Operating Systems 2
Name :- Ajinkya Bokade
Roll No :- CS17BTECH11001

Aim

To implement Readers−Writers problem and Fair Readers−Writers problem using Semaphores algorithms and compare
the average and worst−case time taken for each thread to access the critical section (shared resources).

Design of Program

The input parameters to the program will be a file, named inp−params.txt, consisting of all the parameters:− nw: the number of writer threads, nr: the number of reader threads, kw: the number of times each writer thread tries to enter the CS, kr: the number of times each reader thread tries enter the CS, µCS, µRem (delay values for CS and Remainder section that are exponentially distributed with an average of µCS, µRem milli−seconds.)
In Reader−Writers problem, reader threads are given more priority as comapred to writer threads which may result in writer thread starvation.
While in Fair Reader−writers problem, this problem is solved. As a result there is no starvation.
Both implementations are based on semaphores.
Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.
We are using pthread semaphores.
Pthread Semaphores support two operations:−
1) int sem_wait(sem_t *sem) :− decrements (locks) the semaphore pointed to by *sem*. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement (i.e., the semaphore valuerises above zero), or a signal handler interrupts the call.

2) int sem_post(sem_t *sem) :− increments (unlocks) the semaphore pointed to by *sem*. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait() call will be woken up and proceed to lock the semaphore.

Implementation of Program

The nw number of writer and nr number of reader threads are created using pthreads.
In Reader Writers implementation, we are using two binary semaphores namely rw_mutex and mutex initialized to 1. mutex is used in reader's code to provide mutual exclusion to shared variable read_count among other reader threads. rw_mutex is used in both writer's and reader's code so that at a moment either writer thread can access shared data or multiple readers can access shared data. Hence before accessing shared data, writer and first reader thread has to acquire rw_mutex semaphore. If first reader thread has acquired it, then other reader threads can also access shared data at the same time. Hence it leads to writer threads starvation since it gives higher priority to reader threads.
In Fair Reader Writers implementation, starvation problem is solved by introducing another binary semaphore order_mutex initialized to 1. In this, if a writer thread requests to access CS, then it calls sem_wait(&order_mutex), as a result further reader threads which are trying to enter CS afterwards are blocked by this semaphore and have to wait until writer thread completes its access. As a result, there is no starvation and its fair.

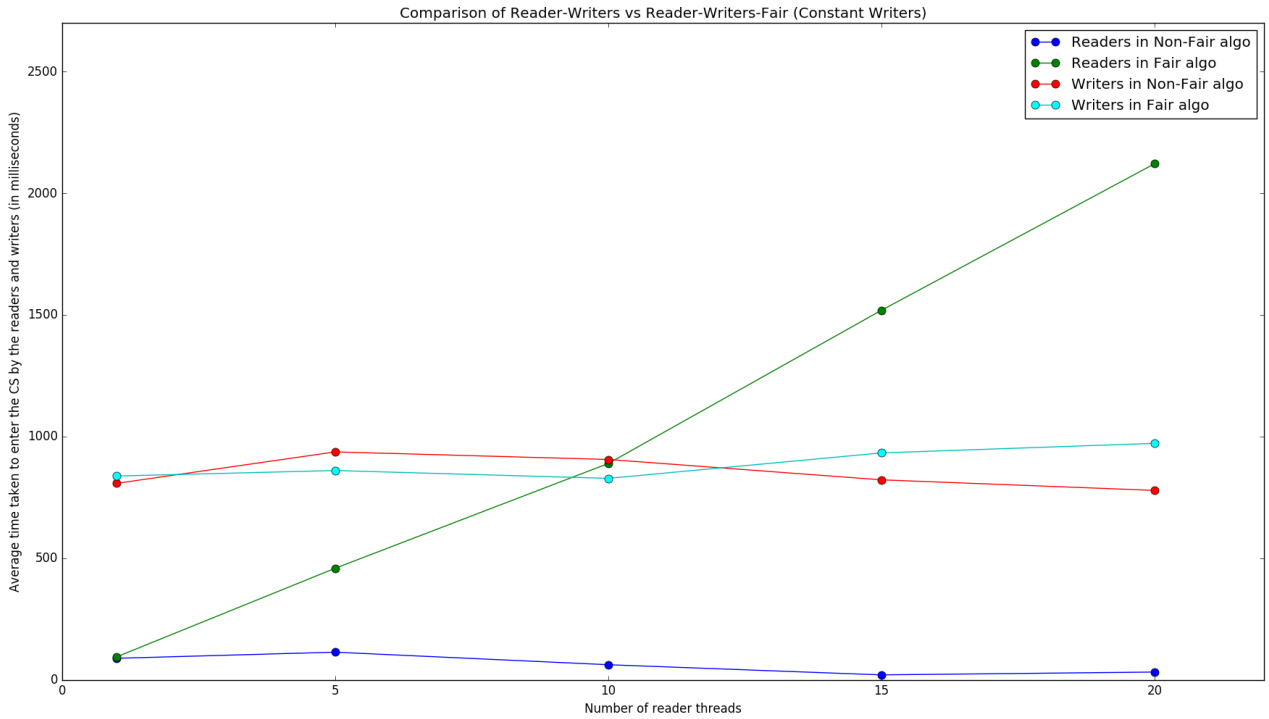
Comparison of the performance of producer and consumer threads

