# Operating Systems-2
# Assignment 4
# Report

<u>Related Terms:</u>

- Entry Section: A segment of the code which is common to all the threads and where threads request & wait to enter the critical section.

- Critical Section(CS): A segment of the code where race condition can happen. So, only one thread should enter this section at a time.

- Exit Section: A segment of the code where threads exit the critical section and another thread is allowed to enter the critical section.

- Remainder Section: The remaining segment of the code which is not critical i.e. where race condition will not happen.

- Waiting Time: It is the time that a threads waits in the entry section to get into the critical section.

## Design & Implementation of code:

The design of the two programs are almost the same except for the fact that FairRW consists of an extra semaphore.

- At the start of the program, main() reads the input from “inp-params.txt” and store the values which are needed in all the threads globally.
- Then, we create two arrays of threads, one for writers and the other for readers.
- We have a variable named read_count which keeps track of the no. of readers that are currently in the CS.
- To generate random numbers, we use default_random_engine.
- To implement the exponential distribution of time, we use exponential_distribution<> in library <random>.
- We pass the readers and writers function to their respective threads and all of them run simultaneously.
- Reader threads measures the time at which they request for the allocation of CS, they

enter the CS and the time at which they exit from the CS by using functions from library <chrono> and prints that to a log file.

- Consumer threads also do the same for each request, entery and exit & writes it to a log file.
- To get the waiting time of each thread, we just subtract the timestamp at which it enters the CS and the timestamp at which it requests for the CS.
- To simulate the actions of writing and reading in the CS, we sleep the thread. To implement it, we used usleep() which takes parameters in microseconds.
- To make implementation of semaphore atomic, we used functions from "<semaphore.h>".

