Operating Systems - 2 : CS3523 Spring 2022
Programming Assignment 5 - Implement solutions to Readers-Writers problem using Semaphores

Question:
Implement two solutions using semaphores for the Reader-Writers Problem. One solution with reader preference (Readers-Writers) and one fair solution (Fair-Readers-Writers). You have to implement these two algorithms and compare the average and worst-case time taken for each thread to access the critical section (shared resources).

Input:
Here, the input should be in the form

<nw> <nr> <kw> <kr> <l1> <l2>

Implementation:

For Readers Writers Problem:
Here, we take 2 semaphores and an int

```
semaphore rw_mutex = 1;
semaphore mutex = 1;
int read_count = 0;
```

Here, the rw_mutex is the semaphore which is locked when at least one reader or one writer is accessing the resource / document. mutex is the semaphore which is locked when a reader is trying to enter or exit the mutex to change the readcount so that the readcount is not changed in 2 threads at the same time.

Here, for the writers process, we check whether anyone is present using the rw_mutex and then enter the critical section. The readers process takes mutex and then changes the mutex to lock and then increases the readcount. If the readcount is 1 now, i.e, if the document is not having any visitors till now, So, rw_mutex is locked now. After that the readers do their work and when they are leaving, it checks the mutex and then it takes the mutex and decreases the readcount again. Now, if readcount = 0, then it frees the rw_mutex and then finishes it's execution.

For Fair Readers-Writers Problem:

Here, in Readers-Writers Problem, we give priority to the reader. So, sometimes the writer threads may starve. To avoid that problem, we use a new algorithm named Fair Readers-Writers Algorithm. Here, we use 3 semaphores and an int

```
int readcount;
semaphore resource;
semaphore rmutex;
semaphore servicequeue;
```

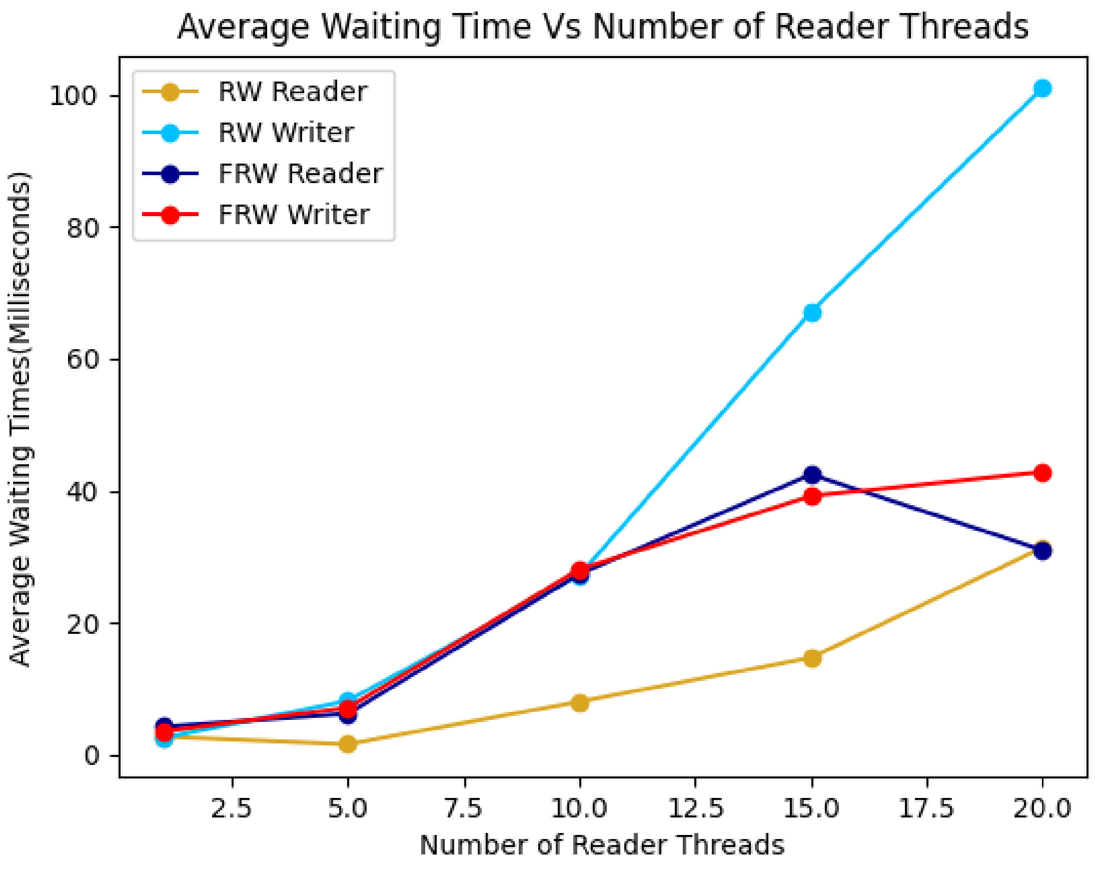
Here, we do the same thing as above but here, we use a semaphore named servicequeue which checks whether there is any reader or writer waiting in the queue for the critical section and then if there is any one waiting then it locks itself and doesn't allow the next person to wait for the resource lock until the previous one gets the lock first. In this case we can prevent the Writer from starving.