Input parameters :− Count of writer threads (kw) = 10
Count of reader threads (kr) = 10
μCS = 10
μRem =10
These parameters are kept constant throughout.

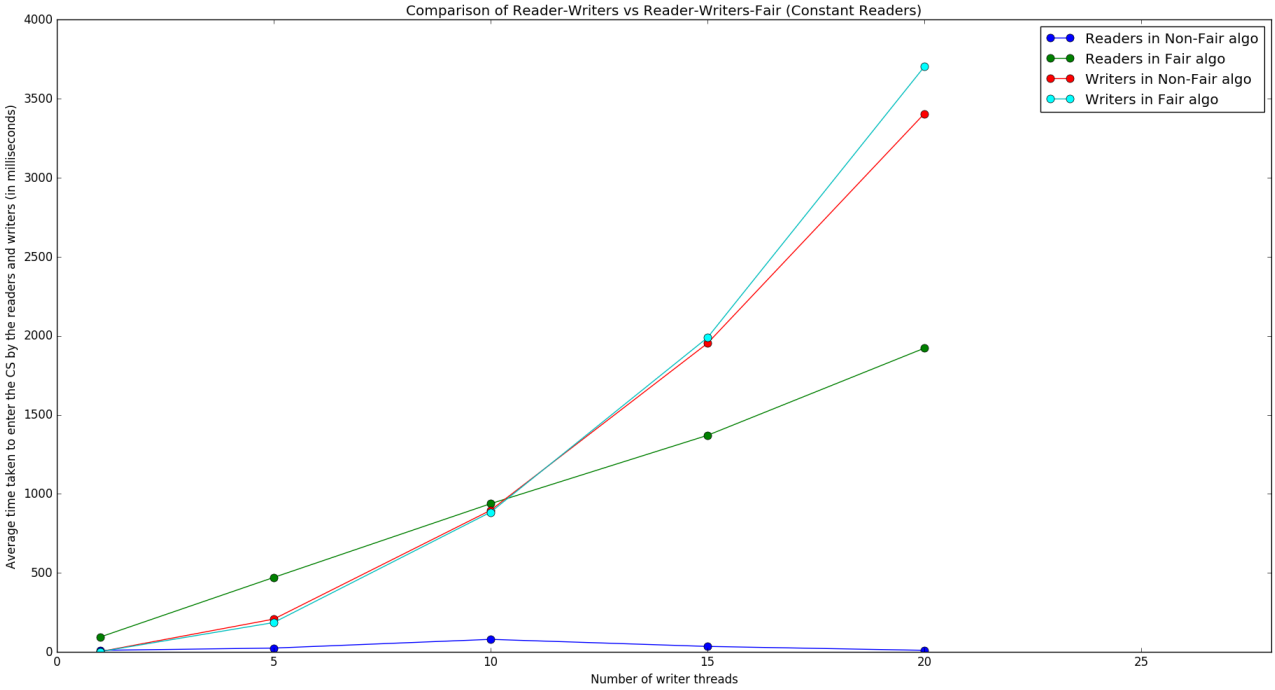
In first case, number of writer (nw) is kept constant at value 10 and number of reader threads (nr) is varied from 1 to 20 in increments of 5.
In second case, number of reader (nr) is kept constant at value 10 and number of writer threads (nr) is varied from 1 to 20 in increments of 5.

