ASSIGNMENT 4 OPERATING SYSTEM

Name:- Ashwin kumar

Roll:- CE21BTECH11008

Implementation of the program

Reading The file My input file name is “inp.txt” and there are two output file name is “out.txt” . For OutFile I have made a file pointer that is called at the end of the program and getting written using answer matrix.

INPUT:-

For input file using ifstream from fstream library and taking input in the main () function. Input contains a line containing nw,nr,kw,kr,mucs,murem respectively each separated by the blank space.

nw=number of writer threads

nr=number of reader threads

kw=number of times each writer threads tries to enter the critical section

kr=number of times each reader threads tries to enter the critical section

mucs= mean of cs time (in exponential distribution)

murem= mean of remainder section time (in exponential distribution)

OUTPUT: -

Two output files are made to display the log of all the events as shown for each of the algorithms.

output files: RW-log.txt (containing the log event of reader writer with writer preference) and FairRW-log.txt (containing the log event of fair reader writer algorithm), consisting of events.

Output in log (format)

1st CS Request by Writer Thread 1 at 923:256

1st CS Entry by Writer Thread 1 at 923:262 ...

Time 923:256:-(millisecond: microsecond) -> 923 is showing milliseconds of the time stamp while 256 is showing microsecond part of the time stamp. Second hand was not needed because this microsecond part will cover all such time.

To record their average time a thread is taking made another file "Average time.txt", consisting of the average time a thread takes to gain entry to the Critical

Section for each algorithm: RW and Fair-RW.

To measure Time:

For time stamp measurement using a function name current time whose return type is string and returning a string in format "millisecond: microsecond". This function also takes one argument of time and store the current time in microsecond (double *time). It is helpful in finding difference between two time like it is used when a program requested critical section (double timeStamp1) and when program get chance to run in critical section (double timeStamp2) and at last for each for loop I am adding to sum=sum+(timeStamp2-timeStamp1).

Time for each reader/writer:

As sum is storing total sum of time taken for given thread to get critical section after request and as each thread is going kw(if it is writer threads) and kr(if it is a reader threads) thus for each thread finding the average time taken by each reader and writer thread by dividing the sum/kr or sum/kw. Maintain a separate array for each thread index to store time taken in that as each thread has been allocated fixed index so no possibility of race condition is there.

Worst & Average case timing: -

Note:- I have taken average to report a time taken by a thread and for worst I will find minimum of those average time of each thread and for average I will find average of the average time for each thread.

To simulate the work done

In critical section used "usleep (randCSTime(mucst)) " Here randCSTime is a function made that produces exponential distribution. With the mean of mucst .

In remainder section "usleep(randRemTime(murem))". Here randRemTime is a function made that produces the exponential distribution with the mean murem.

Note: - Both (randCSTime and randRemTime do exactly same job but for clarification made two different function)

Threading Methodology I have used Posix thread and called pthread_create function with its arguments and for parameter passing I have made struct data type. That is containing N and K and 2d vector matrix to be passed as this is the data used by all. For passing pthread_create I have made array of pthread_t of size k where the function instance will be stored. And last have called pthread_join function as no return value is there so passed its argument as null and thread id th[i].

Implementation of Reader-Writer's problem (writer preference)

In this reader writer problem with writer preference is opposite to general reader-writer problem. In this writer is given more preference means when a writer started executing then it will execute all available writers keeping all the reader to the hold. In this case it is possible than reader thread may starve.

Code implementation: -

```
int readcount = 0;  //(initial value = 0)
int writecount = 0; //(initial value = 0)
sem_t available_reseource;
sem_t lock_reader;
sem_t lock_writer;
sem_t queueservice;
```

For this maintain separate variable of readcount and wirtecount with four semaphore . First is available_resource that will tell resource if available or not , separate lock for reader and writer and queueservice.

//reader implementation

```
        sem_wait(&queueservice);
        sem_wait(&lock_reader);
        readcount++;

        if (readcount == 1)
            sem_wait(&available_reseource);

        sem_post(&lock_reader);
        sem_post(&queueservice);

        // this the citical section and at a type many readers can run in critical section

        // code for thread to exit the criticalsection

        sem_wait(&lock_reader);
        readcount--;

        if (readcount == 0)
```

```
            sem_post(&available_reseource);

        sem_post(&lock_reader);
```

Now for its implementation first we have kept two semaphore lock .It checks “queueservice” if available and get the reader lock (lock_reader) and increment its value after that if readcount is 1 then it grab the available resource after that it left the lock and queueservice as it is actually got the service.

Does its work in critical section and again reduce readcount by keeping the lock and if readcount become 0 then it will give signal to available resource and then unlocks the lock_reader.

One point to be noted that suppose if resource is taken by reader then it will allow many reader at a time also it will release the available resource only when there is no reader there.

//writer implementation

```
        sem_wait(&lock_writer);
        writecount++;

        if (writecount == 1)
            sem_wait(&queueservice);
        sem_post(&lock_writer);

        sem_wait(&available_reseource);

           //critical section

        sem_post(&available_reseource);

        sem_wait(&lock_writer);
        writecount--;

        if (writecount == 0)
            sem_post(&queueservice);
        sem_post(&lock_writer);
```

writer

Implementation of Reader-Writer’s problem (Fair solution) it will increase the writecount and if writecount will be 1 it will grab the queueservice and it will keep till writecount become 0 means when a writer thread started the executing it will allow all the writer thread to allow run keeping hold of reader thread.Thus it will prefer writer then reader .It was not similar to reader section as in reader section the service queue grab and release just after increase of readcount.

