# OPERATING SYSTEMS 2

INSTRUCTOR: DR. SATHYA PERI
Hitesh Sehrawat - ma17btech11004@iith.ac.in

## Theory

**Reader-Writer Problem : In the readers-writers problem, there is a shared memory that both reader and writer can access. Reader only reads from the memory while writer can both read and write to the memory. More than one reader can read at the same time. A writer cannot access the memory while a reader is reading. No other thread can access the memory while a writer is accessing the memory.**

**Fair Readers-Writers Problem : In the readers-writers problem, there is a shared memory that both reader and writer can access. Reader only reads from the memory while writer can both read and write to the memory. More than one reader can read at the same time. A writer cannot access the memory while a reader is reading. No other thread can access the memory while a writer is accessing the memory. When a writer requests permission, no new reader can read the memory.**

## Design of Program

### Readers-Writers Program

**In the main function, nw writer pthreads and nr reader pthreads are created. Each reader and writer thread is assigned reader and writer functions respectively. Also, all the semaphores are initialised.**

### Semaphores :

**mutex - binary: initialised to 1**

**rwmutex - binary: initialised to 1**

**avgmutex - binary: initialised to 1**

**Writer function : Before entering wait(rwmutex) i.e waits for rwmutex to grant access to the shared variable. Once given access, enter the critical section. At exit, signal(rwmutex) i.e signals rwmutex to allow other threads waiting for access.**

**As writer does not allow any other thread to enter CS therefore a standard wait and signal operation covering access can solve the purpose.**

**Reader function : Before entering, increment read count and checks if read_count is 1, if so wait(rwmutex). Put this inside mutex's wait and signal. At exit, decrement read count and if read count is zero, signal(rwmutex). Put this inside mutex's wait and signal.**

**As reader only allows other readers to enter so check if this is the first reader, if so then stop other writers from entering. If not first reader then just enter the CS. At exit, check if this is the last reader to exit only then allow writers to enter.**

**Fair Readers-Writers Program : In the main function, nw writer pthreads and nr reader pthreads are created. Each reader and writer thread is assigned reader and writer functions respectively. Also, all the semaphores are initialised.**

**Semaphores :**

**in_mutex - binary: initialised to 1**

**out_mutex - binary: initialised to 1**

**wrt_mutex - binary: initialised to 0**

**avgmutex - binary: initialised to 1**

**Writer function : Before entering wait(in_mutex) then wait(out_mutex). signal(rwmutex) i.e signals rwmutex to allow other threads waiting for access. Check if in_count == out_count then signal(out_mutex) else make wait true and signal(out_mutex) then wait(wrt_mutex) and wait is again false. At exit, signal(in_mutex).**

**As before entering writer stops new threads from entering. If there are no more readers left to read from memory, signal out_mutex else make wait true so that readers know that writer wants to enter. After that, signal out mutex so that other readers can grant writer permission. Now wait for permission. Once permission is given make wait false again. At exit simply allow other threads to enter.**

**Reader function : Before entering, increment in_count and put this inside in_mutex's wait and signal. At exit, increment out_count. If wait is true and in_count is equal to out_count then signal(wrt_mutex). Put this inside out_mutex's wait and signal.**

**As reader first increments the in_count because it's about to enter the CS. after exit, the last reader to exit CS simply checks if a writer is waiting and if it is then allow writer to enter CS.**

