Operating System -2 Programming Assignment 4

Name : Ahmik Virani
Roll Number : ES22BTECH11001

Low Level Design

For the writers preference : no writer, once added to the queue, shall be kept waiting longer than absolutely necessary, I have followed the following pseudo code

```
int readcount, writecount;                   //(initial value = 0)
semaphore rmutex, wmutex, readTry, resource; //(initial value = 1)

//READER
reader() {
<ENTRY Section>
  readTry.P();                 //Indicate a reader is trying to enter
  rmutex.P();                  //lock entry section to avoid race
condition with other readers
  readcount++;                 //report yourself as a reader
  if (readcount == 1)          //checks if you are first reader
    resource.P();              //if you are first reader, lock  the
resource
  rmutex.V();                  //release entry section for other readers
  readTry.V();                 //indicate you are done trying to access
the resource

<CRITICAL Section>
//reading is performed

<EXIT Section>
  rmutex.P();                  //reserve exit section - avoids race
condition with readers
  readcount--;                 //indicate you're leaving
  if (readcount == 0)          //checks if you are last reader leaving
    resource.V();              //if last, you must release the locked
resource
  rmutex.V();                  //release exit section for other readers
}

//WRITER
writer() {
<ENTRY Section>
  wmutex.P();                  //reserve entry section for writers -
avoids race conditions
  writecount++;                //report yourself as a writer entering
  if (writecount == 1)         //checks if you're first writer
    readTry.P();               //if you're first, then you must lock the
readers out. Prevent them from trying to enter CS
```

```
  wmutex.V();                     //release entry section
  resource.P();                   //reserve the resource for yourself -
prevents other writers from simultaneously editing the shared resource
<CRITICAL Section>
  //writing is performed
  resource.V();                   //release file

<EXIT Section>
  wmutex.P();                     //reserve exit section
  writecount--;                   //indicate you're leaving
  if (writecount == 0)            //checks if you're the last writer
    readTry.V();                  //if you're last writer, you must unlock
the readers. Allows them to try enter CS for reading
  wmutex.V();                     //release exit section
}
```

In the above solution, Only the first writer will lock the readtry and then all subsequent writers can simply use the resource as it gets freed by the previous writer. The very last writer must release the readtry semaphore. On the other hand, if a particular reader has locked the readtry semaphore, this will indicate to any potential concurrent writer that there is a reader in the entry section. So the writer will wait for the reader to release the readtry and then the writer will immediately lock it for itself and all subsequent writers. However, the writer will not be able to access the resource until the current reader has released the resource, which only occurs after the reader is finished with the resource in the critical section.

For the fair readers writers solution, I have followed the following pseudo code

```


int readcount;                   // init to 0; number of readers currently
accessing resource

// all semaphores initialised to 1
semaphore resource;              // controls access (read/write) to the
resource. Binary semaphore.
semaphore rmutex;                // for syncing changes to shared variable
readcount
semaphore serviceQueue;          // FAIRNESS: preserves ordering of requests
(signaling must be FIFO)

//READER
reader() {
<ENTRY Section>
  serviceQueue.P();              // wait in line to be serviced
  rmutex.P();                    // request exclusive access to readcount
  readcount++;                   // update count of active readers
  if (readcount == 1)            // if I am the first reader
    resource.P();                // request resource access for readers
(writers blocked)
```

```
    serviceQueue.V();              // let next in line be serviced
    rmutex.V();                    // release access to readcount

  <CRITICAL Section>
  //reading is performed

  <EXIT Section>
    rmutex.P();                    // request exclusive access to readcount
    readcount--;                   // update count of active readers
    if (readcount == 0)            // if there are no readers left
      resource.V();                // release resource access for all
    rmutex.V();                    // release access to readcount
  }

  //WRITER
  writer() {
  <ENTRY Section>
    serviceQueue.P();              // wait in line to be serviced
    resource.P();                  // request exclusive access to resource
    serviceQueue.V();              // let next in line be serviced

  <CRITICAL Section>
  // writing is performed

  <EXIT Section>
    resource.V();                  // release resource access for next
  reader/writer
  }
```

In the above code : no thread shall be allowed to starve; that is, the operation of obtaining a lock on the shared data will always terminate in a bounded amount of time.