Implementation of Reader-Writer's problem (Fair solution)

It is fair solution of reader writer means which thread ask for the cs first get chance to run in critical section. The code we have implemented previous like reader writer and reader writer (with writer preference) will lead to starvation and thus is not fair so to avoid this fair solution is implemented.The operation of obtaining a lock on the shared data will always terminate in a bounded amount of time

Code implementation.

```
int readcount = 0; //(initial value = 0)
sem_t available_reseource;
sem_t lock_reader;
sem_t queueservice;
```

readcount variable is maintained and initialize with 0 and three semaphores i.e. available_resource that will tell critical section is acquired or not and lock for the reader, queueservice for the fairness.

//Readers code

```
        sem_wait(&queueservice);
        sem_wait(&lock_reader);
        readcount++;

        if (readcount == 1)
            sem_wait(&available_reseource);

        sem_post(&queueservice);
        sem_post(&lock_reader);

        // this the citical section and at a type many readers can run in

        sem_wait(&lock_reader);
        readcount--;
        if (readcount == 0)
            sem_post(&available_reseource);

        sem_post(&lock_reader);
```

For implementation first we have kept two semaphore lock. It checks "queueservice" if available and get the reader lock (lock_reader) and increment its value after that if readcount is 1 then it grab the available resource after that it left the lock and queueservice as it is actually got the service.

Does its work in critical section and again reduce readcount by keeping the lock and if readcount become 0 then it will give signal to available resource and then unlocks the lock_reader. It is quite similar to previous one implementation.

//writer section

```
        sem_wait(&queueservice);
        sem_wait(&available_reseource);
        sem_post(&queueservice);
// critical section
        sem_post(&available_reseource);
```

Its implementation is quite simple like there is no readcount is there to count the number of writer.

First its calling queueservice and if it gets this then checking lock on available resource and if found it is grabbing and after that releasing the queueservice as it got served and entering into the critical section and finally coming out of the critical section by releasing the available resource.

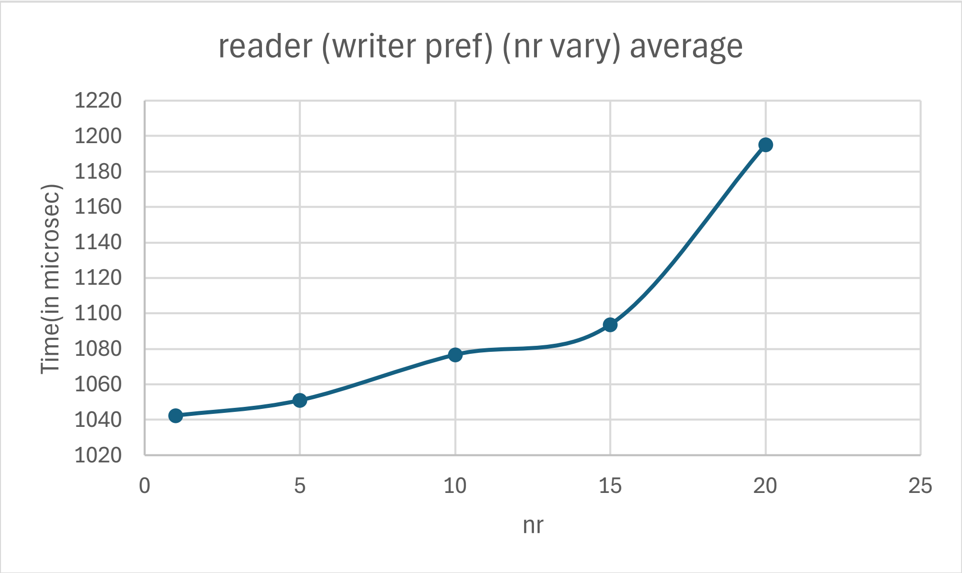
Now queueservice as defined internally make a FIFO queue means the one who raises for the critical secetion first will come and join to the tail part of queue. Thus this will ensure that the thread which raises for cs first will get chance to execute in the raised order thus it maintain the fairness.

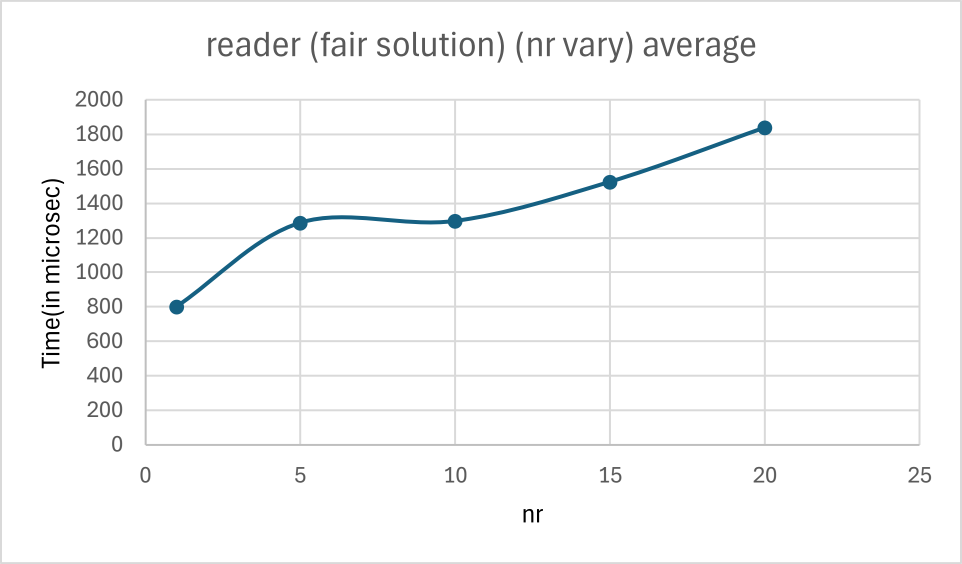
Note: I have measured the time in microsecond for better accuracy.

