CS3523 OPERATING SYSTEMS-2:

REPORT :

Design of the Program:

Simple Reader-Writer Algorithm:

- The global varaibles named n_w , n_ r , k_w , k_r denotes the number of writer threads , number of reader threads , no of times a writer thread enters the CS , no of times a reader thread enters the CS respectively.
- In int main( ) the code accepts the input from the input file named “inp-params.txt” and stores those values in the global variables.
- writer_tid , reader_tid are the arrays for storing the thread id’s of writer , reader threads respectively.
- Two semaphores, mutex & rw_mutex are used in the program. Mutex , rw_mutex are intiliazed to 1.
- The mutex semaphore is for handling the increment , decrement of the number of readers i.e. readers count .
- The rw_mutex semaphore is used by both readers , writers & it is used for ensuring the mutual exclusion in the CS.
- All writer threads access the writer function & All reader threads access the reader function.

Writer’s function :

- The passed parameter is stored in a local variable thread_id.cs_time & rem_time are the exponential distributions with an average of μ_cs & μ_rem respectively.
- All the writer threads who requests to enter CS, enters the request section but only one acquires the semaphore rw_mutex & then it performs writing in the CS and then it releases semaphore so that others can access it.
- **Note:** Here I have kept the request , entry , exit print statements before the signaling the semaphore rw_mutex. Since keeping normally as given in psuedo code will result in race conditions between 2 writer threads then the lines won’t be printed correctly .

Reader's function:

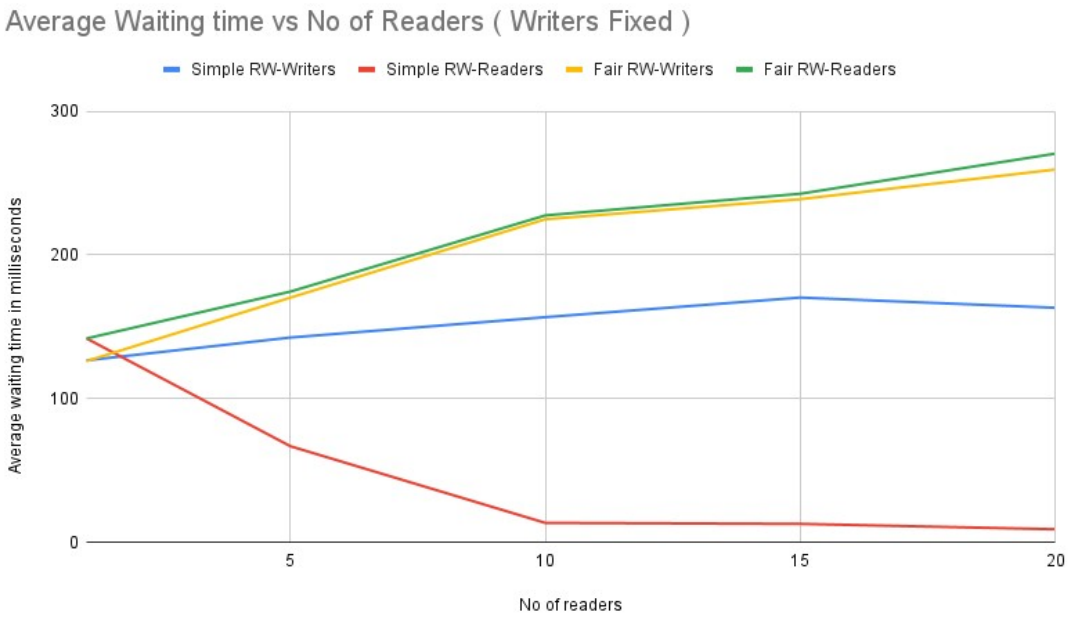
- Storing the parameter & exponential distributions are same as in the writers function .
- All the readers who requests to enter the CS , enters the request section. A reader who wants to enter CS will acquire the semaphore mutex & then the program updates the count of readers i.e. read_count++.
- If read_count=1 then the resource will be allocated to readers i.e. when a first reader arrives the CS then the program keeps the resource available to readers which is done by acquring semaphore rw_mutex. The thread which acquired semaphore mutex releases it ensuring others can access it .
- After completion of reading by a thread the program will decrement the readers count i.e. read_count-- & it is handled by semaphore mutex . So, Every reader thread who finished reading must acquire the semaphore mutex & then the program decrements the readers count .
- If read_count=0 then the resource will be freed by the readers i.e. when the last reader completed his reading then the program releases resource from the readers ensuring it can be accessable to others which can be done by releasing semaphore rw_mutex .The thread which acquired semaphore mutex releases it .
- Here in each reader's and writer's function . We caluclate the waiting times of each thread by taking the difference between their request time & entry time . Then the program caluclates average , maximum waiting times for all writers , readers and then it ouputs the values to Average_time.txt file from the main thread.
- Here in this code reader is given more priority than writer.
- **Note:** Here I kept the request, entry print statements just after incrementing the readers count such that at a time only one thread can do the printing to the ouput file & I kept the exit print statement just after the decrementing the readers count such that at a time only one thread can do the printing . If we keep print statements according to psuedo code we will get race conditions between writer thread while printing to the output file & then the lines won't be printed correctly.All the print statements will be printed in RW-log.txt file.