Reference : Both of the above code are from Wikipedia page :
https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem
(This was hinted in class and the same pseudocodes were discussed when sir was discussing the solutions of the theory assignment, therefore I have used these to write my answer)

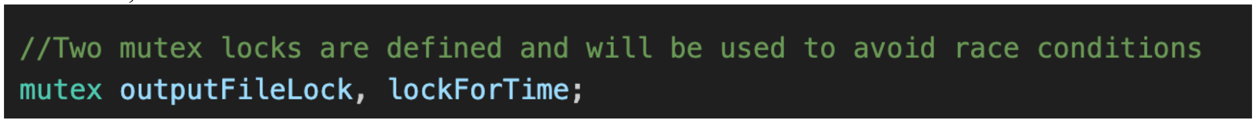
Please Note from the problem statement : Each of these threads will access the shared object(or Critical Section), kw or kr times.
But in the implementation sir showed, it had kw + 1 or kr + 1 times,

```
for(int i=0; i ≤ kw ;i++)
{
```

(because of the <= sign)

So I modified this and removed the = sign.

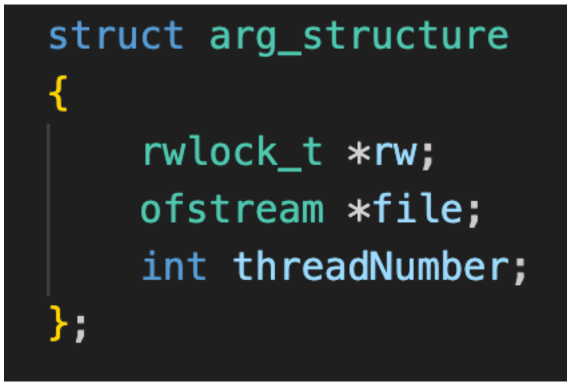
Also note, that I have used additional locks



```
//Two mutex locks are defined and will be used to avoid race conditions
mutex outputFileLock, lockForTime;
```

The outputFileLock -> To avoid race conditions while writing to the output file
The lockForTime -> To avoid race condition when multiple threads try to modify the various global variables in the code

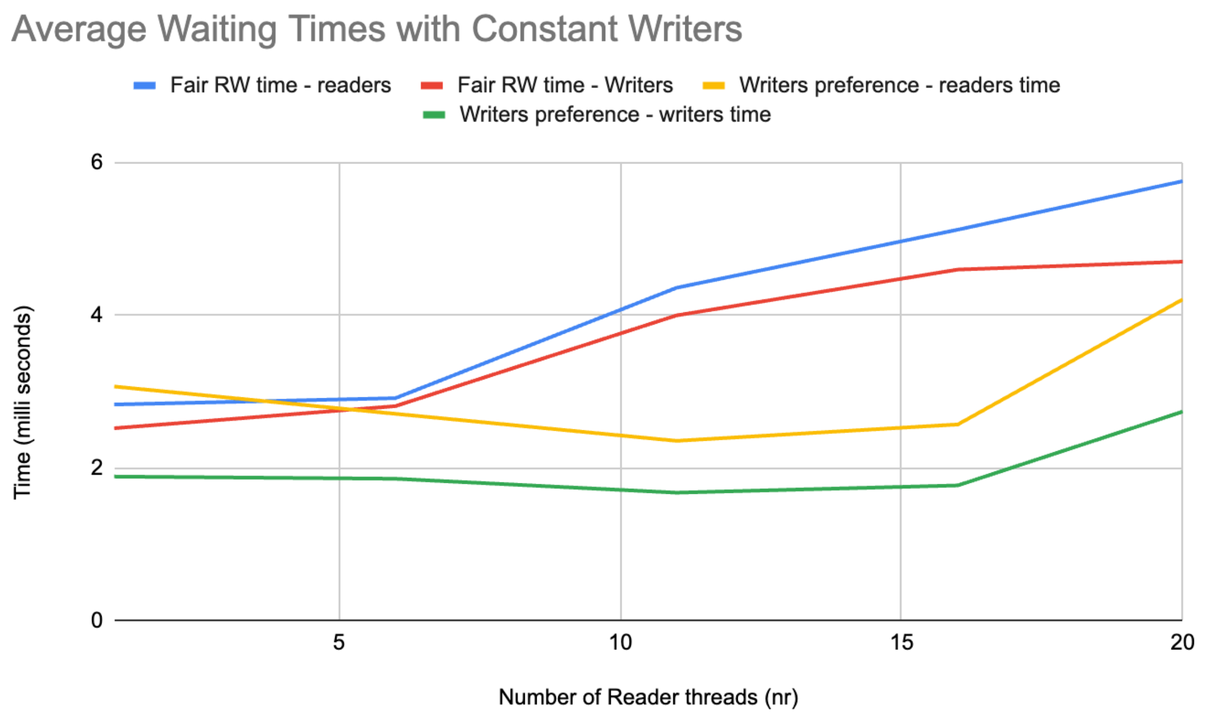
Also, here since I had to pass several arguments to the reader/writer functions, I have used a structure to pass multiple variables to the function.



```
struct arg_structure
{
    rwlock_t *rw;
    ofstream *file;
    int threadNumber;
};
```