<u>Graphs:</u>

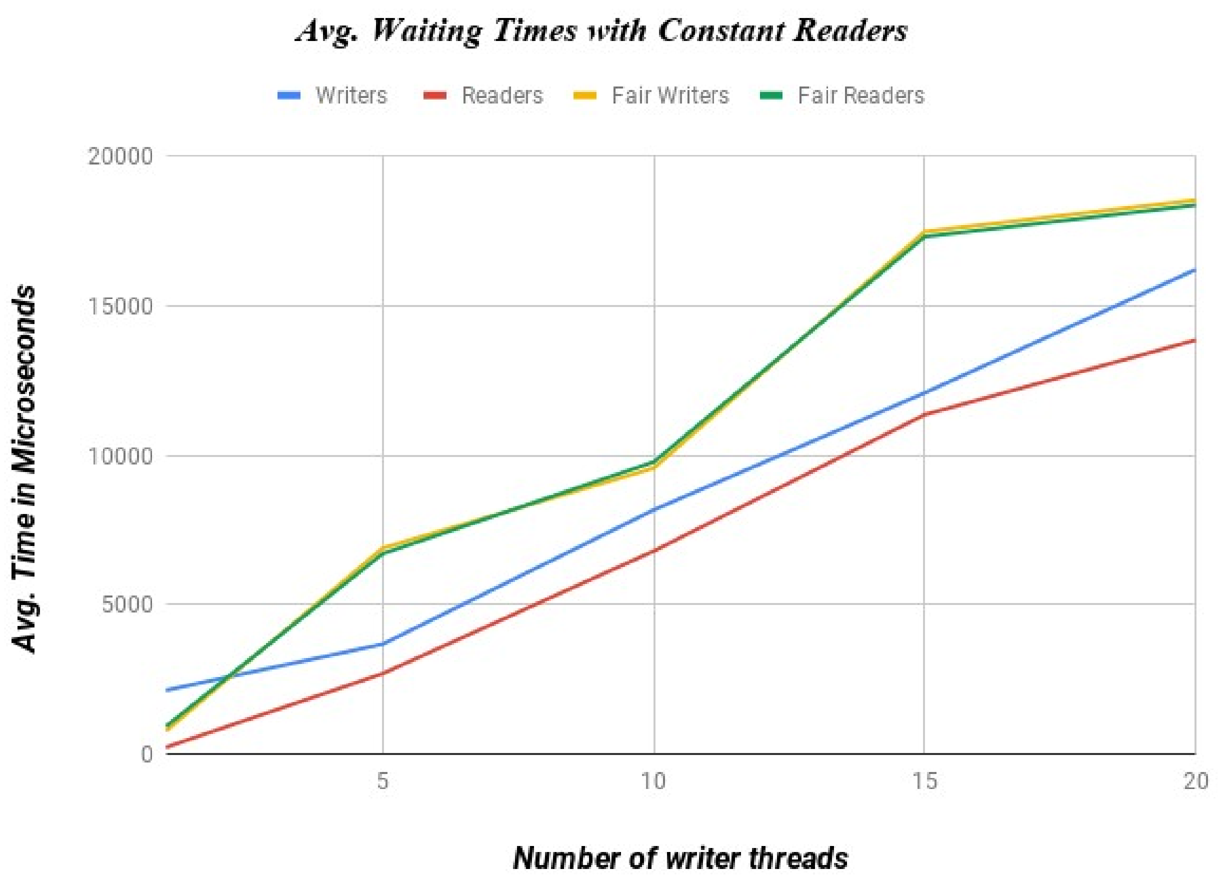
Graphs are plotted for

(kw, kr, μCS, μRem) = (10, 10, 1000, 1000).

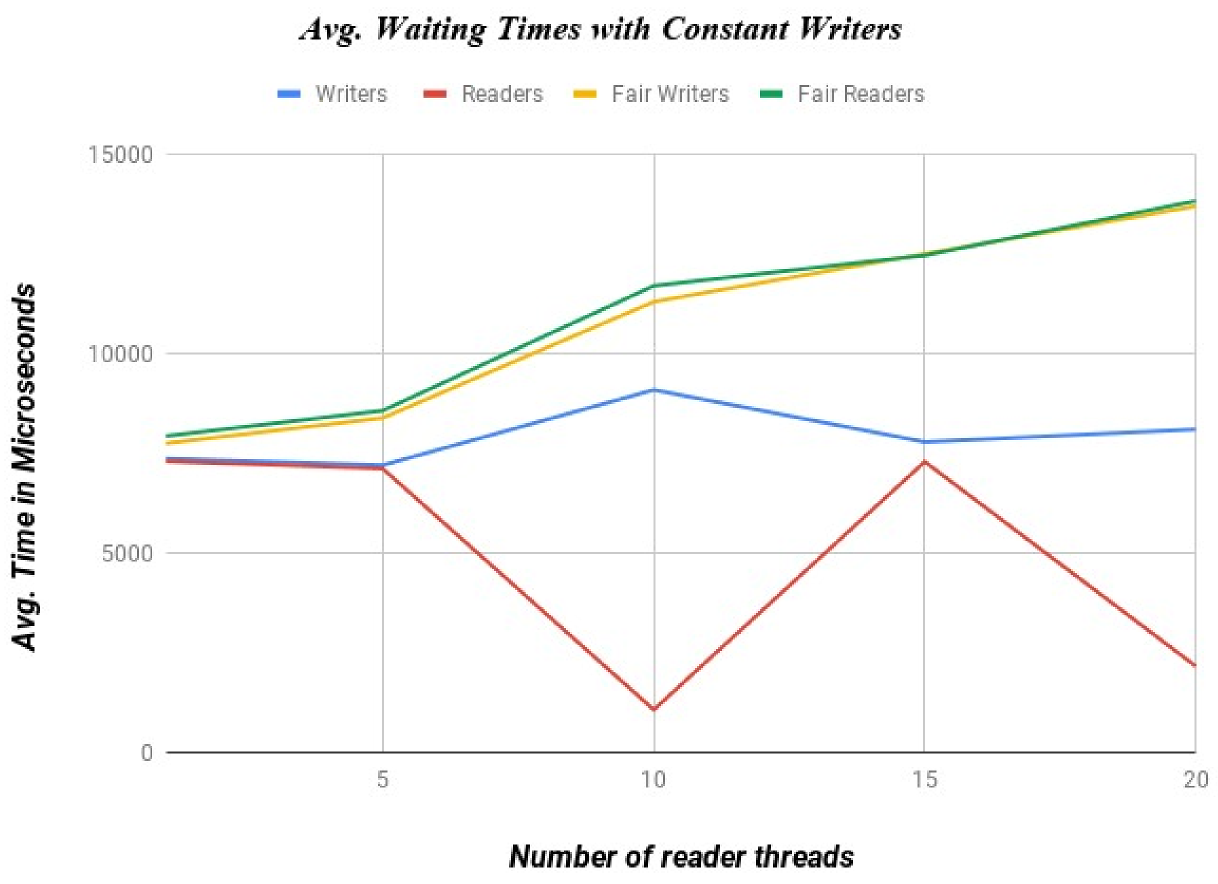
The values of np and nr are varied from 1 to 20 and graphs for the average waiting time and worst

