Report

I. Design of the code

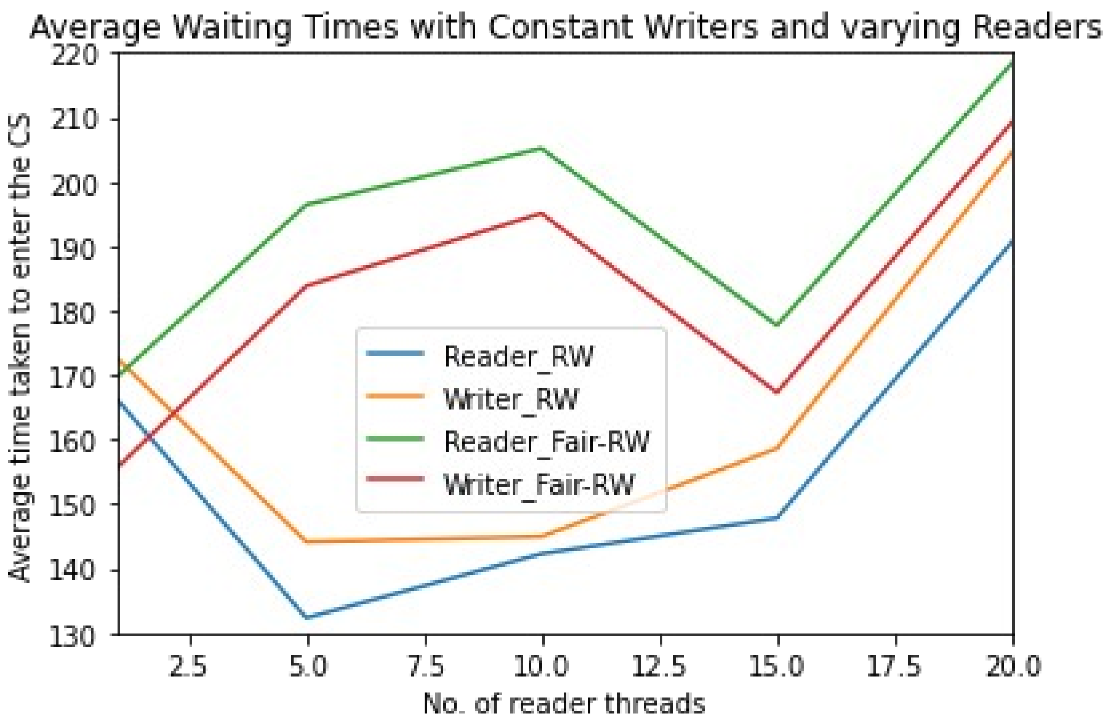
- We have used the functionality of semaphores by using semaphore.h header file. To define semaphore, we have used *sem_t name_of_sem.* To initialize semaphore, we have used sem_init(sem_t * name_of_sem, int pshared, int value) function which initializes name_of_sem to value. To destroy semaphore, we have used sem_destroy(sem_t * name_of_sem). This will destroy name_of_sem. We have initialized all semaphores to 1.
- We use rw_mutex to controls access to (read/write) resource. We use mutex for syncing changes to shared variable read_count. The read_count is initialzed to 0, which indicates the number of readers currently reading or accessing the resource.
- To calculate average waiting time and worst case waiting time, we use variables of double namely, totalWaitingTime_reader, totalWaitingTime_writer, worstCaseWaitingTime_reader, worstCaseWaitingTime_writer, all initialzed to 0.
- To write the logistics of process, we make use of output file logfile and for statistics, we make use of statfile.
- The function getCurrentTimeStamp function returns the current time in hr:min:sec:millisec format. We also have functions for reader and writer seperately.
- We start the main function by taking input from input file inp-params.txt. Then we create reader and writer threads and start the process after initializing all the semaphores to 1. Then we wait for all threads to join, after which we will have our logfile ready with logistics of the process and we can proceed to write the statistics to statfile. Also we destroy the semaphores after all threads finish their work.
- default_random_engine is used to generate random numbers from exponential_distribution to simulate critical and remainder sections. Parameters for both critical and remainder section were taken 1/mu_cs and 1/mu_rem respectively.

- We have used usleep function to make threads sleep for time given by exponential distribution in microseconds. It is available in unistd.h header file.
- We have used inbuilt exponential distribution functionality in the code. It is available in random header file.
- To get time when thread requested, entered and exited critical section, we have used functionality available in chrono header file.
- We have used an additional semaphore in fair solution, by the name serviceQueue. It preserves ordering of requests and ensures fairness.
- The reader and writer function take thread_id as their input and continue with the solution proposed.
- We are printing the average waiting time and worst case waiting time in milliseconds with a precision set to 2 places after decimal point.

