Assignment-5 Report
OS2 – CS3523

Name: *Taha Adeel Mohammed*
Roll Number: *CS20BTECH11052*

Code Design

- Except for the mutual exclusion algorithms implementation, the reader preference solution and the fair solution codes share a similar design.
- We have declared many global variables to store the simulation parameters(nw, nr, kw, kr, mu_cs, mu_rem), the input and output files, and simulation statistics required in order to run our simulation.
- We define a class **Time** to allow us to easily convert the current time to a string, and to find the difference between two time points using operator overloading.
- The function ***getSysTime()*** is defined to get the current time with the help of standard functions from the <ctime> and <chrono> libraries.
- We also define the ***sleep()*** function, which causes the invoking thread to sleep for a certain floating point time in milliseconds..

- We read the input from the file in the main function, and then create nw writer threads which run the ***writer()*** function and nr reader threads which run the ***reader()*** function , with the thread number as a parameter.
- In the reader() and writer() functions, we first generate randCSTime and randRemTime using an exponential distribution with average mu_cs and mu_rem using functions/classes from the <random> library.
- Then we run our simulation, i.e. run the reader-writers with the critical section kr and kw times respectively.
- In each iteration of the reader/writers processes, we log the events of when the process requests to enter the critical section, when it enters the critical section, and when it leaves the critical section into the log file.
- The reader/writer processes are made to sleep for mu_cs milliseconds to simulate the critical section and then for mu_rem milliseconds to simulate the remaining section of the process.

- ❖ We also measure the time between a request and entry into the critical section of the process to determine the waiting times.
- ❖ Finally, we ensure solve the reader-writer problem using the following solutions:-

Reader preference

We have a semaphore rw_mutex to ensure readers don't read while a writer is updating the data. We also have a global int called read_counter to keep track of the number of active readers. We have another semaphore defined to ensure that we update the counter atomically only.

- **Writer thread:-** Before entering the CS, we wait to acquire the rw_mutex. Then we release the rw_mutex after the CS is over. Hence only one writer executes at once.
- **Reader thread:-** Before entering the CS for the reader thread, we increment the reader counter(atomically by surrounding it inside a mutex), and if it is our first reader, then we wait to acquire the rw_mutex. We release the rw_mutex after the CS is over only if it was the last reader.

Hence, we solve the reader-writer problem with reader preference.

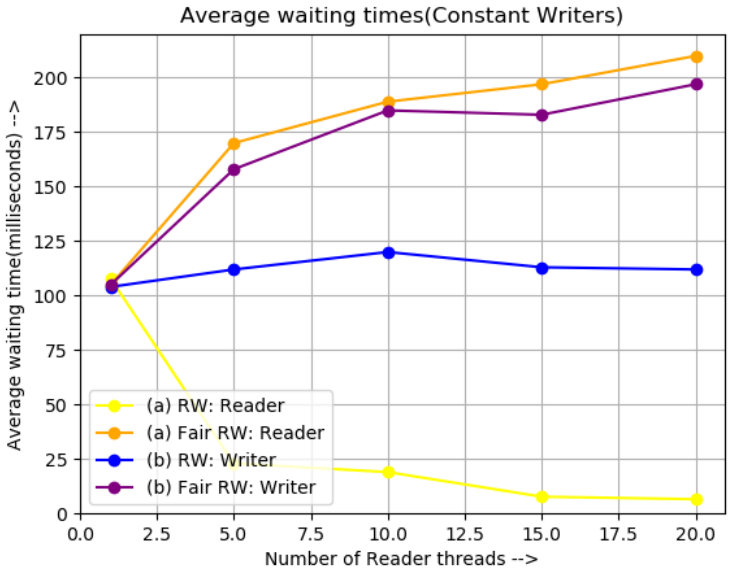
Fair solution

Our fair solution to the reader-writer problem is very similar to the above solution, except we have a semaphore “queue” (which is assumed to be FIFO), to ensure that no threads have to undergo starvation.