Here, in this code, we used the distribution which is an exponential distribution. It is from the library random in cpp and it chooses a random distribution such that the mean is l1 or l2 and gives the times for which the processes are executing their critical section.

Analysis:

Graph1:

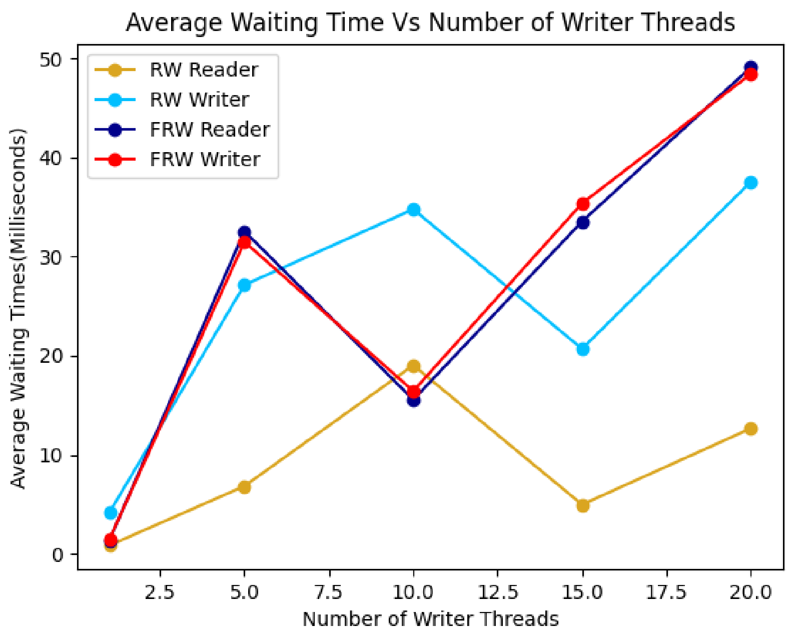
Average Waiting Time Vs Number of Reader Threads



Here, in this graph we have varied the number of reader threads and checked the average waiting times of reader threads and writer threads. Here, we can see that the average waiting time of RW Reader is lower compared to the other threads because in the RW algorithm, we give preference to the Reader. Here, we can also see that the FRW Reader and FRW Writer waiting times are very close to each other because in this case, we give preference to both the reader and writer equally.

Graph2:

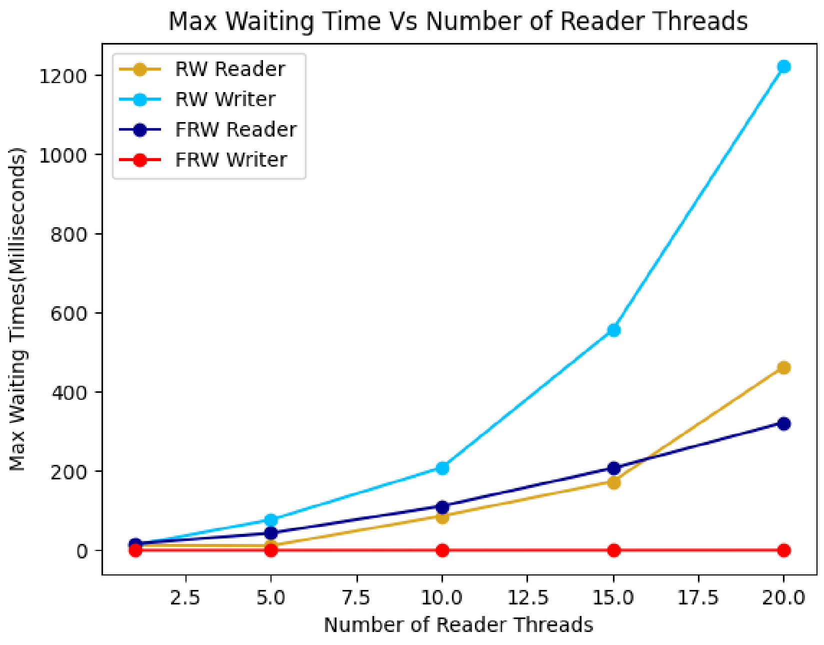
Average Waiting Time Vs Number of Writer Threads