# Graphs

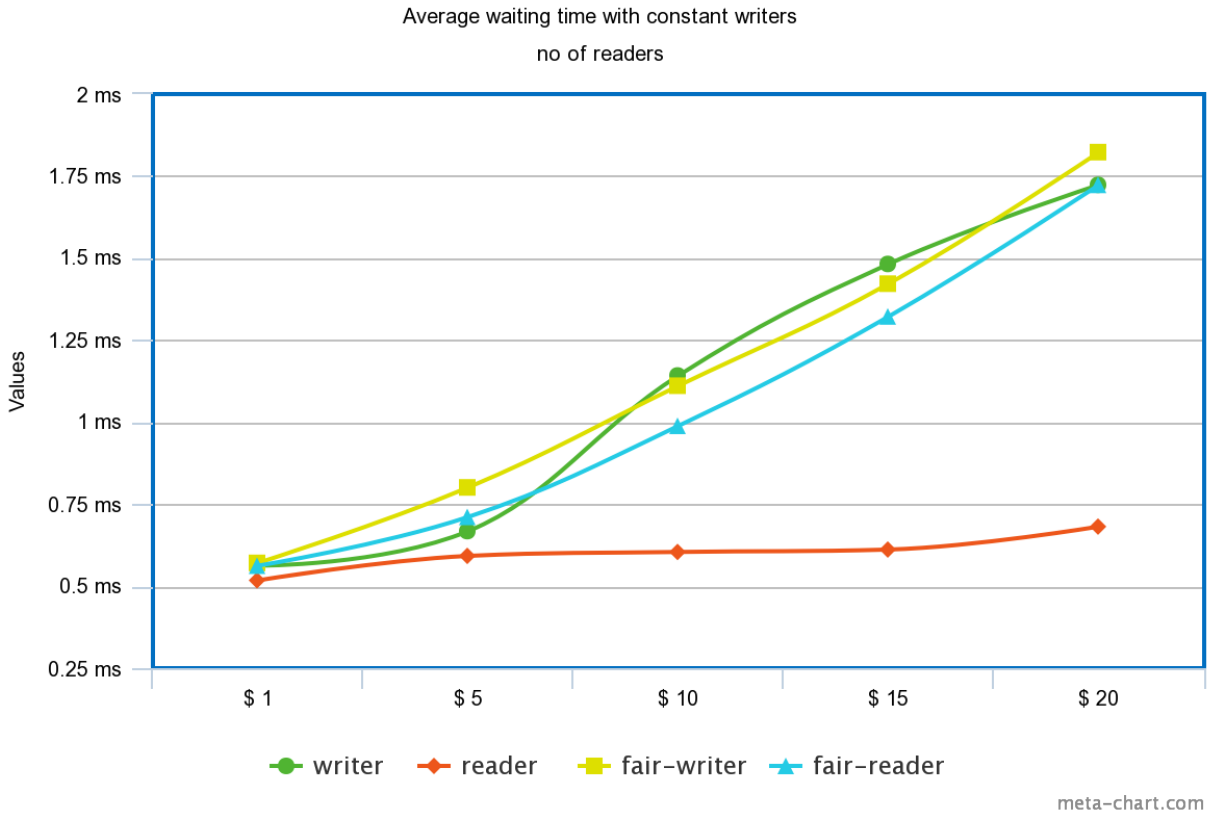
## Graph 1:

**1.Average waiting time with constant writers.**

**The Graph has 4 curves:**

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

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



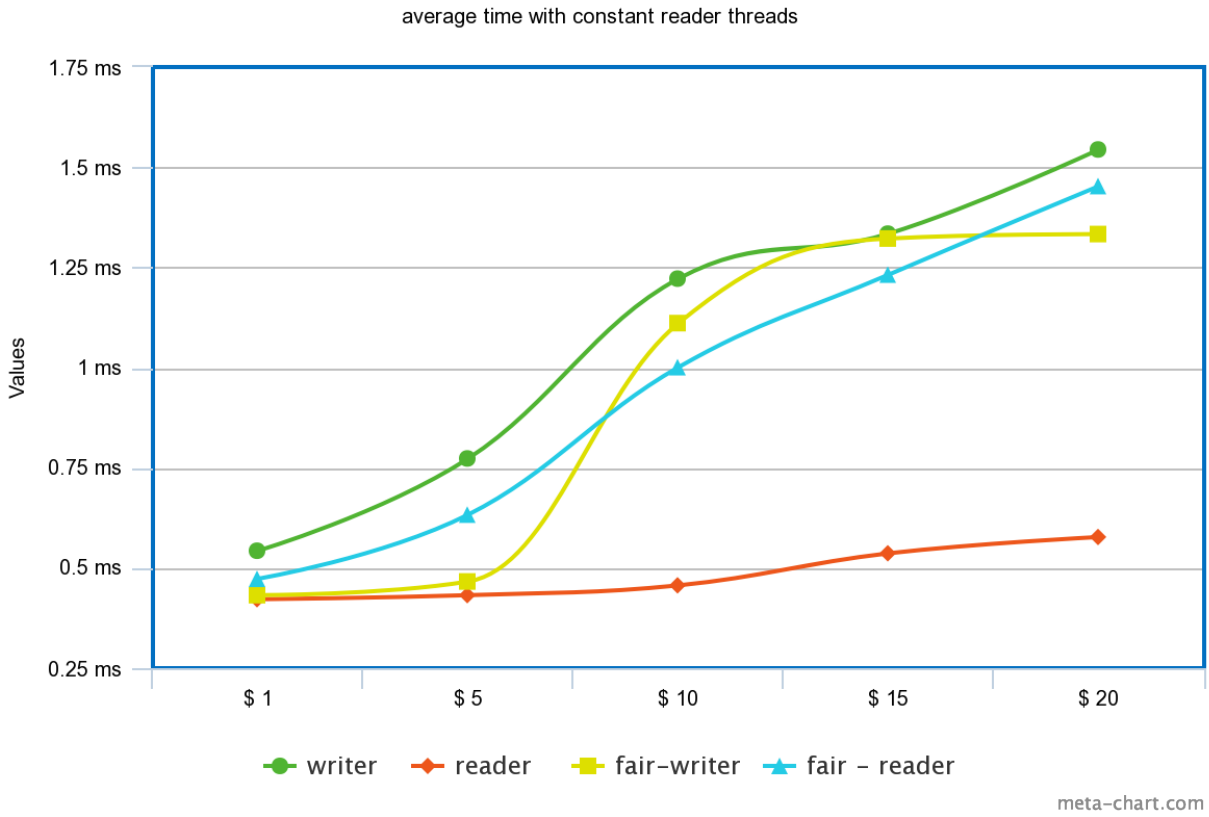

## Graph 2:

**2.Average waiting time with constant readers.**

**The Graph has 4 curves:**

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

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




## Graph 3:

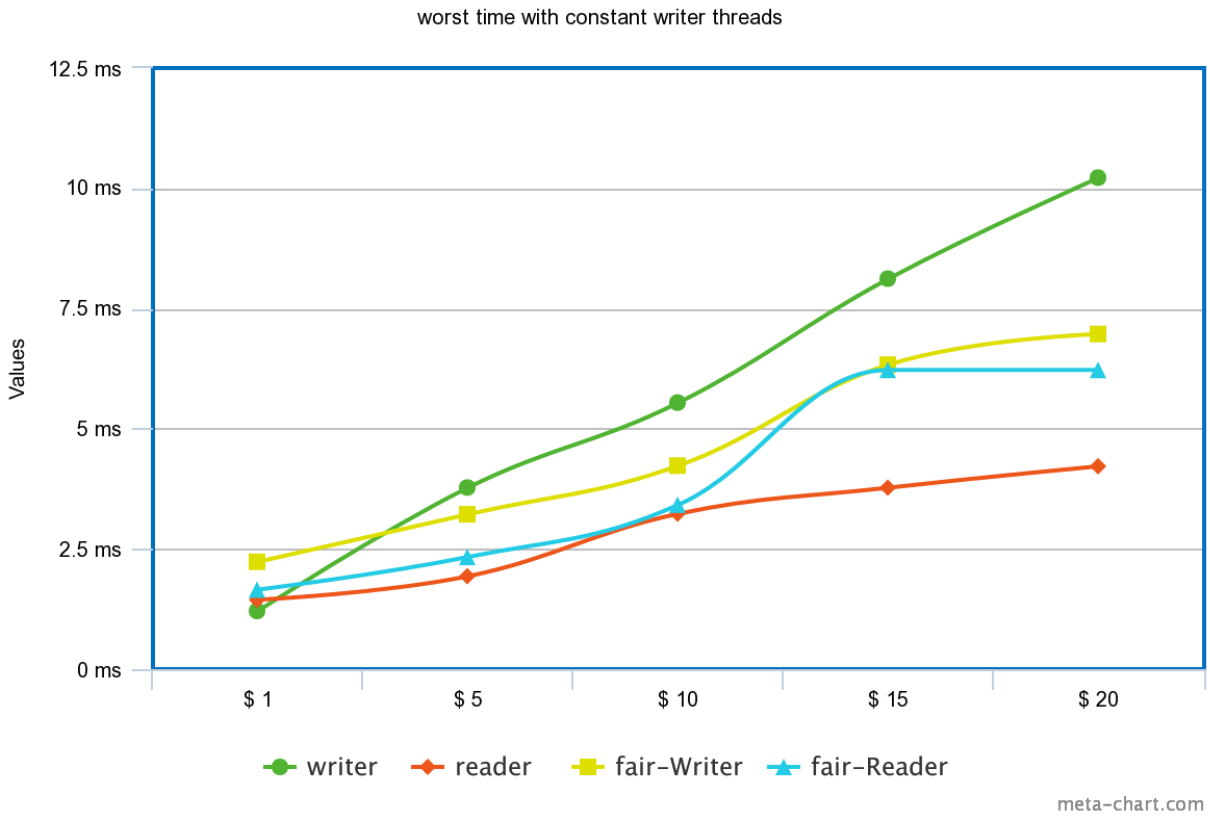
**3.Worst time with constant writers.**

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

**Readers Writers() and Fair Readers Writers().**

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

**Readers Writers() and Fair Readers Writers().**




## Graph 4:

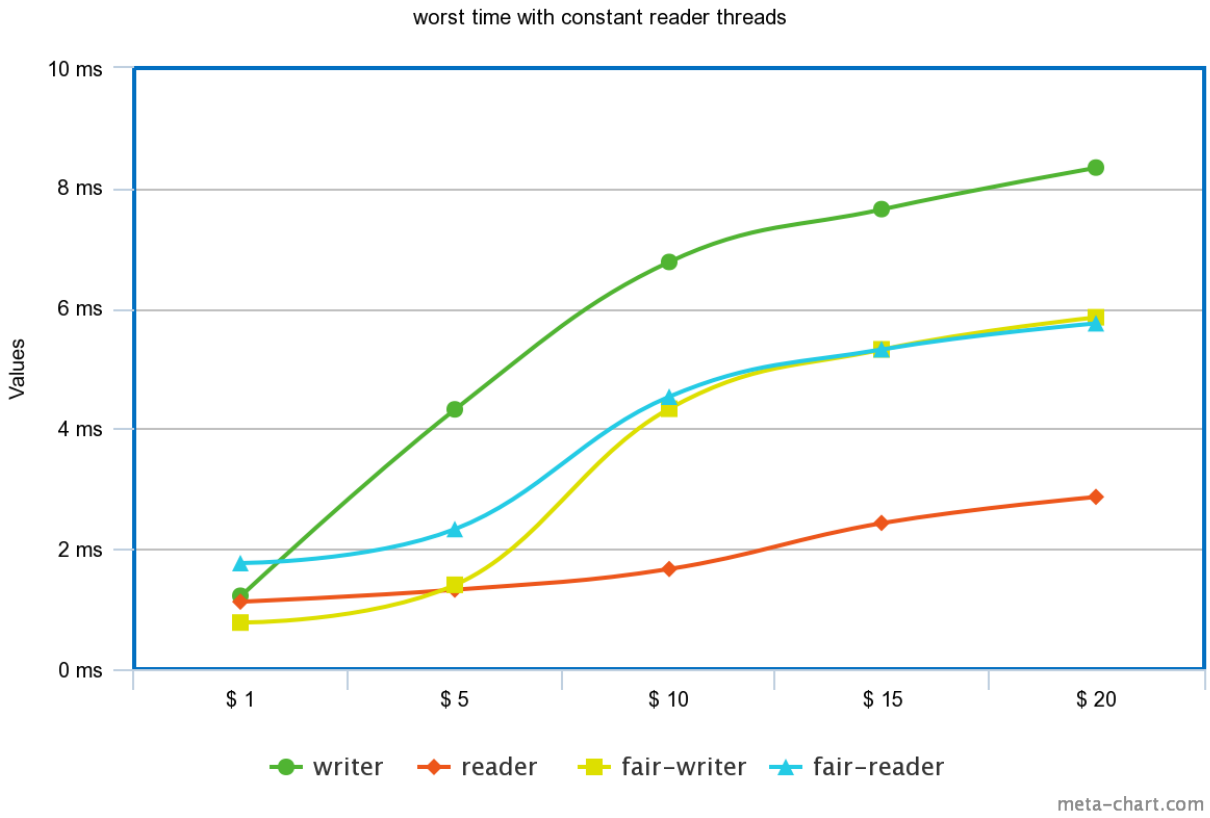
**4.Worst case with constant Readers**

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

**Readers Writers() and Fair Readers Writers().**

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

**Readers Writers() and Fair Readers Writers().**




# Conclusion

- **The average waiting for writer threads is more than average time waiting for reader, but in fair reader writer solution the gap between average time of reader and writer is less than the gap in normal reader writer solution. This is because in fair reader writer solution a reader whom request after a reader cannot get into critical section. This reader thread must wait for writer thread to complete its execution.**
- **The worst time for writer threads is more than the worst time for reader threads, because writer threads should wait for all the reader threads to complete their execution.**