II. Graphs

■ Average Waiting Times with Constant Writers and varying Readers

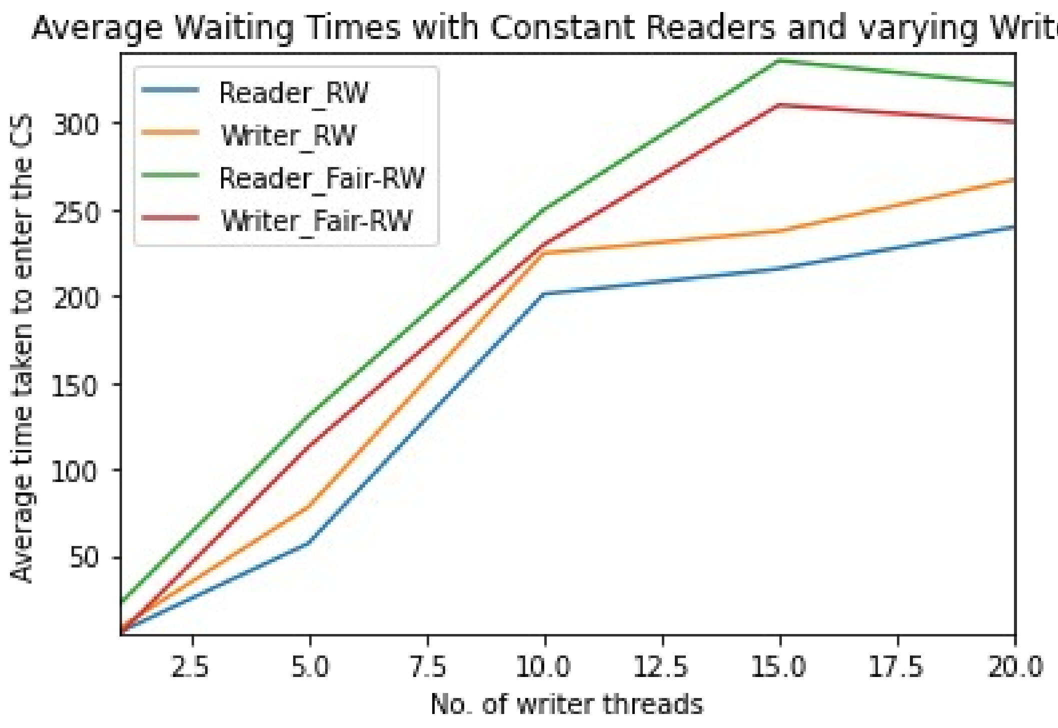
Parameters taken: *nw* = 10, *kw* = 10, *kr* = 10



Here we observe that with number of writers fixed, average waiting time for both readers and writers increases in the case of Fair-RW compared to RW. This is because of fairness that is introduced. Also we observe that average waiting time of readers increases drastically in case of Fair-RW when compared to RW. Though there is increase in average waiting time of writers from RW to Fair-RW, but it is less compared to that of readers.

- Average Waiting Times with Constant Readers and varying Writers

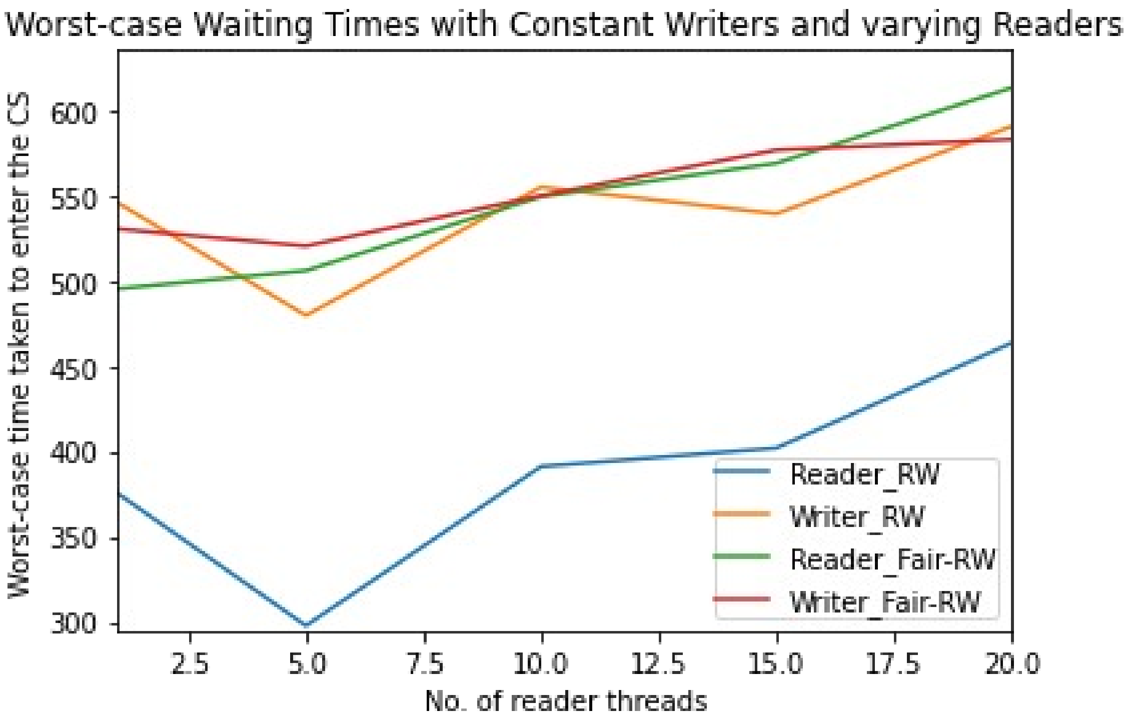
Parameters taken: *nr* = 10, *kw* = 10, *kr* = 10