Q1.

we vary the number of reader **threads nr from 1 to 20** in increments of 5 on the X-axis. All the other parameters are fixed: Number of writer threads, nw = 10, kr =kw = 10. The Y-axis will have time in microsecond and will measure the average time taken to enter CS for each reader and writer thread.

(a) Average time the **reader threads** take to enter the CS for each algorithm: Readers Writers () and Fair Readers Writers().



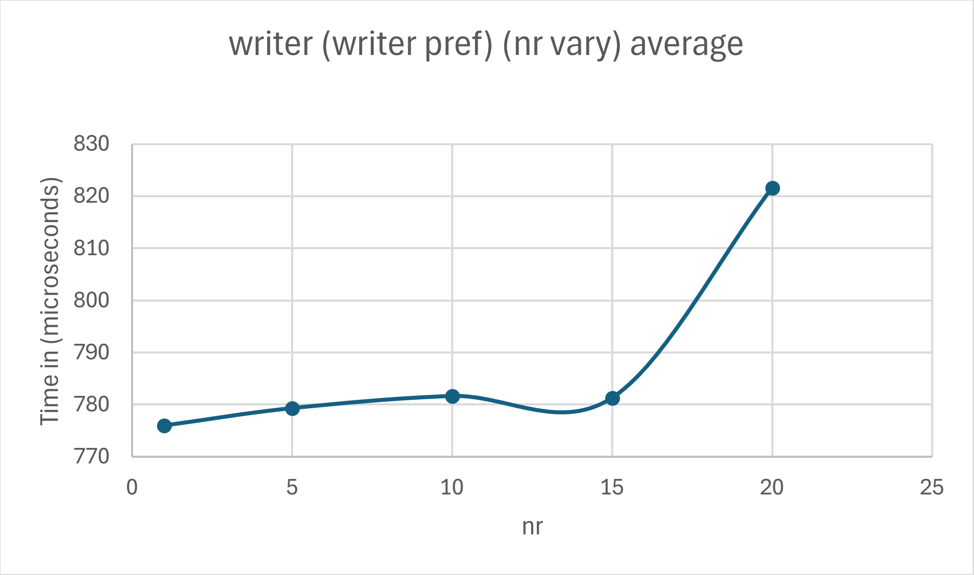


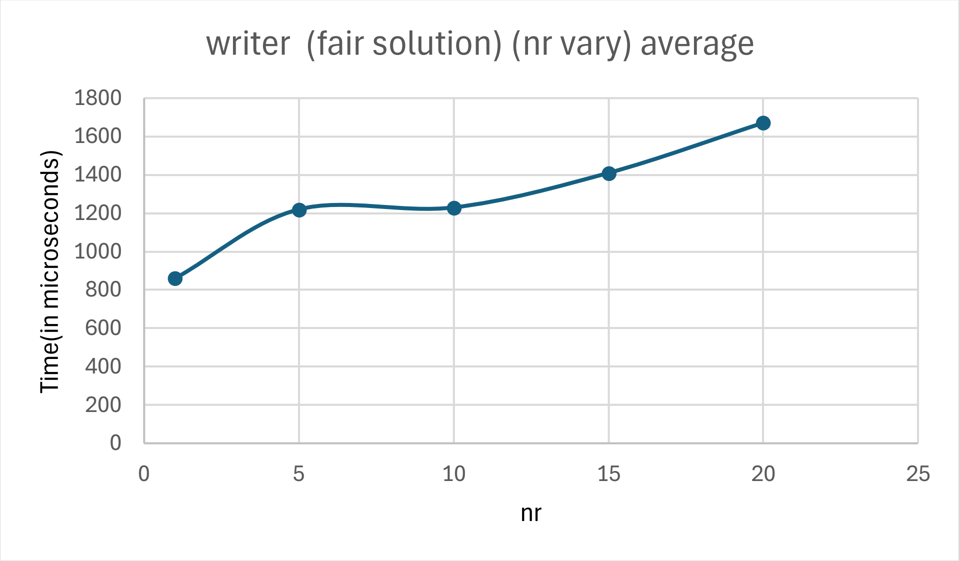
Comparison & explanation: -

In the above graph changing the number of reader thread nr = 1,5,10,15,20 respectively. It can be clearly observed that time taken by reader in reader-writer (with writer preference) is much higher than the fair solution. This is how it works because in writer preference writer get more opportunity than reader leading to reader to starve sometime.

Also, in both the cases as nr is increasing time taken by reader thread (from request to getting of critical section is also increasing). This is because more will be number of reader threads less will be the chances that a single reader thread would be chosen for execution.

(b) Average time the **writer threads** take to enter the CS for each algorithm: Readers Writers () and Fair Readers Writers ().