Experiment Analysis

<u>Average Waiting Times with Constant Writers</u>



General Trend : As number of reader threads increase, the waiting time also increases. The reason for this is trivial, as more threads are being created and as there is a restriction for multiple threads modifying the critical section, the more other threads will have to wait on average.

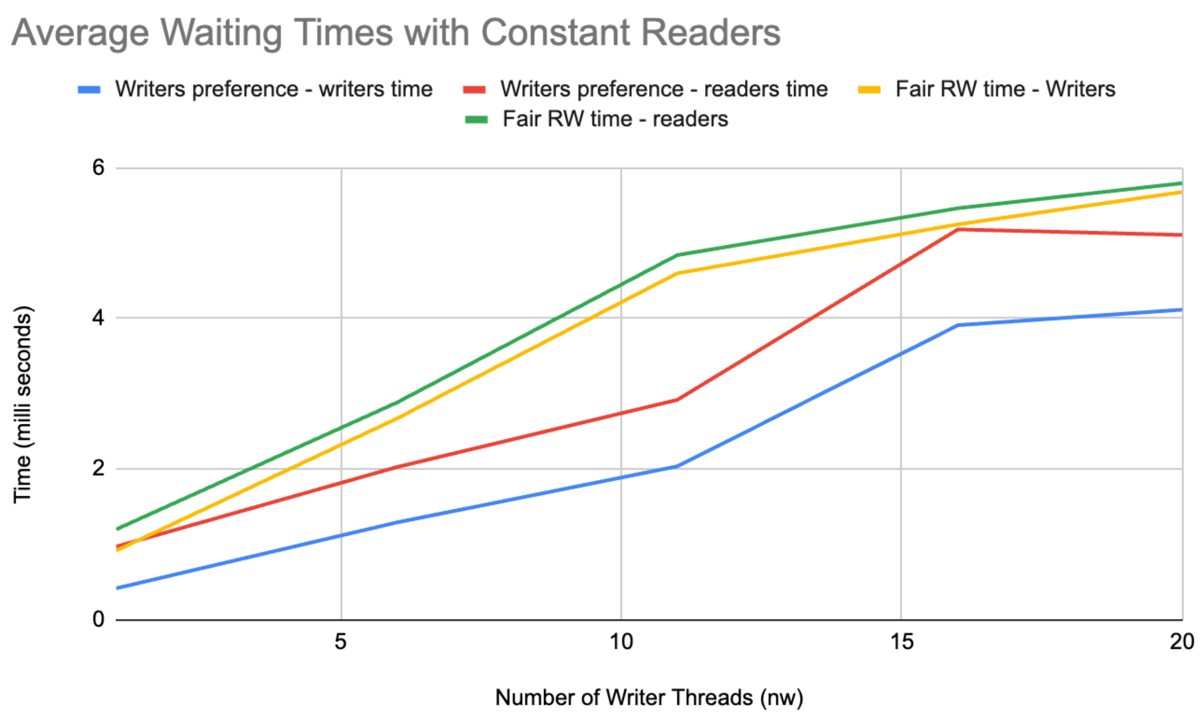
Algorithm analysis : comparing time for each algorithm
Writers preference – writers time < Writers preference – readers time < Fair RW – writers time < Fair RW – readers time

Reason for this :

1. In writers preference, writers are given priority, so no writer waits longer than necessary and finishes very quickly. Whereas on the contrary, the readers do have to wait, but their average time is lower than others because once a reader gains access to the resource, it can stay in the critical section as long as there are no writers waiting. Therefore, although readers may experience some waiting time, their overall average waiting time tends to be lower than in scenarios where writers have to wait for readers to finish.
2. Fair RW we can see that even tough writers waiting time on general is less, they are very much comparable and don't have too much significant difference compared to the other differences (except for the last case). This ensures fairness between readers and writers and prevents either group from being starved. Consequently, although writers may occasionally need to wait, they typically experience shorter waiting times compared to scenarios where they are given less priority.

