REPORT
CS20BTECH11028

Design:

Normal Readers Writers (Higher priority to Readers):

We could use two semaphores mutex, rw_mutex both initialized to 1. The rw_mutex semaphore is used to maintain Mutual Exlusion between a writter and readers or other writers. The mutex semaphore is used to maintain Mutual Exlusion between readers for incrementing the readers count etc.

At the start of the program, the input is taken from the input file regarding the number of reader threads, writer threads, number of requests to CS etc. Then it creates the given number of threads and allocates the writer function to the writer threads and reader function to reader threads.

Writers:
In the writers, the threads start executing and wait for the rw_mutex. If available, they enter the CS and do the necessary writing (simulated using exponentially distributed random variables and usleep function) and after exitting the CS, it signals the rw_mutex. If not available, the writer thread is put to block on invoking wait and waits till it is signalled. After the exit from CS, the threads execute the remainder section which is simulated by sleeping for random amount of time which is given by the exponential distribution of a random variable μ_{rem}.

Readers:
In the readers, the threads start executing, each thread waits for the mutex, if it is available, the threads increment the read count and if read count is 1, the thread waits for the rw_mutex, if available, it signals the mutx, so that another reader thread could enter. Theere can be multiple readers at a time in the CS. After the completion of reading, each thread again waits on the mutex, and decreases the read count as it is exiting the CS, and if reader_count == 0 then it signals the rw_mutex, and after decrementing the reader_count it signals the mutex. It then enters the remainder section which is simulated by sleeping for random amount of time which is given by the exponential distribution of a random variable μ_{rem}

Because of this design, the readers can keep on entering the CS even though a writer is waiting from a long time on the rw_mutex, which can lead to starvation of writers.

Fair Readers Writers:

The design is almost same as above. Here in addition to the normal readers writers, we use another semaphore queue_mutex, initialized to 1. This is used in preserving the order in which Readers and Writers have made a request for CS.

Writers:
Before waiting for the rw_mutex, the writer threads waits for the queue_mutex, which helps in preserving the order in which the threads are requesting. Once the queue_mutex is available, the writer thread waits on rw_mutex and after that signals the queue_mutex.

Readers:
Here before waiting on mutex and incrementing thread count, the thread is made to wait on the queue_mutex. Once it is available, the reader waits for the mutex, and after incrementing the count and acquiring the rw_mutex (if reader_count==1) then queue_mutex is signalled.

Because of this design, every thread is executed in the order they request, so if a writer is already waiting on the queue_mutex, no new readers can enter the CS. This helps in preserving fairness.
This design is a modification of the above discussed design.

Note:

- As multiple threads requests at a time for the CS, if we output the logs as given by the sample pcode in the problem statement, the lines may not be printed in good format, hence i printed them in thread safe manner, even though the times are not effected but the order in which the statements print is affected.