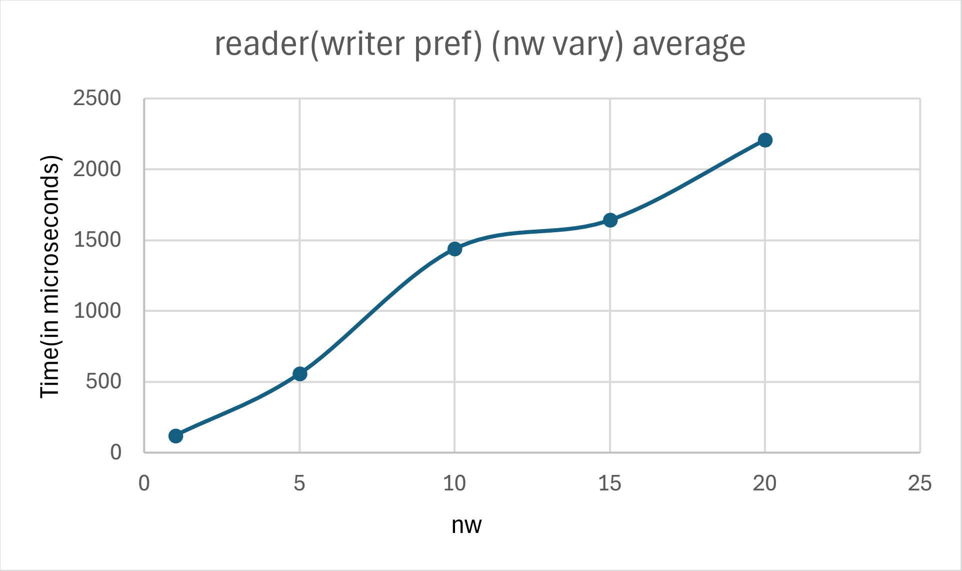


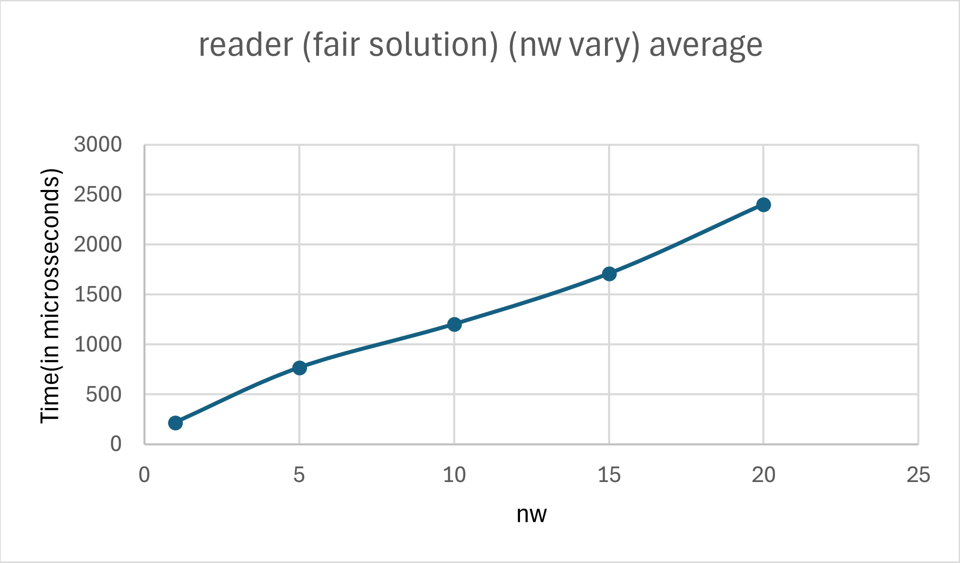
Comparison & explanation: -

Comparing above – reader writer (with writer preference) is working well than fair solution. This can be attributed to the way algorithm works that prefer reader over writer while in fair solution it have to give equal preference to the leading to more time in fair solution. Also, as the nr increasing time is also increasing because of increase of nr eventually decreasing the room for writer to go in critical section.

Q2. Here, we vary the number of writer threads nw from 1 to 20 in increments of 5 on the X-axis. All the other parameters are fixed: Number of reader threads, nr = 10, kr = kw = 10. The Y-axis will have time in milliseconds, and the average taken to enter CS will be measured for each reader and writer thread.

(A)-Average time the **reader threads** take to enter the CS for each algorithm: Readers Writers () and Fair Readers Writers ().