Fair Readers-Writers Algorithm:

- Global variables & accessing the input file is same as Simple RW.
- The difference here is we use three semaphores instead of two naming mutex , rw_mutex , queue . All the semaphores are intialized to 1 .
- Working of mutex , rw_mutex is same as previous & semaphore queue is the one which handles the servicequeue of the threads i.e. it preserves the order of requests .
- Writer's & Reader's function are same as previous except that every thread which try to enter the CS will be added to service queue maintained by semaphore queue & this queue will follow FIFO. This ensures no starvation & there are no priorities of one over the other .

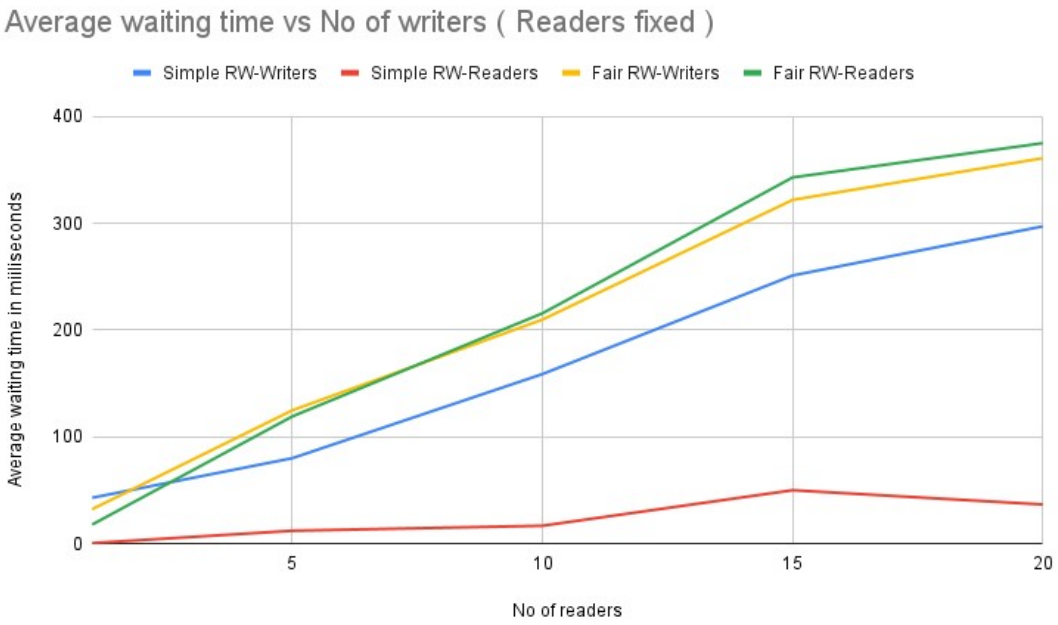
Comparision between RW & FRW Algorithms :

- The below graph i.e. Graph-1 is the comparision between RW & FRW algorithms by caluclating average waiting times of writer , reader threads with a constant no of writers & varying the no of readers . Input values taken for the graph are n_w=k_r=k_w=10 & μ_cs= 15 , μ_rem=25.



Graph-1

- The below graph i.e Graph-2 is the comparision between RW & FRW algorithms by caluclating average waiting times of writer , reader threads with a constant no of readers & varying the no of writers . Input values taken for the graph are n_r=k_w_=k_r=10 & μ_cs= 15 , μ_rem=25.