Here we observe that with number of readers fixed, the average waiting time for both readers and writers increases in the case of Fair-RW compared to RW. This is because of fairness that is introduced. But there is less drastic change in average waiting time for both readers and writer compared to the previous graph when number of writers were fixed.

- <u>Worst-case Waiting Times with Constant Writers and varying Readers</u>

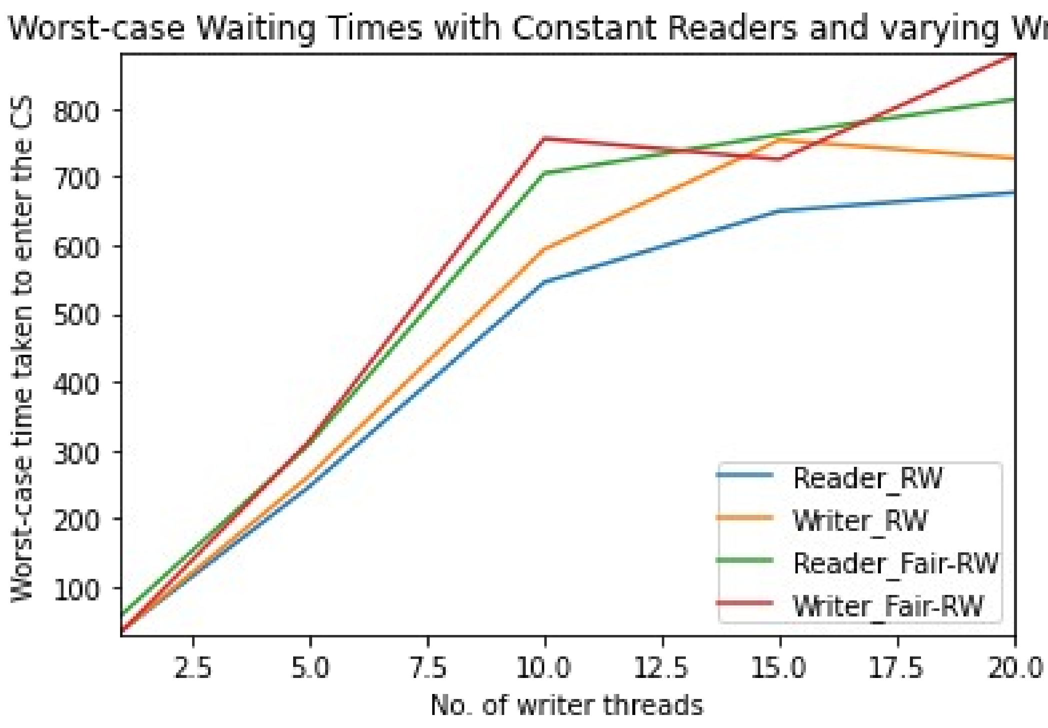
  Parameters taken: *nw* = 10, *kw* = 10, *kr* = 10



Here with number of writers fixed, worst case waiting time for readers increases drastically from RW to Fair-RW. Also there is increase in worst case waiting time for writers from RW to Fair-RW but it is very minimal. The reason behind drastic change of worst case waiting time for readers is that it's preference was lowered and made equal to that of writers in Fair-RW from RW.

- <u>Worst-case Waiting Times with Constant Readers and varying Writers</u>

  Parameters taken: *nr* = 10, *kw* = 10, *kr* = 10



Here we observe that with number of readers fixed, there is increase in worst case waiting time fo both readers and writers from RW to Fair-RW but the increase observed is not so large when compared to the previous graph where the number of writers is fixed.

■ <u>Conclusion</u>

Average waiting time as well as worst case waiting time for both readers and writers increases as we move from RW to Fair-RW. This is bacause of fairness that is introduced. In other words, the effect of lowering the preference of reader and making it equal to that of writer in Fair-RW. Also drastic change in average waiting time and worst case waiting time is observed when number of writers are fixed with varying number of readers. This is because with increase in number of readers, the opportunity that writer gets to access the resource decreases in RW.