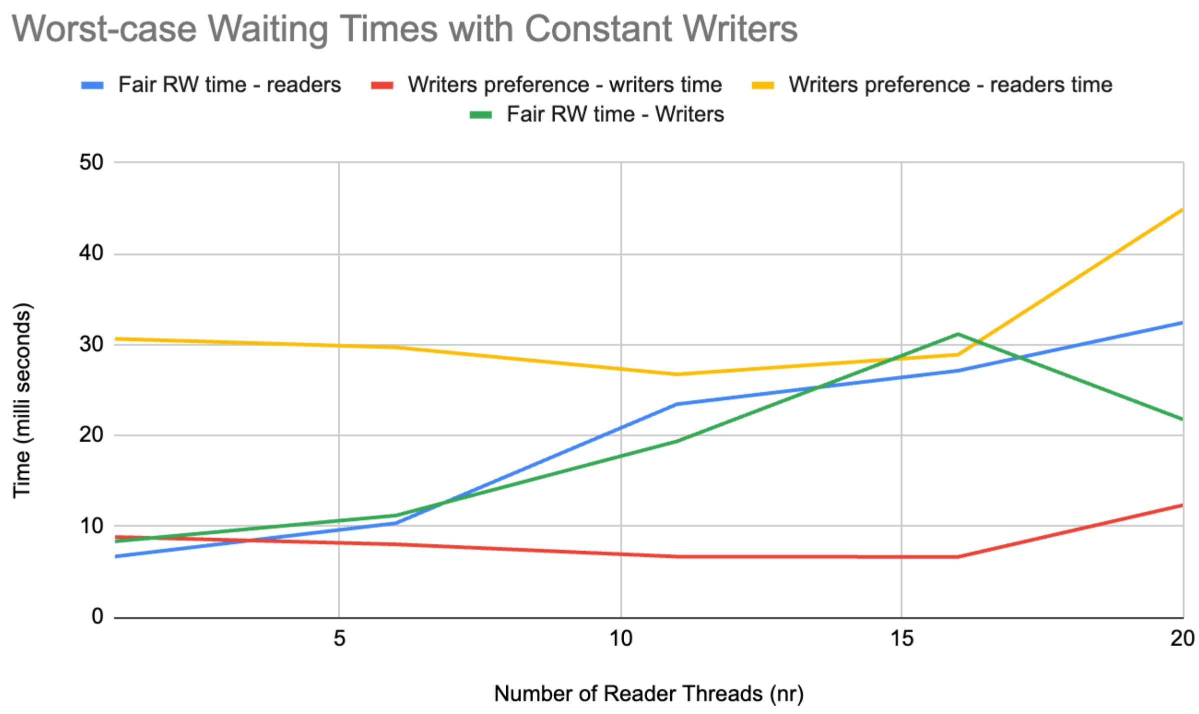
Also note that the average time for writers preference - writers thread is almost constant, the reason for this is because, even if we increase the number of readers, the priority is given to the writers, so no matter how many readers are there, the writer wont wait longer than necessary and will enter the critical section.

Average Waiting Times with Constant Readers



The trends are very much similar to the above graph and can be explained in the similar way. Also adding one thing, the time for fair RW > writers preference because in fair, both are waiting similar time which is close to the average time for any thread, but in writers preference, the writers finish fast and also multiple writers are waiting in the queue, so extra threads don't need to wait longer. In fair, it may be that first reader goes in then writer then reader, so each thread has to wait longer and closer to the worst case waiting time -> this is a possibility.