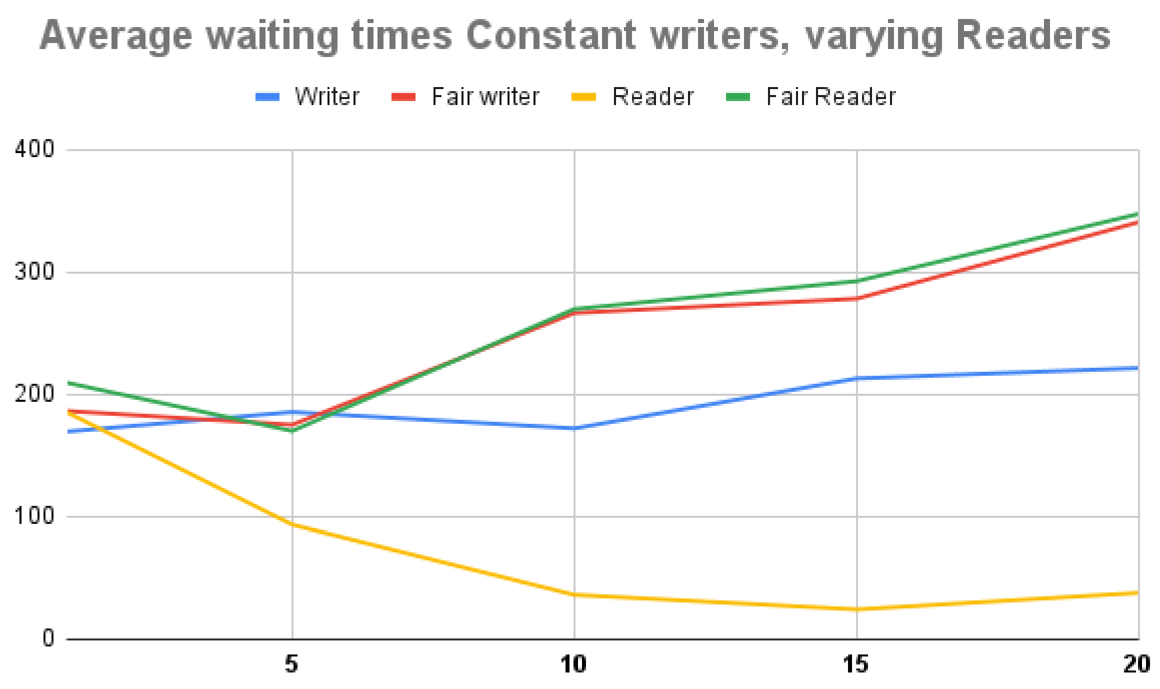
Calculation of waiting times:
In the critical sections and remainder sections we simulate the execution times using exponentially distributed random varaibles μ_{cs} and μ_{rem}. We calculate the wait time using the difference of entry time and request time, for each entry to CS of a thread. This calculation is done inside the CS to ensure thread safety while incementing the wait time. The program prints the current system time using gettimeofday(struct timeval*),ctime(time_t*) functions.

Comparison using Graphs:

For all the graphs:

$$\mathbf{k_w = 10}$$
$$\mathbf{k_r = 10}$$
$$\mathbf{\mu_{cs} = 20\ milliseconds}$$
$$\mathbf{\mu_{rem} = 30\ milliseconds}$$

A) Average waiting times with constant writers and varying readers



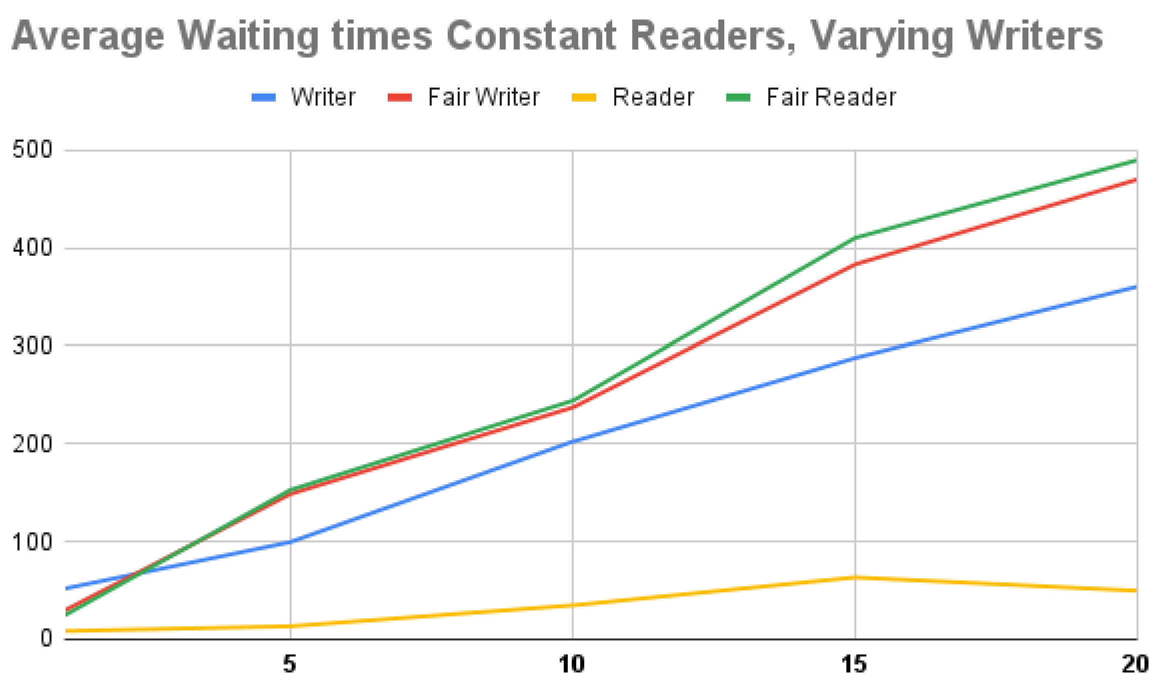
X-axis – Number of Reader threads
Y-axis – Average Waiting times (in ms)
Number of Writers = 10