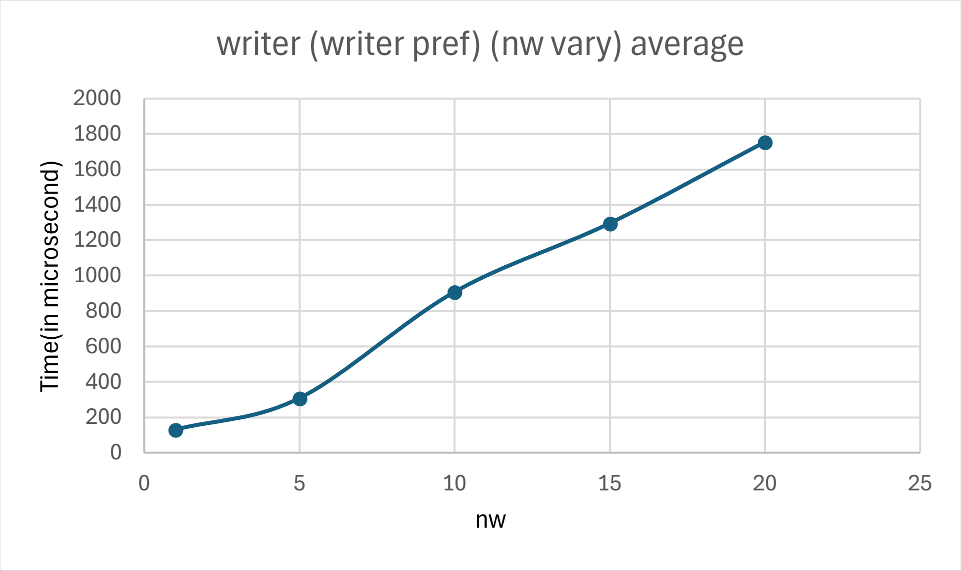


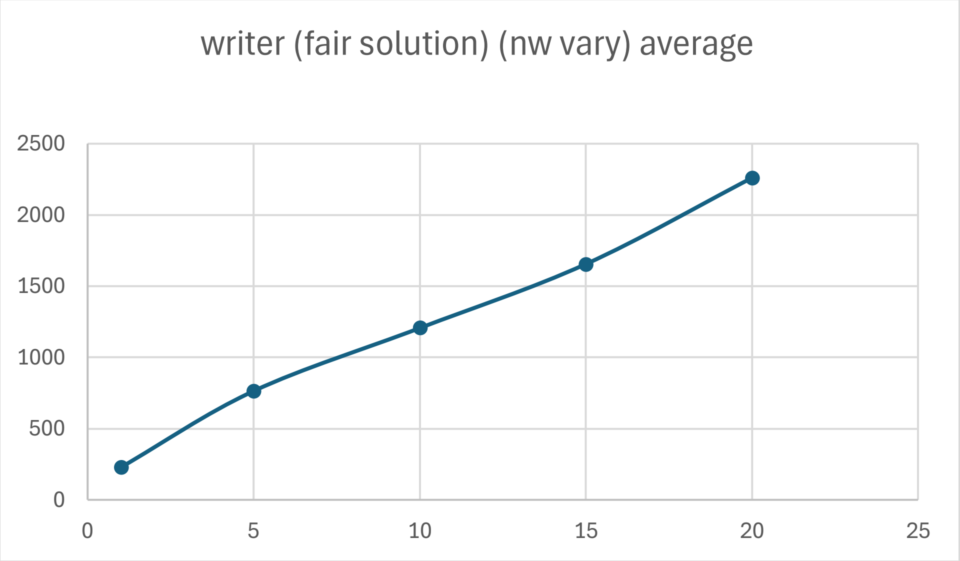


<u>Comparison & explaination</u>

Here it is observed that in most the cases reader is performing well in reader-writer (with writer preference) while at some point there is strong competition between fair solution and writer preference. It may be attributed to maintaining the reader and writer reader is getting more penalty in terms of waiting as writer threads are coming into the picture (writer threads increasing) leading to more time than writer preference solution. This may be the possible reason. Also increasing the number of writer threads increasing the time this can be explained like the previous arguments of less room to execute.

(B) -Average time **the writer threads** take to enter the CS for each algorithm: Readers Writers () and Fair Readers Writers().