Graphs

1) Graph of Average waiting time of readers and writers vs Number of reader threads (writer threads kept constant at 10)

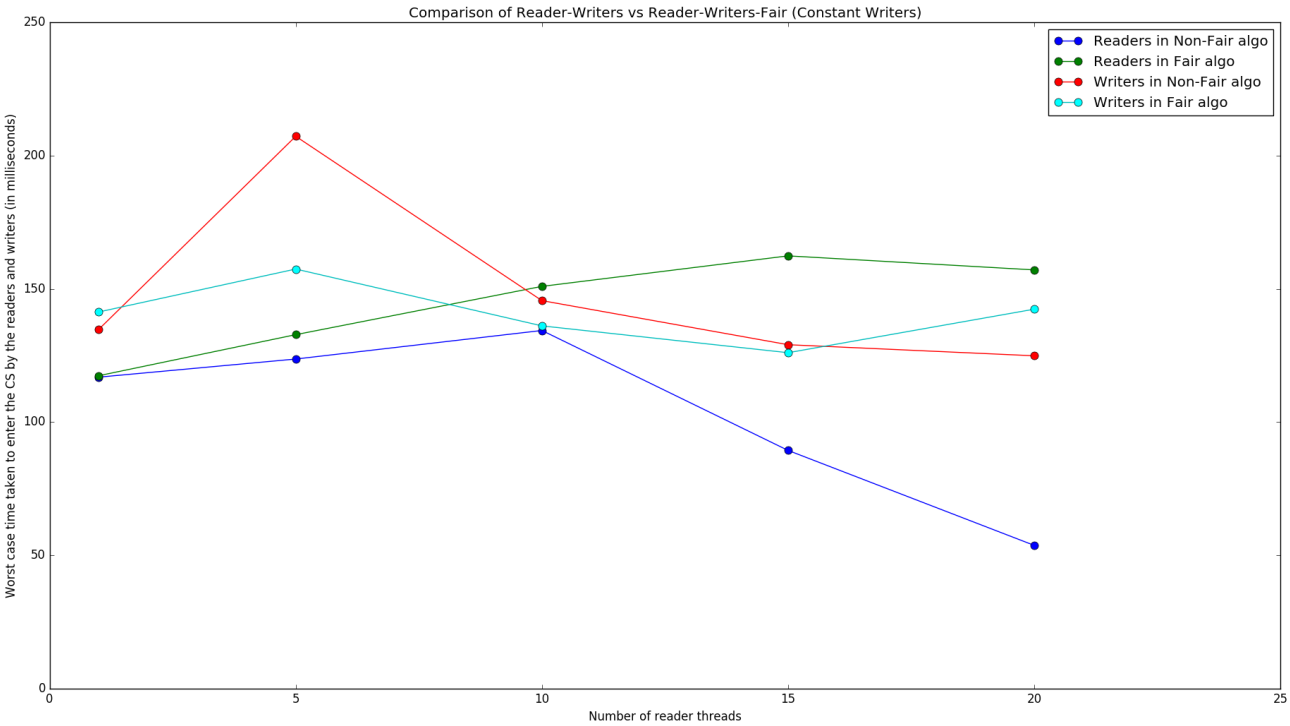


2) Graph of Average waiting time of readers and writers vs

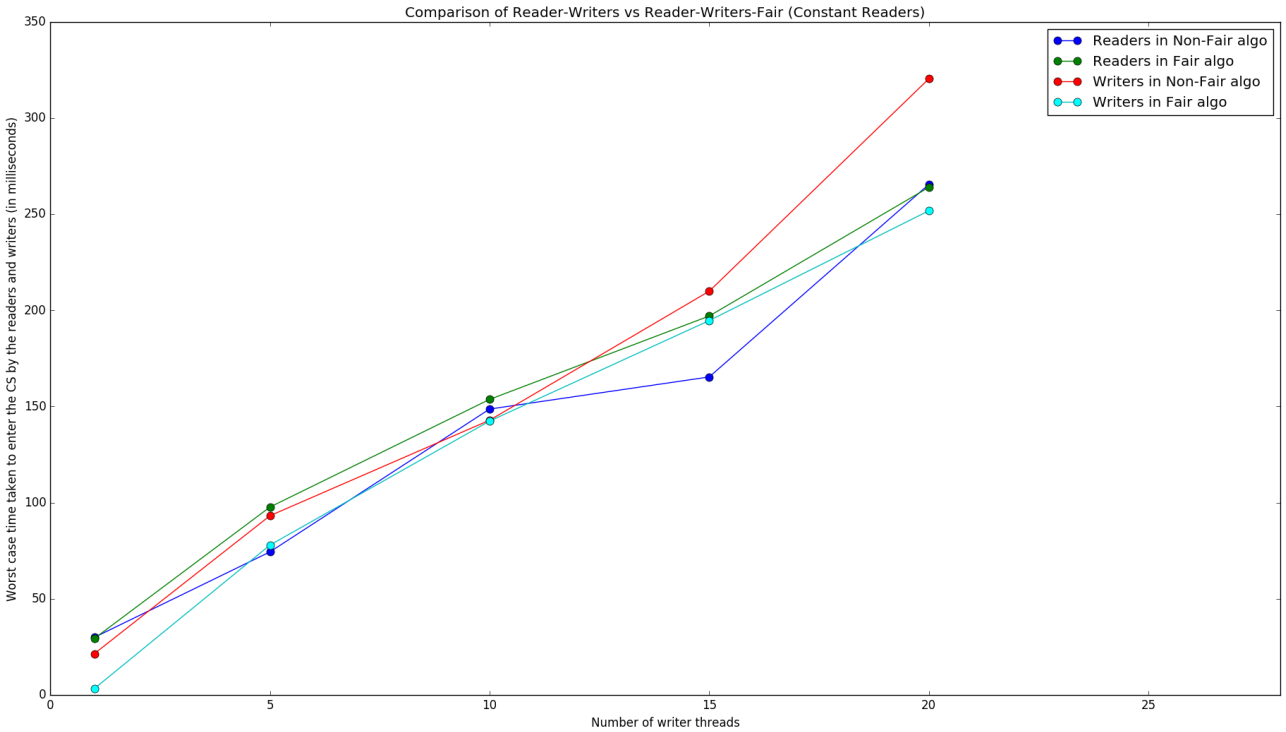


Number of writer threads (reader threads kept constant at 10)

3) Graph of Worst case waiting time of readers and writers vs Number of reader threads (writer threads kept constant at 10)



4) Graph of Worst case waiting time of readers and writers vs Number of writer threads (reader threads kept constant at 10)



Conclusion

From first graph, where writer threads are kept constant and writer threads are varing, we observe that average waiting for reader threads is significantly less in case of Reader−Writers algo than in case of Fair Reader−Writers algo. It is true since in Reader−Writers, preference is given to reader−threads as compared to writer threads. Also, average waiting for writer threads is almost same and doesnt increase significantly in both cases.

From second graph, where reader threads are kept constant, its similar to first graph except that average waiting time for writer threads here increases as expected since writer threads are also increasing.

From third graph, where writer threads are kept constant, worst case waiting time is less for both readers and writers in case of Fair Readers−Writers as compared to Readers−Writers since Fair−Readers−Writers give equal preference to both readers as well as writer threads.

From fourth graph, where reader threads are kept constant, here also worst case waiting time is less for both readers and writers

in case of Fair Readers−Writers as compared to Readers−Writers since Fair−Readers−Writers give equal preference to both readers as well as writer threads.