Analysis:
Here, when we keep on increasing readers,

- The average waiting time of readers reduced initially and maintained almost same.
- The average waiting time of writers has shown a little bit of increment.
- The average waiting time in case of fair readers and fair writers kept on increasing, and both of them have almost same average waiting times.

The average waiting times for the fair case readers and writers are almost equal and higher when compared with the normal reader and writer threads.
The average waiting time of writer was higher when compared with the readers in the normal case.
The average waiting times of readers was lowest among all.

B) Average Waiting times with Constant Readers and Varying Writers

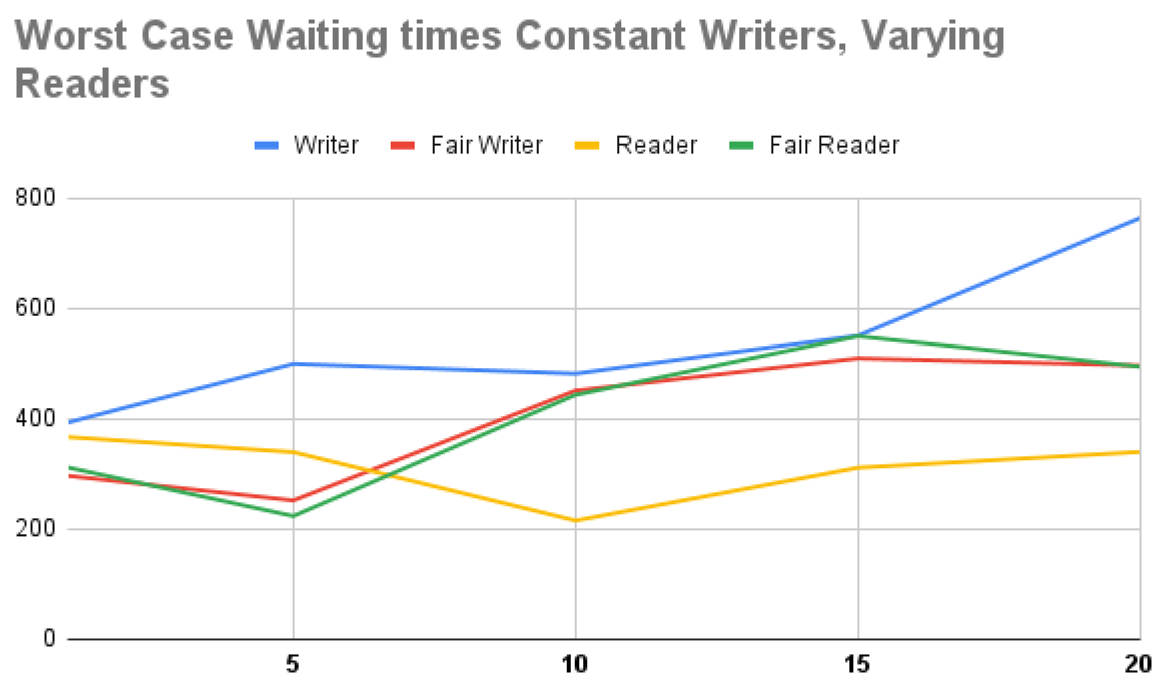


X-axis – Number of Writers threads
Y-axis – Average Waiting times (in ms)
Number of Readers = 10
Analysis:
Here as we kept on increasing writers,

- The aveage waiting times of readers doesn't have a much impact
- The average waiting times of writers have shown a lot of increment.
- The average waiting times of Fair readers and Fair writers also had shown increment, and both of their average waiting times are almost equal and higher when compared to the waiting times of normal readers and writers.