waiting time are plotted for readers and writers threads.
For all the graphs time is in microseconds.

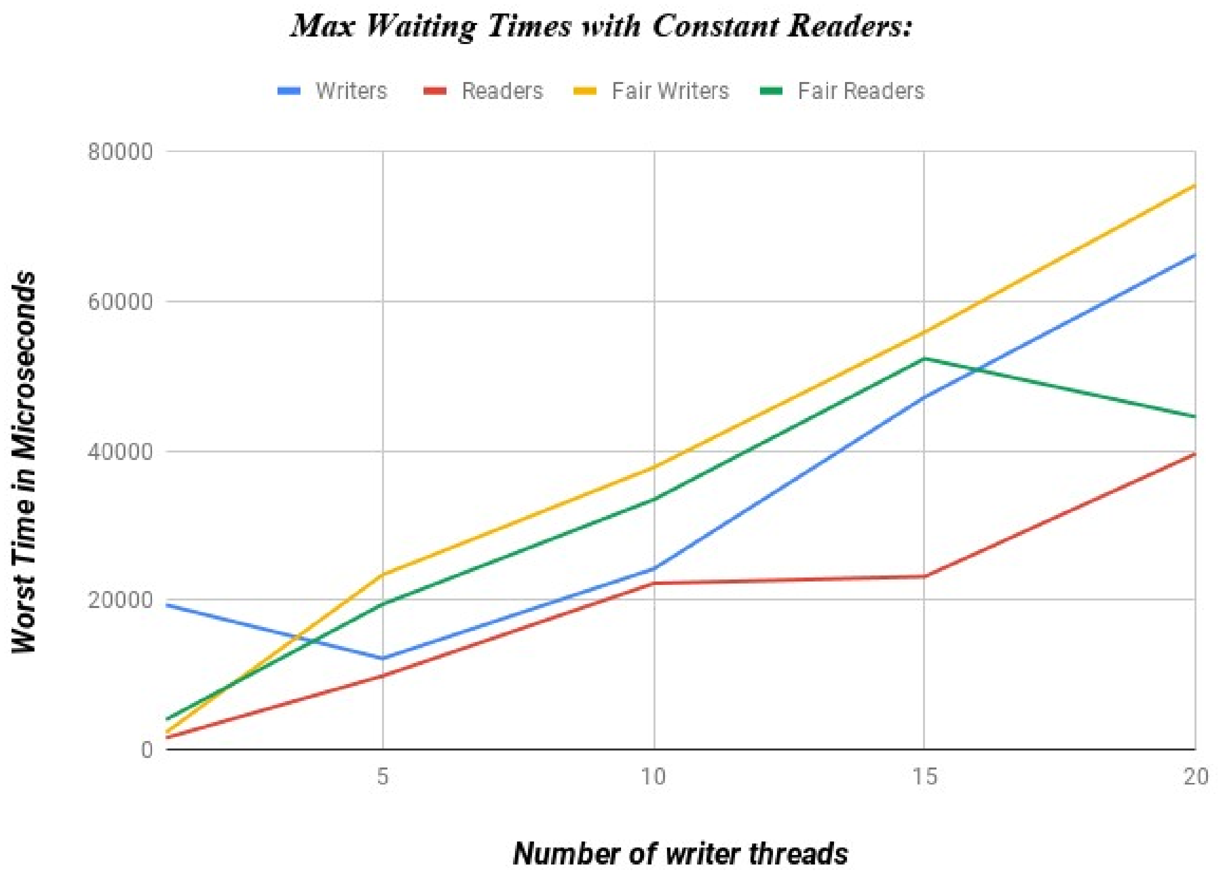
Average time taken at constant readers:



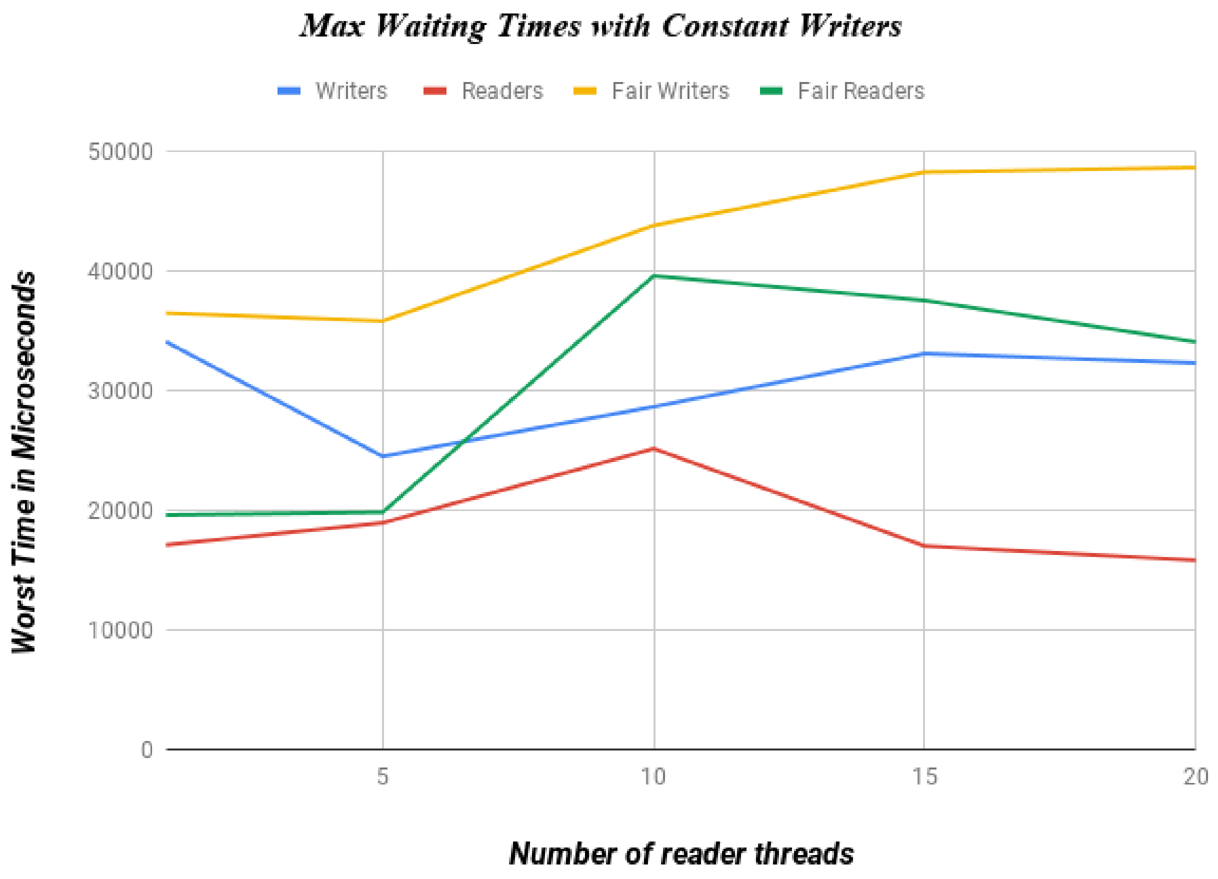
Average time taken at constant writers:



## Worst time taken at constant readers:



Worst time taken at constant writers:



<u>Conclusion</u>:

As we can see, average time taken by the Reader-Writer is less than the fair RW. Also, the worst time taken by the RW is the less than the fair RW, it is because of the fact that we used an extra semaphore in the implementation of fair RW. The functions of the semaphore are increasing the overall time for the fair RW. As the expectation should be that fair RW takes less time for worst case of writers. But, there is no such corner case here, so the overhead of the semaphore dominates and increases the overall time.