Operating Systems 2 – CS3523
Programming Assignment 5 – Report

Suraj Telugu – CS20BTECH11050

Design and Implementation:

Reader Writer Problem Solution (Reader Preference):

- All global variables (inputs and outputs) are defined. Rthread_id and wthread_id are defined as atomic_int so that it does not get raced between threads. rw_mutex and sem_mutex are defined as semaphores and initialized to 0.
- Reader threads and Writer threads are created using pthread_create, Reader() and Writer() functions they later joined using pthread_join operation.

In Reader() function

- The RW problem solution is implemented using two semaphores every reader goes through the entry section; only one reader is allowed in the entry section at a time to avoid race conditions.
- Then after going through the entry section it will unlock the sem_mutex by signaling (sem_post). Any number of readers can enter the critical section at the same time but only a single reader can enter the exit section.

In Writer() function

- It is implemented using rw_mutex semaphore which allows only one thread to enter the critical section at a time and all the other writer threads have to wait until the writer thread exits.
- If a reader thread is waiting for a writer thread to signal the rw_mutex, all the other reader threads are waiting on mutex. After the writer thread signals the mutex, all the reader threads can simultaneously perform the read operations. Till all the readers are done, all the writer threads are paused on rw_mutex, thus, causing starvation.

Reader Writer Problem Solution (Fair Solution):

- All global variables (inputs and outputs) are defined. Rthread_id and wthread_id are defined as atomic_int so that it does not get raced between threads. Rw_mutex, sem_mutex and in_mutex are defined as semaphores and initialized to 0 in main.
- Reader threads and Writer threads are created using pthread_create, Reader() and Writer() functions they later joined using pthread_join operation.

In Reader() function

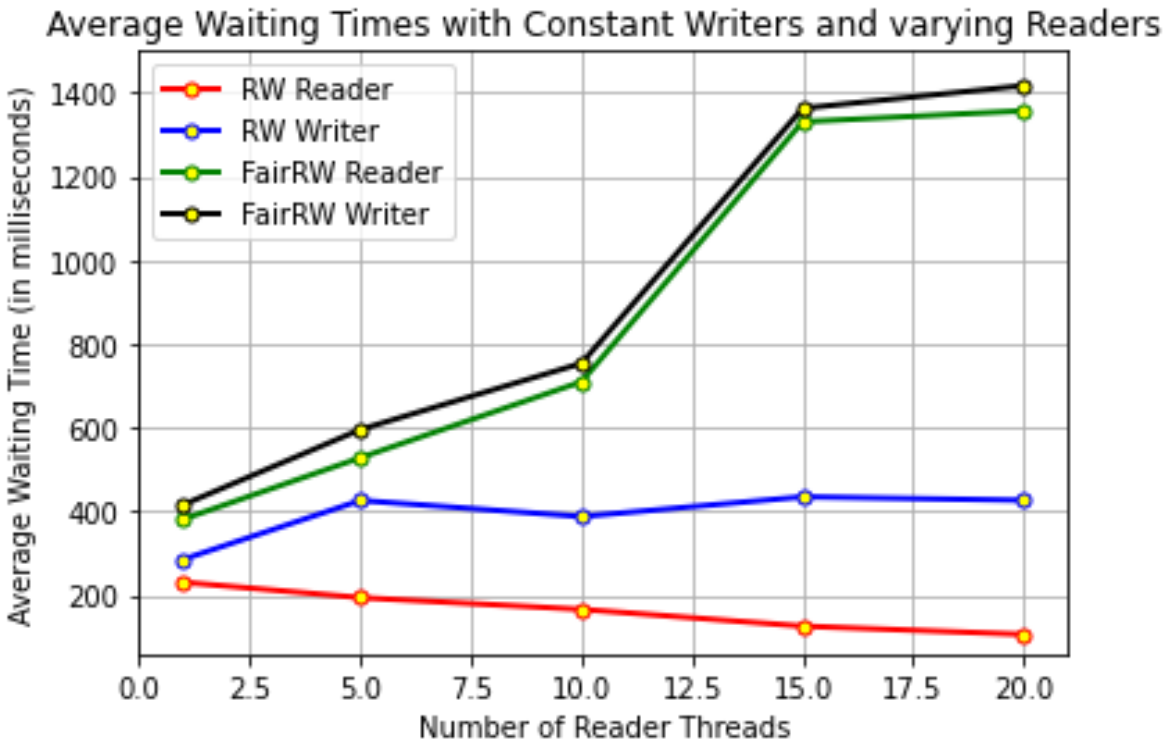
- The RW problem solution is implemented using three semaphores every reader thread that goes through the entry section first tries to acquire in_mutex.
- The thread which acquires the in_mutex can only enter the entry section of the reader thread.It releases the in_mutex after the reader thread goes through the exit section. Thus every reader thread has to acquire in_mutex to execute CS.

