and exiting the critical section; and writer needs two mutex lockings for Writer to enter critical section.

Comparison of Average Time Taken per thread in both solutions:



Observations from Graph: 1) Execution time for [RW(Fair) > RW]
The observation agrees with the theoretical values. As RW(Fair) is a more complex program therefore it's time should be higer.

2) The graph for both the curves is nearly a straight line.
For large values of k there is deviation from straight line.
The deviation is more for RW(Fair). The Fair nature of solution causes this deviation.
For higher number of threads the fairness of distribution increases the time.