Here, we can see that all the observations we saw in Graph1 are also applicable in Graph2.

Graph3:

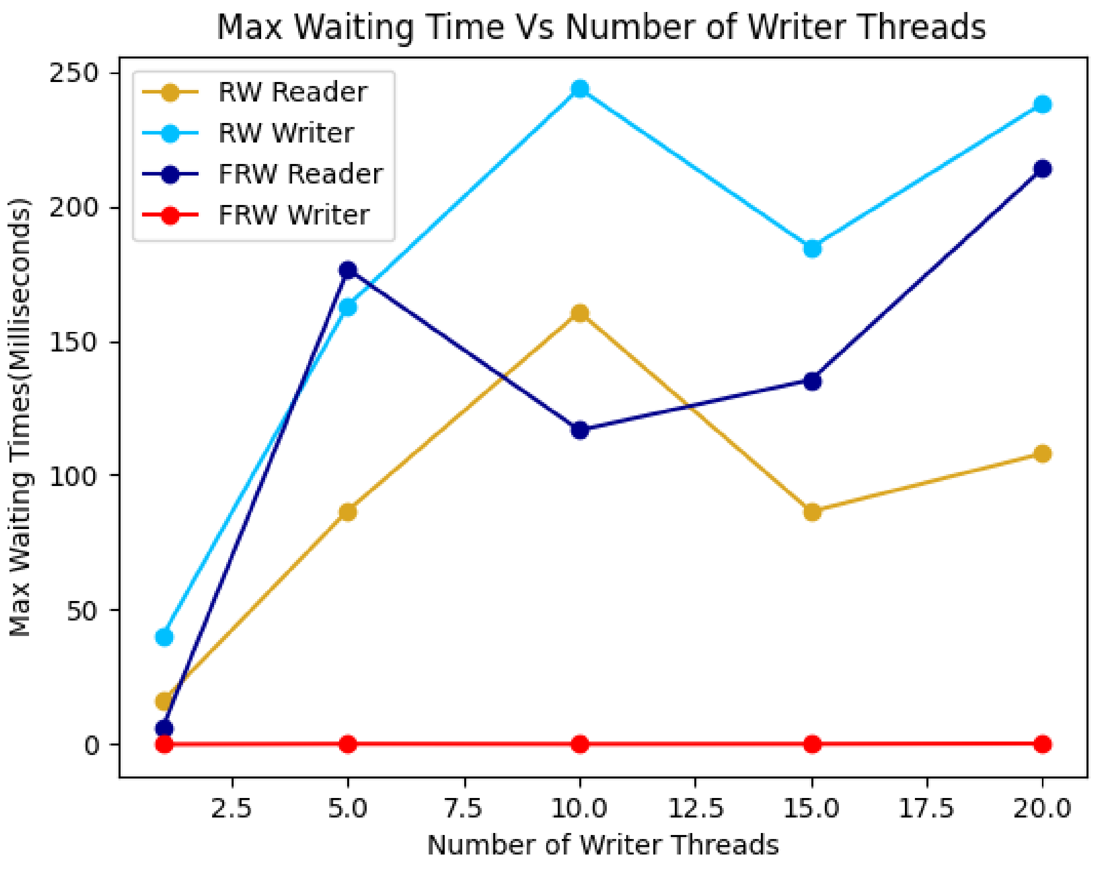
Max Waiting Time Vs Number of Reader Threads



Here, we can see that the RW Reader Max waiting time is less than RW Writer max waiting time.

Graph4:

Max Waiting Time Vs Number of Writer Threads



Here, we can see that the RW Reader Max waiting time is less than RW max waiting time.