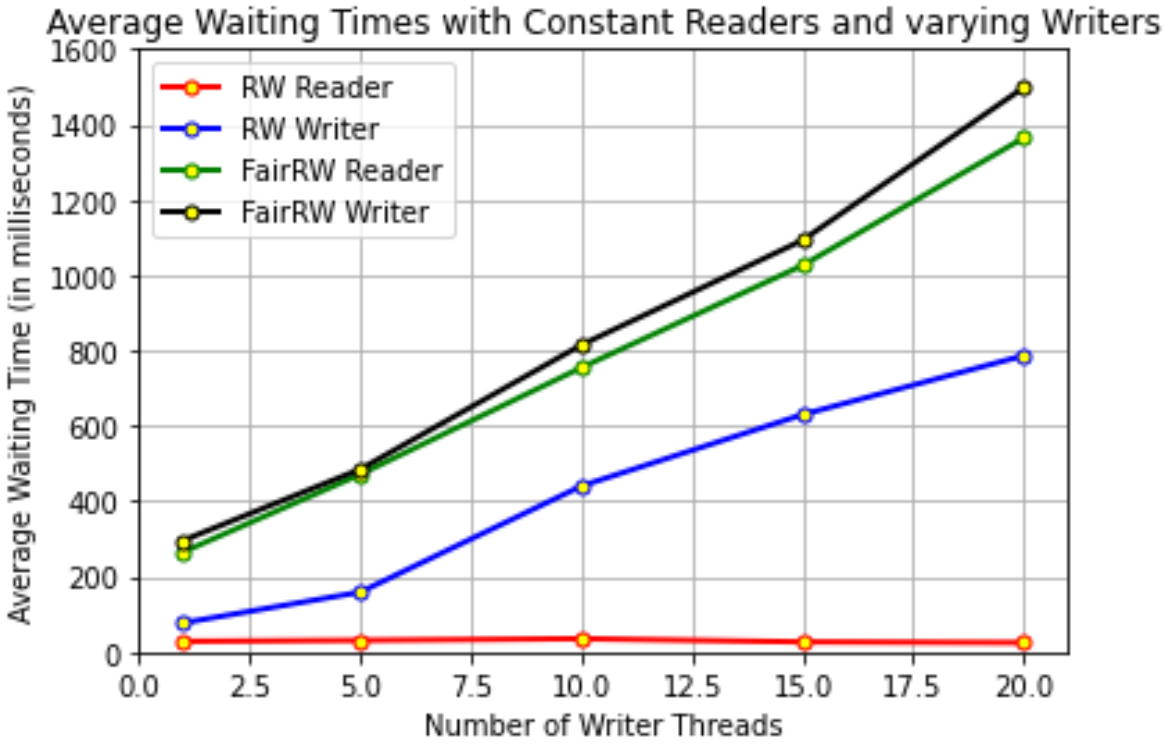
In Writer() function

- It is implemented using rw_mutex and in_mutex semaphores. The in_mutex lock ensures that the writer cannot starve and compete with the reader thread for execution.
- Finally only one thread can enter the critical section at a time and all the other writer threads have to wait until the writer thread exits.
- If a reader thread is waiting for a writer thread to signal the rw_mutex, all the other reader threads are waiting on mutex. This implementation ensures that the reader thread and writer thread do not starve and implement alternatively.