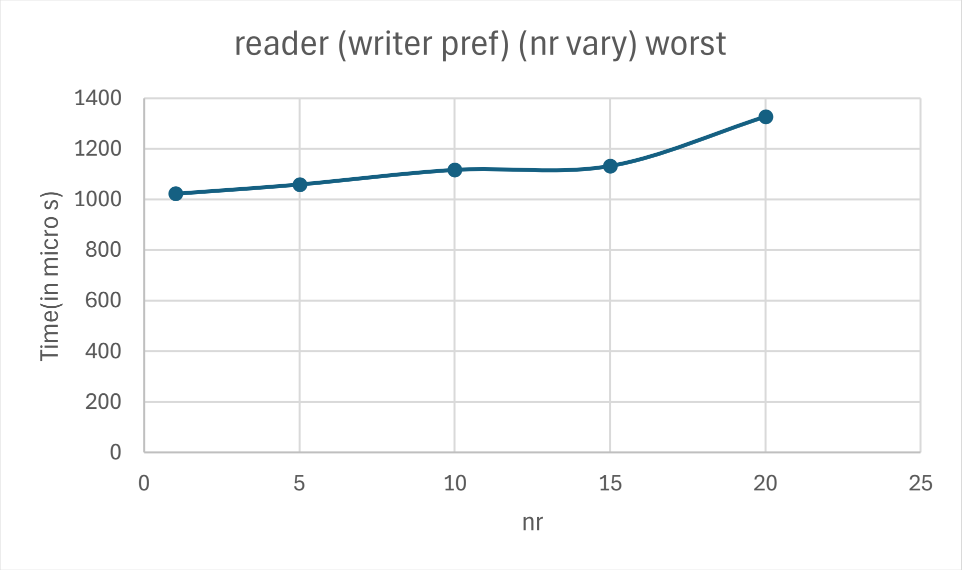
Observation & explanation

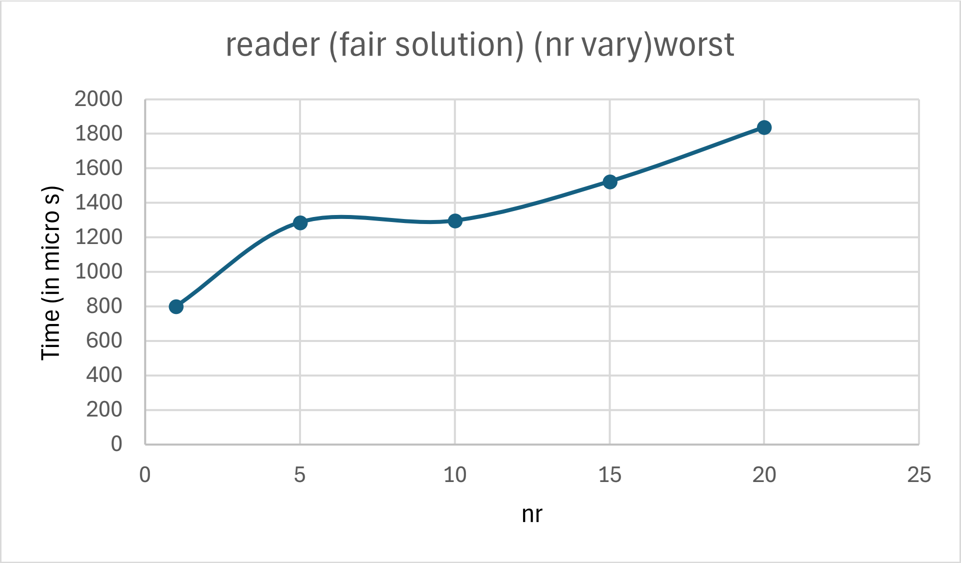
As expected, writer threads are performing reasonably well. It can be clearly observed that reader writer (with writer preference) is at least creating 600 microsecond difference between corresponding no of nw in fair solution of reader-writer. Also increasing number of writer threads lead to compactness in the critical section leading to increase in the timing. Also, graph is quite linear means increasing the number of writer thread linearly increasing the time.

Q3.Measure the worst-case (instead of average) time taken to enter the CS by reader and writer threads with a constant number of writers. Here we vary the number of reader threads nr from 1 to 20 in the increments 5 on the X-axis. All the other parameters are fixed: Number of writer threads, nw = 10, kr = kw = 10.

(a) Worst-case time taken by **the reader threads** to enter the CS for each algorithm:

Readers Writers() and Fair Readers Writers().

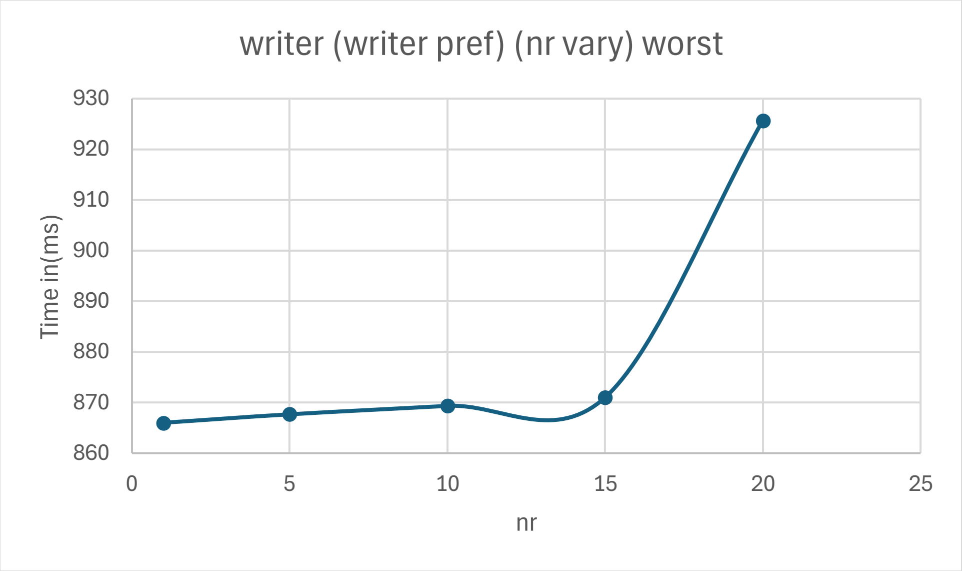


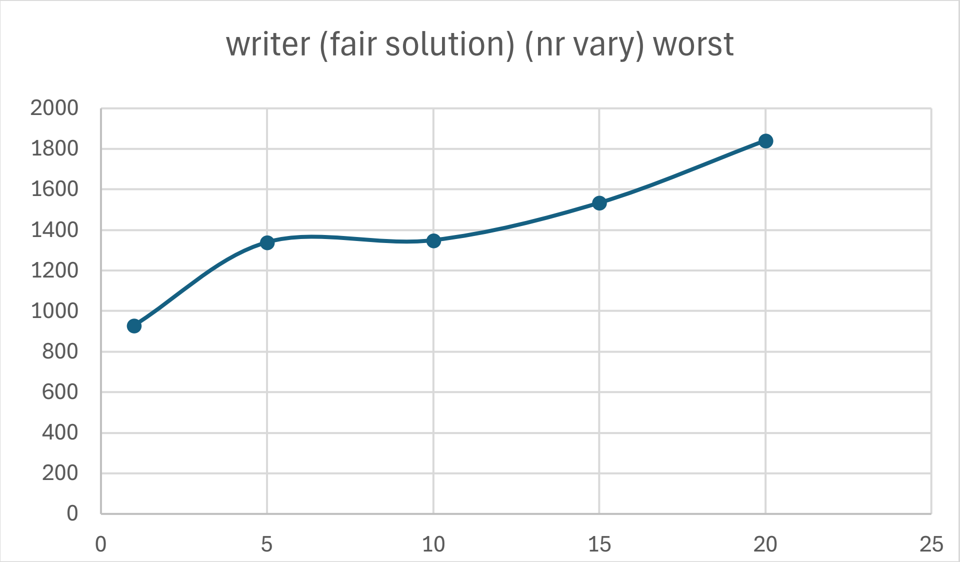


Comparison & explanation

For the worst case initially, fair solution is providing better performance when there are less number of nr while as nr increasing writer preference is providing better performance. It also be noted that worst case in writer preference is is almost constant means increasing the number thread not much affecting the worst case even though we saw previously for the same but in average case it was not like that. It may be due to every case some reader threads are there that are not getting chance even though nr is changing. While in Fair solution clear increase in time there. As maintain the surplus in reader thread is leading to more burden thus increase in time as nr increases.