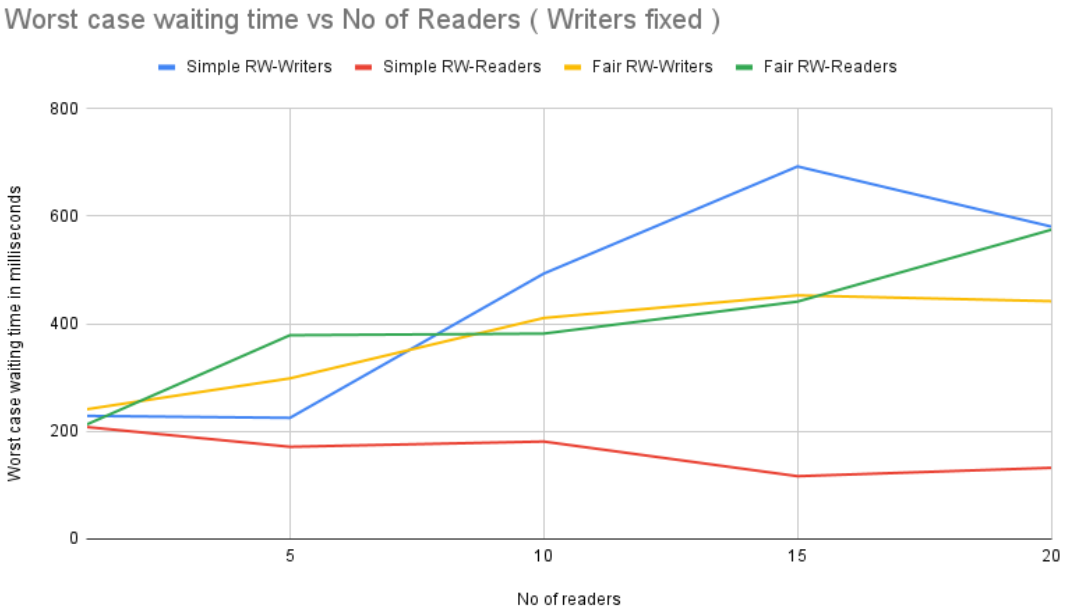


Graph-2

Analysis from Graph-1 & Graph-2 :

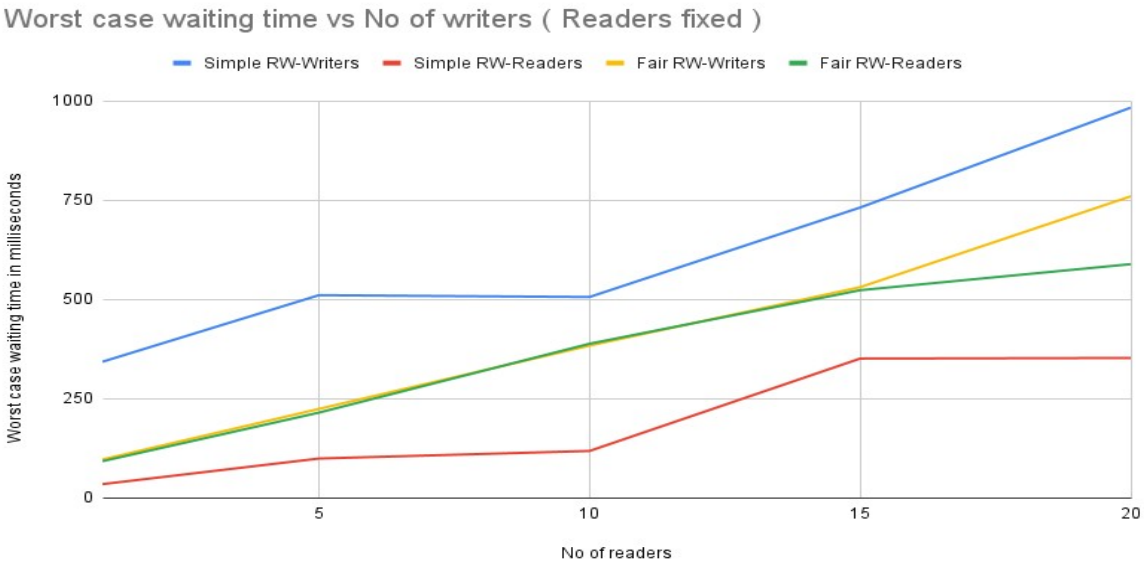
- From the graphs Graph-1 , Graph-2 we can say that FRW algorithm results in high average waiting times than that of RW algorithm . Since FRW ensures fairness without no priorities i.e. all the threads have same priority which results in having high average waiting time.
- Also we can see that the average waiting times of RW-Readers are very less when compared to that of RW-Writers . Since in RW algorithm readers are given high priority than that of the writers which results in high average waiting time of writers & very low average waiting time of readers.
- When we see the average waiting times of FRW-Readers , FRW-writers , there isn’t much significant difference between them .Since in FRW algorithm all the threads have priority.

- The below is the graph i.e. Graph-3 is the comparsion between RW & FRW algorithms by caluclating worst case waiting times of reader , writer threads with a constant no of writers & varying the no of readers .Input values taken for the graph are n_w=k_r=k_w=10 & μ_cs = 15 , μ_rem = 25.



Graph-3

- The below graph i.e. Graph-4 is the comparision between RW & FRW Algorithms by caluclating the worst case waiting times of reader , writer threads with a constant no of readers & varying the no of writers .Input values taken for the graph are n_r=k_w=k_r=10 & μ_cs = 15 , μ_rem = 25.



Graph-4

Analysis from Graph-3 & Graph-4 :

- From the graphs i.e. Graph-3 & Graph-4 we can say that worst case waiting time of writers is less in FRW algorithm than in RW algorithm. Since in RW algorithm writer threads are starving it results in high worst case waiting time of writers in RW algorithm.Where as in FRW all the threads are equal so the writer threads will not be starving.
- We can also say that the worst case running time of readers is low RW algorithm than in FRW algorithm . Since readers are given more priority in RW algorithm .So, reader threads will not starve in RW algorithm . Hence they have less worst case waiting time in RW algorithm.Where as in FRW all the threads have same priority so the readers lost their advantage of having high priority .

Conclusion:

- FRW algorithm results in increase of average waiting time but ensures that no thread is starving.
- When we change from RW to FRW, the worst case waiting times will decrease in writers & increase in readers .

Submitted by

Kodavanti Rama Sravanth

CS20BTECH11027