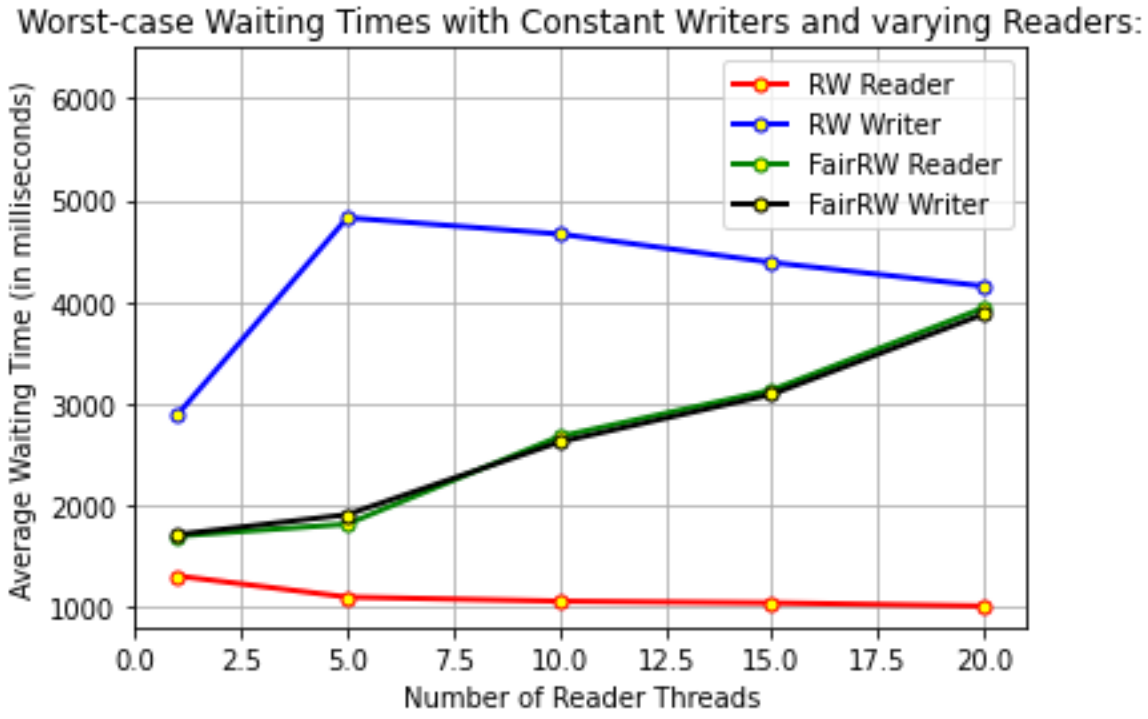
Graphs

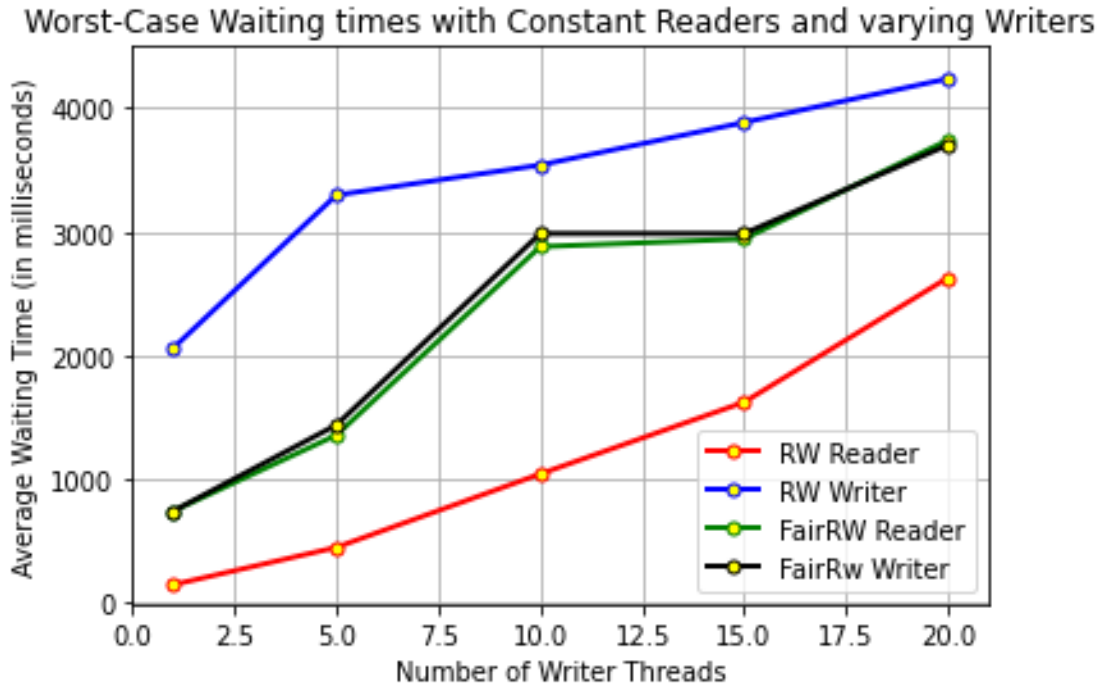
➔ Average Waiting Times vs Varying (Readers or Writers) Threads





➔ Worst - Case Waiting Times vs Varying (Readers or Writers) Threads





Analysis and Conclusion

- The average waiting time for each thread of Fair solution is significantly larger than average waiting times Reader preference solution. The prevention of starvation has resulted in an increase of average waiting times for Fair solution.

- The average waiting time of each reader thread is always less than the average waiting time of each writer thread for Reader preference solution. Since writers starve.

- The Worst waiting Time for Fair solution is nearly the same for both readers and writers this shows there is no starvation.

- Though starvation is prevented through fair solutions, both reader and writer threads starve to some extent, therefore the worst case waiting times fair solution is less than the readers' preference solution.