Worst Waiting Times with Constant Writers

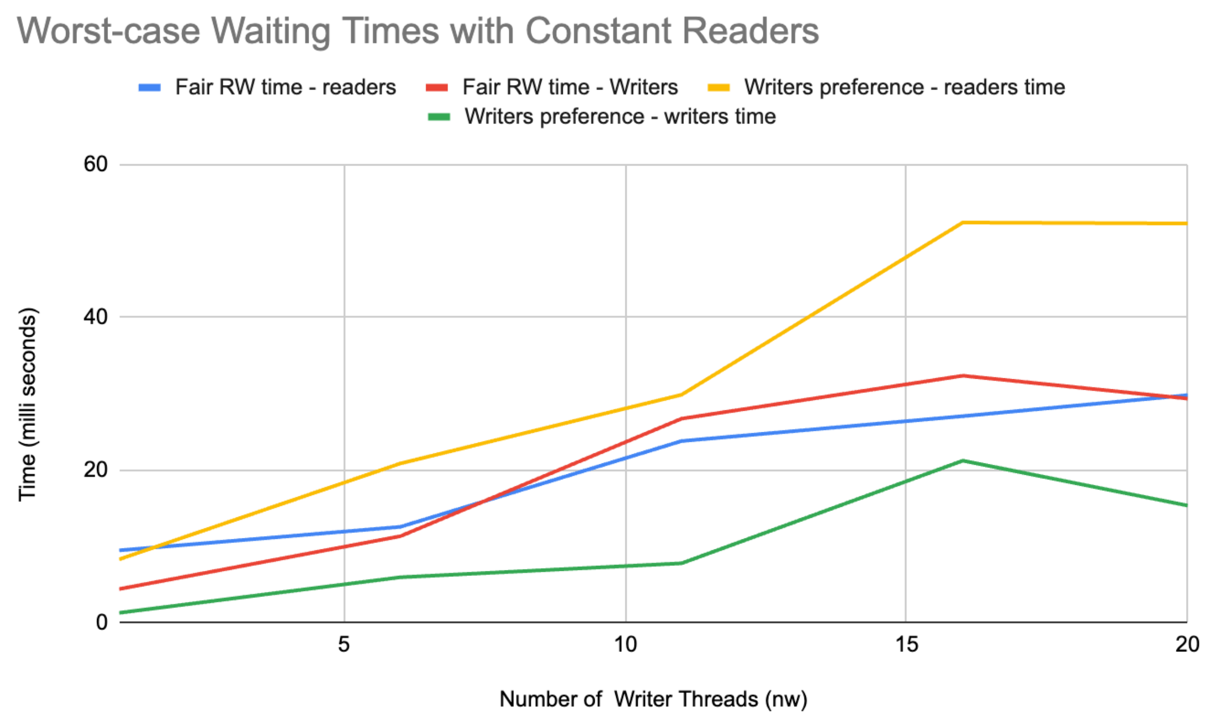


The trend is not too clear here, however we can still somewhat conclude that as number of threads increase the waiting time increases.

Here the time for writers preference – writer threads is less for the same reason as explained before.
Then comes the fair RW times which are similar due to fairness of the algorithm.
The worst time is for the writers preference – reader threads, because, not all, but some readers may have to wait for all writer threads to finish and so have a massive worst time compared to others. (If there's a continuous flow of writer threads, readers might have to wait for all writer threads to finish before gaining access to the resource, potentially leading to significant waiting times, especially in the worst-case scenario.)

Also note that the average time for writers preference - writers thread is almost constant, the reason for this is because, even if we increase the number of readers, the priority is given to the writers, so no matter how many readers are there, the writer wont wait longer than necessary and will enter the critical section.

Worst Waiting Times with Constant Writers



Here again same trend as before, and same explanation.