(b)--Worst-case time taken by the **writer threads** to enter the CS for each algorithm:Readers Writers() and Fair Readers Writers().





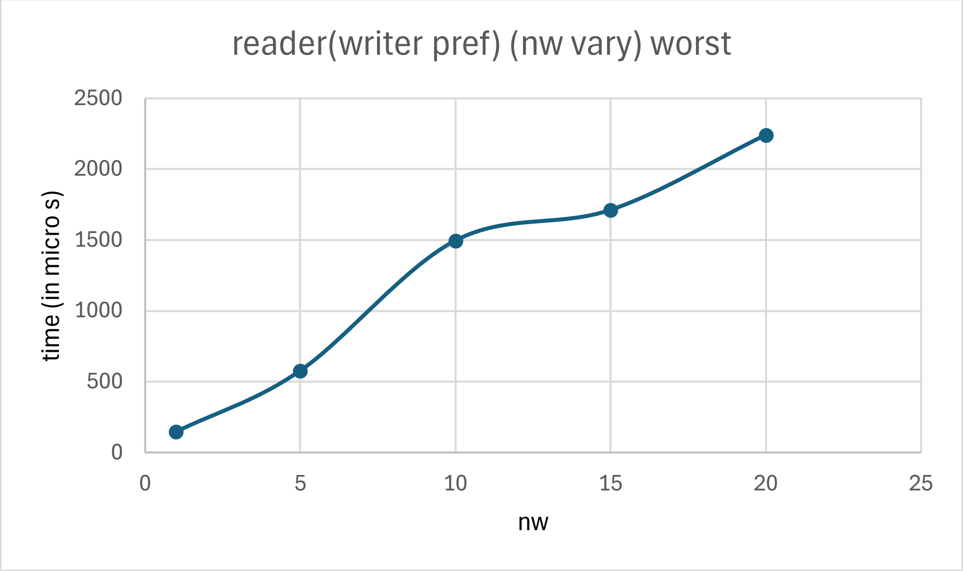
__Ovservaition & comparison__

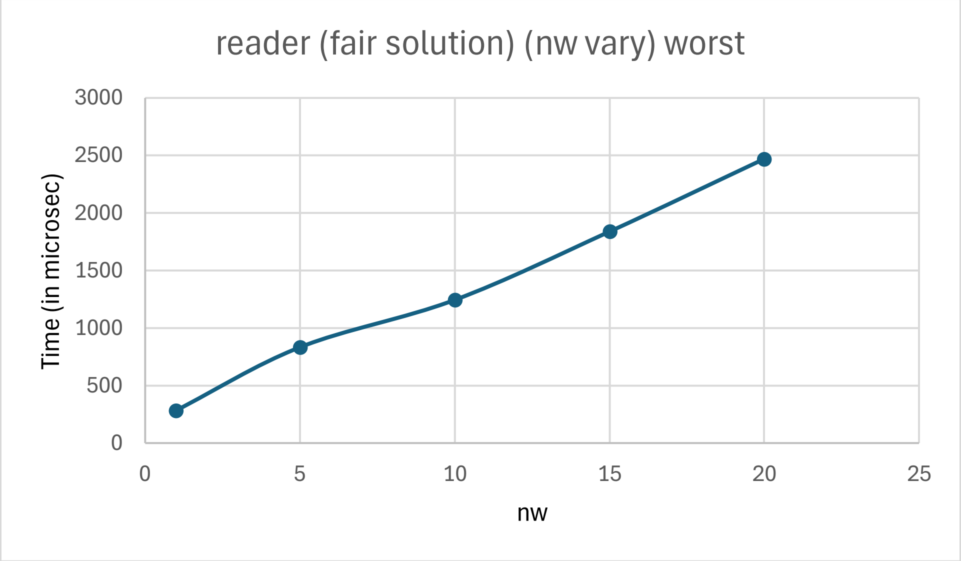
Increase in reader initially have not much affect in time for writer thread to get critical section but there is sudden increase in the time in writer preference. For this we can see how algorithms work .See in writer preference also when any reader thread has started executing it will not allow the write thread to execute when there are no reader thread. It depends on which got its first chance ,like if writer came first will allow all writer friends to execute and then give chance for reader. Thus here also when reader were less in number wirter was winning the race leading to all the writer to complete fast while increase in number of reader thread possibly given turn to reader thread leading long waiting of some writer threads thus leading to much worst time. While fair solution is working as usual and explained as same as previous corresponding argument.

Q4. Measure the worst-case (instead of average) time taken to enter the CS by reader and writer threads with a constant number of writers. Here we vary the number of reader threads nw from 1 to 20 in the increments 5 on the X-axis. All the other parameters are fixed: Number of writer threads, nr = 10, kr = kw = 10.