The average waiting times of normal case writers is higher when compared to normal case readers.
The average waiting times of normal case readers is the least among all.

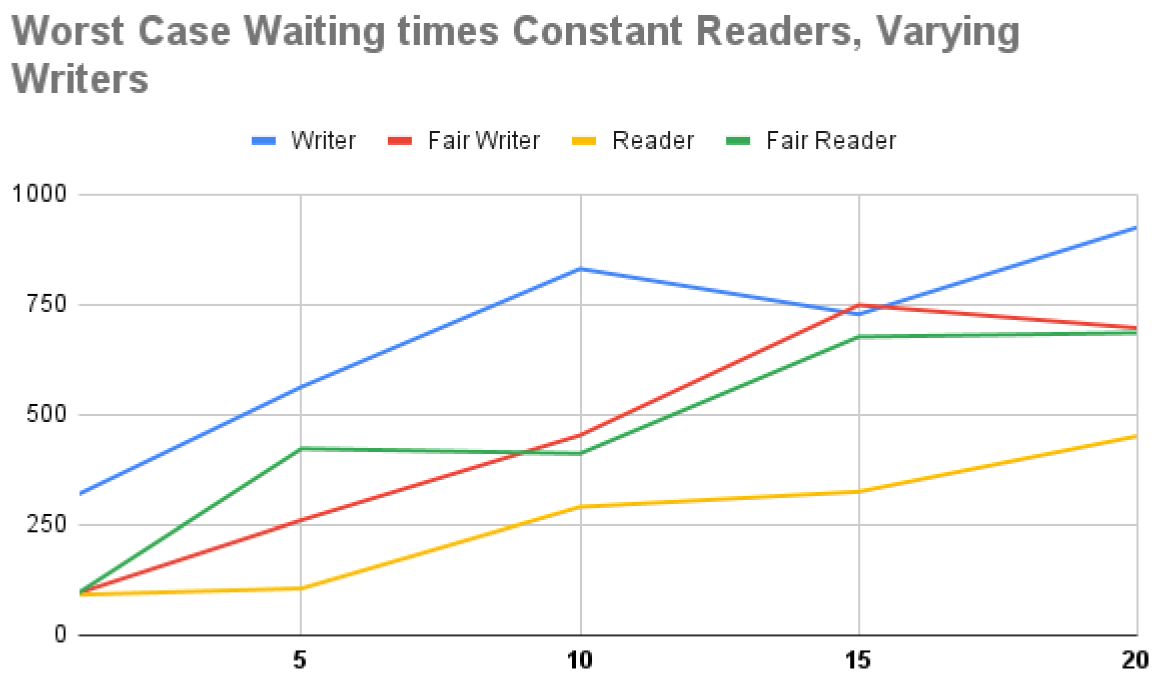
C) Worst Case Waiting times with Constant Writers, Varying Readers



X-axis – Number of Reader threads
Y-axis – Worst case Waiting times (in ms)
Number of Writers = 10
Analysis:

- The writer threads had the highest worst case waiting time among all.
- The reader threads had the lowest worst case waiting time among all.
- The fair case reader and writer threads had almost same worst case waiting time which was in between the worst case waitings times of normal case reader and writers.

D) Worst Case Waiting times with Constant Readers, Varying Writers



X-axis – Number of Writer threads
Y-axis – Worst Case Waiting times (in ms)
Number of Readers = 10
Analysis:

- The writer threads had the highest worst case waiting times among all
- The reader threads had the lowest worst case waiting times among all
- The fair case reader and writer threads had almost the same worst case waiting times which is in between the readers and writers

Hence in the graphs **A and B** we can see that,

In the case of normal reader-writers, as we give priority to readers, their wait time is smaller where as that of writer threads is higher. In case of fair reader writers, both the threads will have almost the same waiting times and as the order is being maintained using semaphore queue and each process waits for queue_mutex and hence their average wait times are higher when compared with the normal case.

From the graphs **C and D**,

We can see that the worst case waiting times of normal writers are much higher. This is because in the normal RW implementation the writers would get starved and readers are given priority. Hence the readers worst case waiting time is very less. In the fair RW case, there is no such starvation, hence both the values are almost same.