Conclusion

General trend : As number of threads increase, the waiting time increases

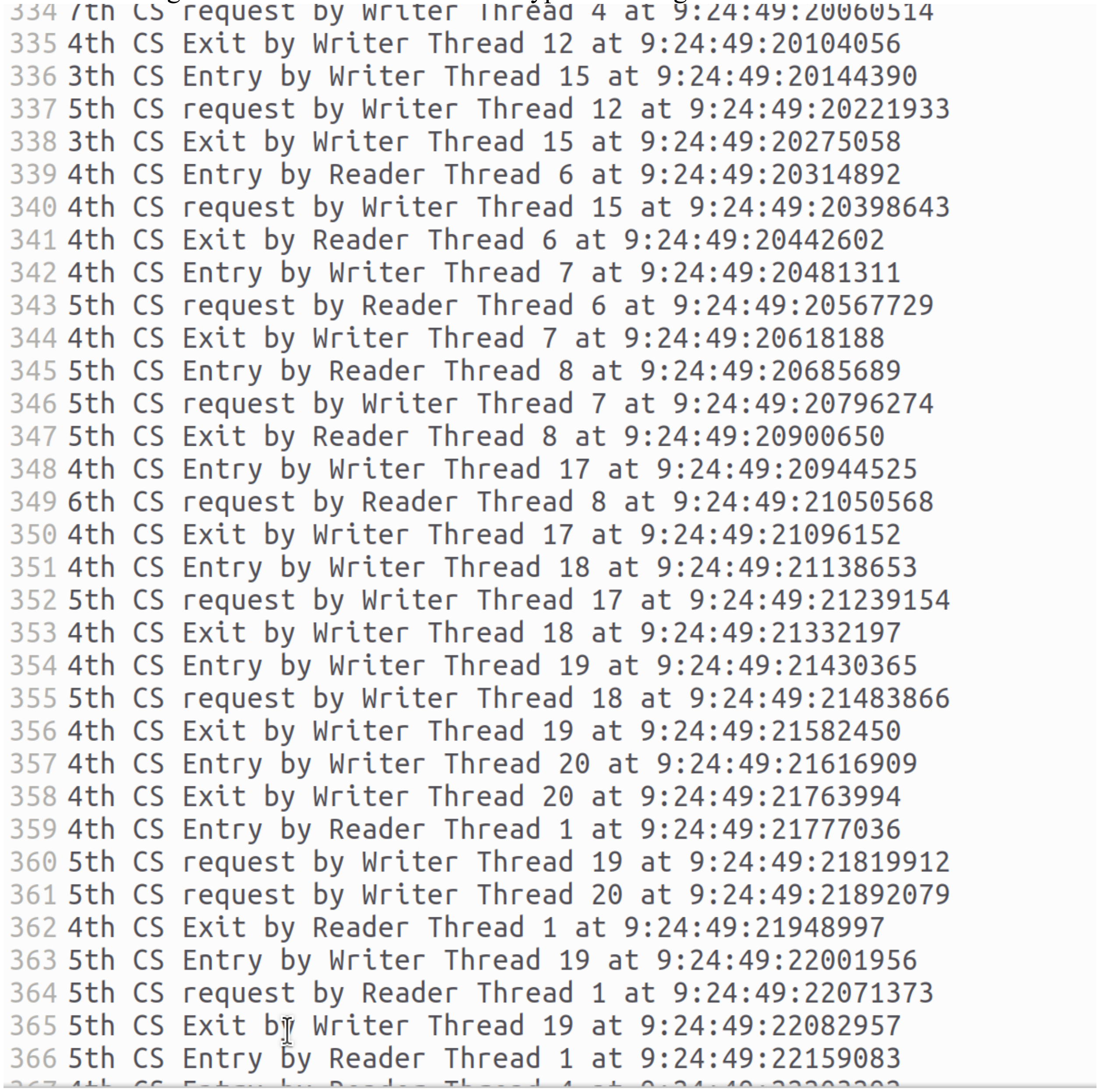
Algorithm analysis :

1. Average time : writers preference – writer threads < writer preference – reader threads < Fair RW thread (both reader and writer have similar time)
2. Worst time : writers preference – writer threads < Fair RW thread (both reader and writer have similar time) < writer preference – reader threads
3. The average time for writers preference – writers thread is almost constant