(a) Worst-case time taken by the **reader threads** to enter the CS for each algorithm:

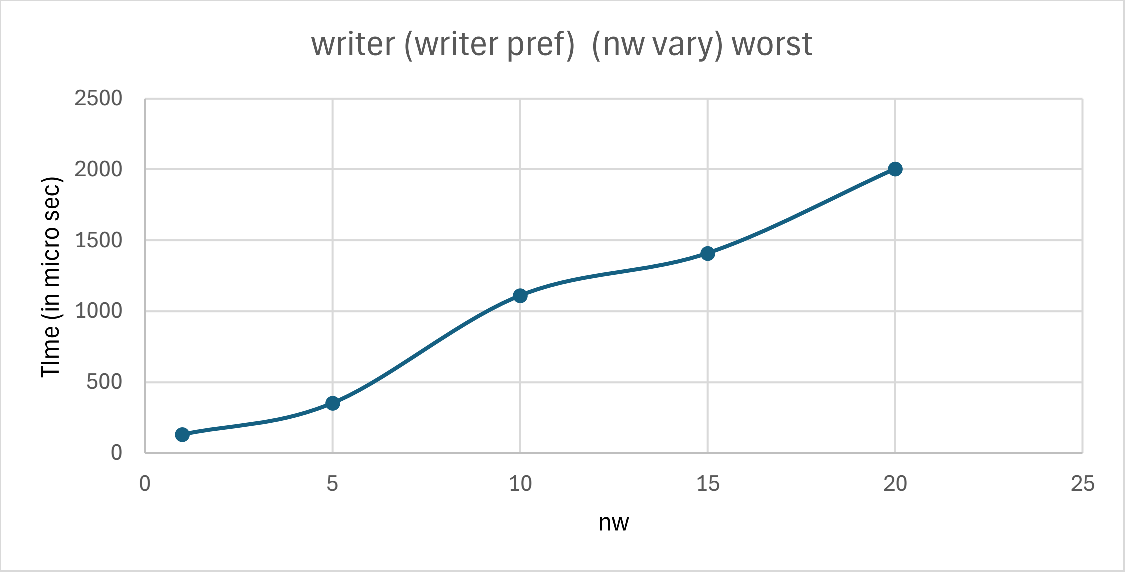
Readers Writers() and Fair Readers Writers().

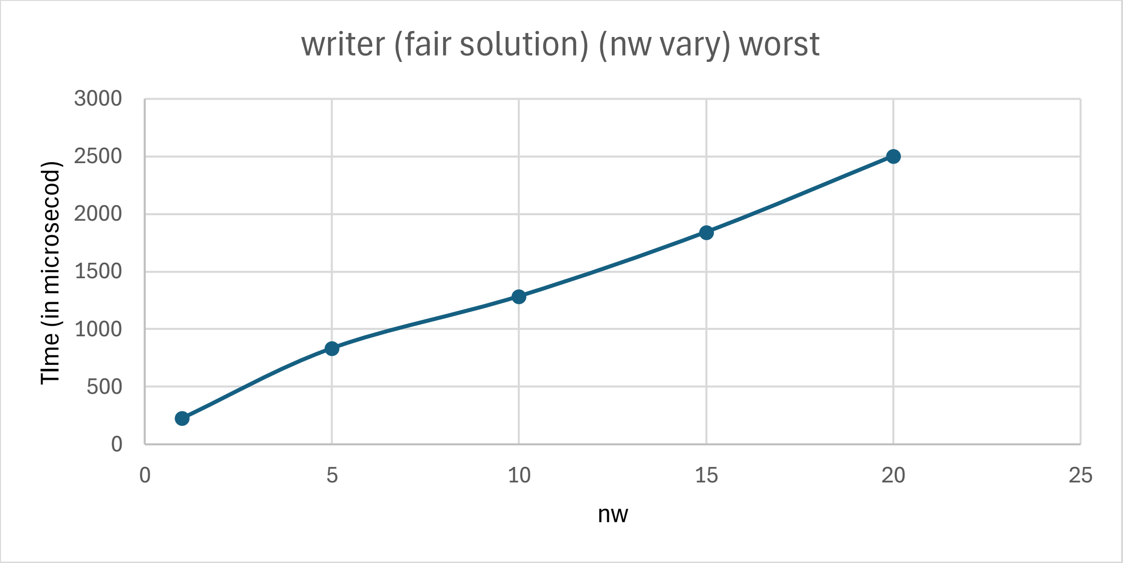




Comparison and explanation: -In worst case situation reader is performing better in write preference than fair solution. This may be due to overhead in fair solution to do these things leading to bad performance than writer preference. As nw is increasing similar effect is shown thus can be given same reason as previous.

(b)Worst-case time taken by **the writer threads** to enter the CS for each algorithm : Readers Writers () and Fair Readers Writers ().





<u>Comparison and explanation: -</u>

As expected, writer threads are performing reasonably well in worst cast timing. It can be clearly observed that reader writer (with writer preference) is at least creating 500 microsecond difference between corresponding no of nw in fair solution of reader-writer. Also increasing number of writer threads lead to compactness in the critical section leading to increase in the timing. Also, graph is quite linear means increasing the number of writer thread linearly increasing the time.

Experiment done above explain the variation by changing different factor like changing nw,nr, Checking in different algorithms. I can conclude that when there was writer preference then it was leading to better performance on an average when writer threads were changing in comparison to fair solution.

Also some observation came unexpectedly that i am not able to explain but I have tried the possibilities that can lead to this.

Also these are machine dependence like when I switched off my pc and then run the same algorithm was working very good means it is affected by the background process and thus leading to some performance measurement ambiguity. But mostly I have tried to keep only code running and closing background process.