Starve free readers and Writers problem: Some threads may read and some may write, with the constraint that no thread may access the shared resource for either reading or writing while another thread is in the act of writing to it.A readers–writer lock is a data structure that solves one or more of the readers–writers problems**.**There are three versions of the problem.

First-Readers Writers problem: Suppose we have a shared memory area (critical section) with the basic constraints detailed above. It is possible to protect the shared data behind a mutual exclusion mutex, in which case no two threads can access the data at the same time. However, this solution is sub-optimal, because it is possible that a reader R1 might have the lock, and then another reader R2 requests access. It would be foolish for R2 to wait until R1 was done before starting its own read operation; instead, R2 should be allowed to read the resource alongside R1 because reads don't modify data, so concurrent reads are safe. This is the motivation for the first readers–writers problem, in which the constraint is added that no reader shall be kept waiting if the share is currently opened for reading. This is also called readers-preference.

In this solution, every writer must claim the resource individually. This means that a stream of readers can subsequently lock all potential writers out and starve them.

This is so, because after the first reader locks the resource, no writer can lock it, before it gets released. And it will only be released by the last reader. Hence, this solution does not satisfy fairness.

<u>Second-Readers Writers problem:</u> The first solution is suboptimal, because it is possible that a reader R1 might have the lock, a writer W be waiting for the lock, and then a reader R2 requests access. It would be unfair for R2 to jump in immediately, ahead of W; if that happened often enough, W would starve. Instead, W should start as soon as possible. This is the motivation for the second readers–writers problem, in which the constraint is added that no writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is also called writers-preference.

If there are no writers wishing to get to the resource, as indicated to the reader by the status of the readtry semaphore, then the readers will not lock the resource. This is done to allow a writer to immediately take control over the resource as soon as the current reader is finished reading. Otherwise, the writer would need to wait for a queue of readers to be done before the last one can unlock the readtry semaphore. As soon as a writer shows up, it will try to set the readtry and hang up there waiting for the current reader to release the readtry. It will then take control over the resource as soon as the current reader is done reading and lock all future readers out. All subsequent readers will hang up at the readtry semaphore waiting for the writers to be finished with the resource and to open the gate by releasing readtry.

Third-Reader Writers problem: n fact, the solutions implied by both problem statements can result in starvation — the first one may starve writers in the queue, and the second one may starve readers. Therefore, the third readers–writers problem is sometimes proposed, which adds the constraint that no thread shall be allowed to starve; that is, the operation of obtaining a lock on the shared data will always terminate in a bounded amount of time. A solution with fairness for both readers and writers.

This solution can solve both the problems that arised during the above two versions of the solution.So this is optimal solution.