Mutual Exclusion

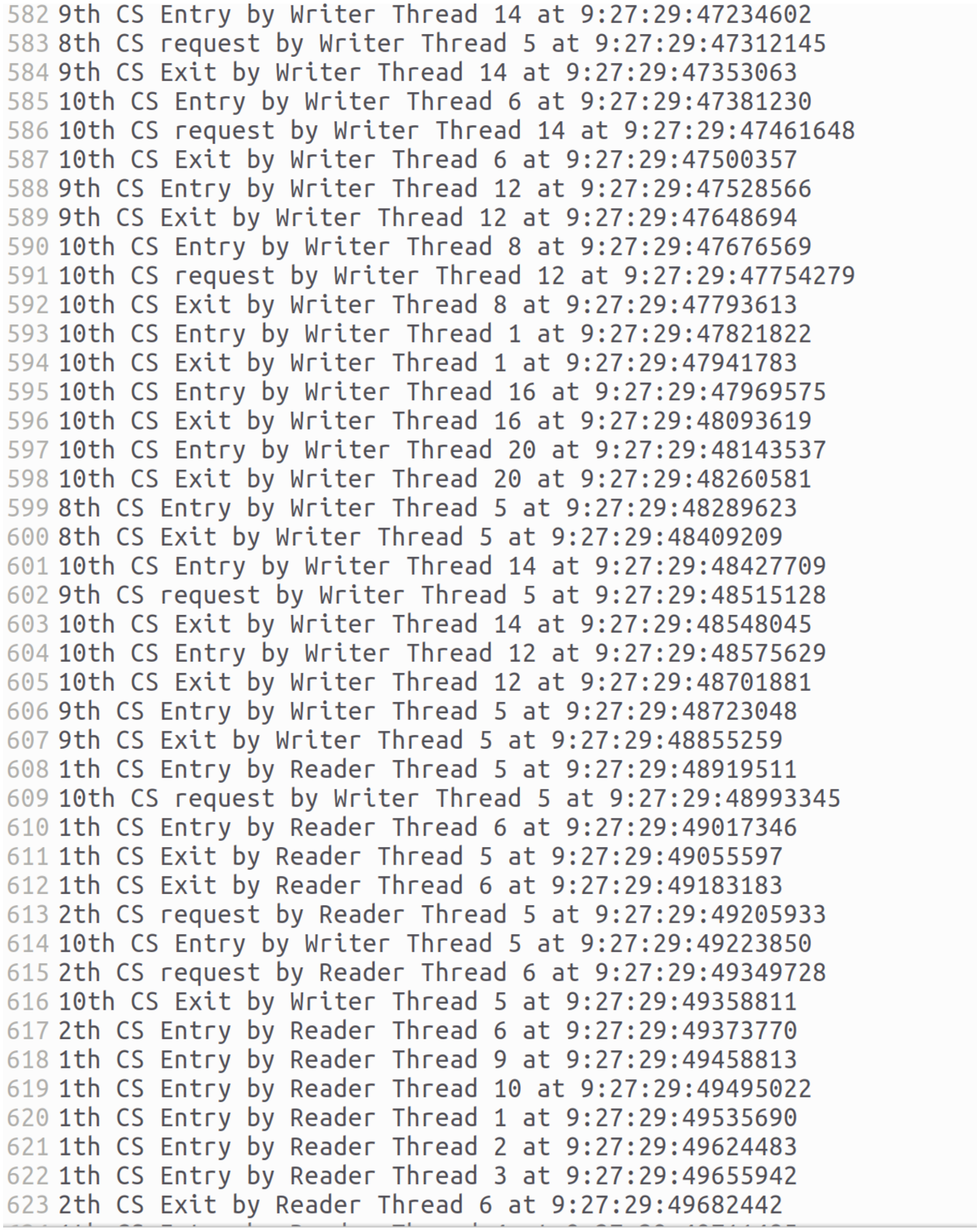
Fair solution

Clearly we can see that no threads are entering the critical section at the same time in fair RW solution as long as another thread of the other type is waiting.



```
7th CS request by Writer Thread 4 at 9:24:49:20060514
4th CS Exit by Writer Thread 12 at 9:24:49:20104056
3th CS Entry by Writer Thread 15 at 9:24:49:20144390
5th CS request by Writer Thread 12 at 9:24:49:20221933
3th CS Exit by Writer Thread 15 at 9:24:49:20275058
4th CS Entry by Reader Thread 6 at 9:24:49:20314892
4th CS request by Writer Thread 15 at 9:24:49:20398643
4th CS Exit by Reader Thread 6 at 9:24:49:20442602
4th CS Entry by Writer Thread 7 at 9:24:49:20481311
5th CS request by Reader Thread 6 at 9:24:49:20567729
4th CS Exit by Writer Thread 7 at 9:24:49:20618188
5th CS Entry by Reader Thread 8 at 9:24:49:20685689
5th CS request by Writer Thread 7 at 9:24:49:20796274
5th CS Exit by Reader Thread 8 at 9:24:49:20900650
4th CS Entry by Writer Thread 17 at 9:24:49:20944525
6th CS request by Reader Thread 8 at 9:24:49:21050568
4th CS Exit by Writer Thread 17 at 9:24:49:21096152
4th CS Entry by Writer Thread 18 at 9:24:49:21138653
5th CS request by Writer Thread 17 at 9:24:49:21239154
4th CS Exit by Writer Thread 18 at 9:24:49:21332197
4th CS Entry by Writer Thread 19 at 9:24:49:21430365
5th CS request by Writer Thread 18 at 9:24:49:21483866
4th CS Exit by Writer Thread 19 at 9:24:49:21582450
4th CS Entry by Writer Thread 20 at 9:24:49:21616909
4th CS Exit by Writer Thread 20 at 9:24:49:21763994
4th CS Entry by Reader Thread 1 at 9:24:49:21777036
5th CS request by Writer Thread 19 at 9:24:49:21819912
5th CS request by Writer Thread 20 at 9:24:49:21892079
4th CS Exit by Reader Thread 1 at 9:24:49:21948997
5th CS Entry by Writer Thread 19 at 9:24:49:22001956
5th CS request by Reader Thread 1 at 9:24:49:22071373
5th CS Exit by Writer Thread 19 at 9:24:49:22082957
5th CS Entry by Reader Thread 1 at 9:24:49:22159083
```