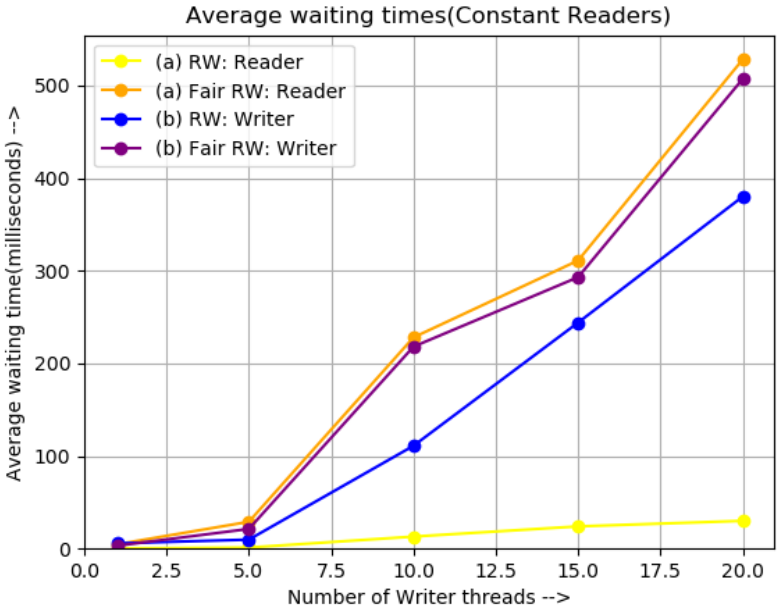
- **Writer thread:-** Similar to above implementation, except that before trying to acquire the rw_mutex, we first try to acquire the queue semaphore (which we release after acquiring the rw_mutex).
- **Reader thread:-** Here too, we first try to acquire the queue semaphore before entering the critical section. The remaining solution is similar to the above solution. As the queue semaphore is provided on FIFO basis, we ensure that starvation doesn't happen.

Result Analysis

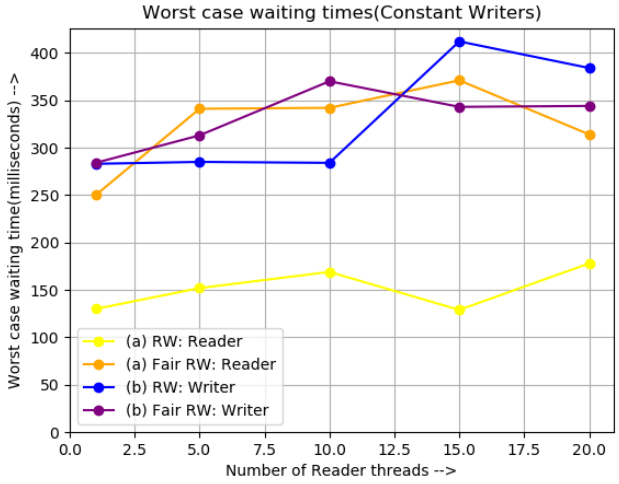
❖ 1. Average Waiting Times with Constant Writers



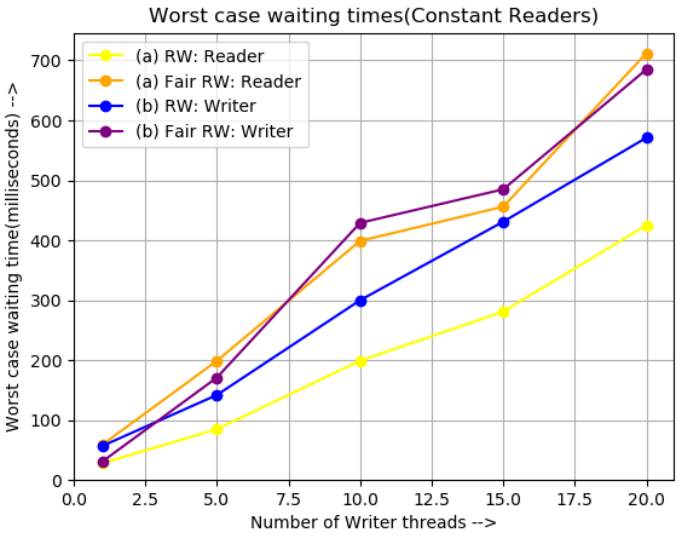
❖ 2.Average Waiting Times with Constant Readers



❖ 3. Worst-case Waiting Times with Constant Writers



❖ 4. Worst-case Waiting Times with Constant Readers



These graphs are generated using input parameters mu_cs and mu_rem as 200 and 300 milliseconds respectively (and kw = kr = 10). We see that the average waiting time is significantly higher in the fair solution, as the fairness comes at the cost of increased waiting time of reader threads who were not waiting normally. We also see that the waiting time for normal reader threads is very low, as they are given preference over writers. We see that worst case waiting time is higher for fair readers and writers due to ensuring fairness, which causes them to wait longer.