Writers preference



```
9th CS Entry by Writer Thread 14 at 9:27:29:47234602
8th CS request by Writer Thread 5 at 9:27:29:47312145
9th CS Exit by Writer Thread 14 at 9:27:29:47353063
10th CS Entry by Writer Thread 6 at 9:27:29:47381230
10th CS request by Writer Thread 14 at 9:27:29:47461648
10th CS Exit by Writer Thread 6 at 9:27:29:47500357
9th CS Entry by Writer Thread 12 at 9:27:29:47528566
9th CS Exit by Writer Thread 12 at 9:27:29:47648694
10th CS Entry by Writer Thread 8 at 9:27:29:47676569
10th CS request by Writer Thread 12 at 9:27:29:47754279
10th CS Exit by Writer Thread 8 at 9:27:29:47793613
10th CS Entry by Writer Thread 1 at 9:27:29:47821822
10th CS Exit by Writer Thread 1 at 9:27:29:47941783
10th CS Entry by Writer Thread 16 at 9:27:29:47969575
10th CS Exit by Writer Thread 16 at 9:27:29:48093619
10th CS Entry by Writer Thread 20 at 9:27:29:48143537
10th CS Exit by Writer Thread 20 at 9:27:29:48260581
8th CS Entry by Writer Thread 5 at 9:27:29:48289623
8th CS Exit by Writer Thread 5 at 9:27:29:48409209
10th CS Entry by Writer Thread 14 at 9:27:29:48427709
9th CS request by Writer Thread 5 at 9:27:29:48515128
10th CS Exit by Writer Thread 14 at 9:27:29:48548045
10th CS Entry by Writer Thread 12 at 9:27:29:48575629
10th CS Exit by Writer Thread 12 at 9:27:29:48701881
9th CS Entry by Writer Thread 5 at 9:27:29:48723048
9th CS Exit by Writer Thread 5 at 9:27:29:48855259
1th CS Entry by Reader Thread 5 at 9:27:29:48919511
10th CS request by Writer Thread 5 at 9:27:29:48993345
1th CS Entry by Reader Thread 6 at 9:27:29:49017346
1th CS Exit by Reader Thread 5 at 9:27:29:49055597
1th CS Exit by Reader Thread 6 at 9:27:29:49183183
2th CS request by Reader Thread 5 at 9:27:29:49205933
10th CS Entry by Writer Thread 5 at 9:27:29:49223850
2th CS request by Reader Thread 6 at 9:27:29:49349728
10th CS Exit by Writer Thread 5 at 9:27:29:49358811
2th CS Entry by Reader Thread 6 at 9:27:29:49373770
1th CS Entry by Reader Thread 9 at 9:27:29:49458813
1th CS Entry by Reader Thread 10 at 9:27:29:49495022
1th CS Entry by Reader Thread 1 at 9:27:29:49535690
1th CS Entry by Reader Thread 2 at 9:27:29:49624483
1th CS Entry by Reader Thread 3 at 9:27:29:49655942
2th CS Exit by Reader Thread 6 at 9:27:29:49682442
```

We can clearly see that here first writer threads are finishing their execution in the critical section but no 2 are writing at the same time, the next writer is entering after the previous leaves.

But for reader, we can see at the end of the screen shot, that several readers are entering at the same time.

Pleas note that I have run my program on a mac with virtual machine. Due to change in environment i.e. OS different results may be obtained and for those maybe different explanations may be given