ASSIGN4: Implement solutions to Readers-Writers (writer preference) and Fair Readers-Writers problems using Semaphores

Gaurav Choudekar - CS22BTECH11015

March 18, 2024

1 Introduction

This report is regarding programming assignment-4 conducted on different algorithms as solutions to reader writer problem.

2 About Code

The code is written in C++.required parameters are given from file as input and log file is obtained as output.average time reader and writer thread has to wait to access critical section in both techniques is obtained in output file.

3 Input format

Number of reader and writer threads,number of times each access critical section and computation time in critical section and after exiting critical section is given as input from file.

4 Code Flow

4.1 Main Function

The main() function initiates the execution of two distinct variations of the reader-writer problem: one favoring writers and the other ensuring fairness between readers and writers. It begins by recording the program's start time and initializing the necessary semaphores and variables to manage thread synchronization. In the first phase, it creates multiple reader and writer threads for the writer-preference version, allowing them to access the critical section according to predefined rules. After ensuring all threads complete their tasks, it repeats the process for the fair version, setting up semaphores to enforce fairness. Once again, it waits for all threads to finish execution. Subsequently, the function calculates average access times to the critical section for readers and writers in both versions and determines the worst-case entry times. Finally, it outputs these results to a file named "Average time.txt" and displays the worst-case entry times for readers and writers of each version. Upon completing these tasks, the function gracefully exits, signaling the successful execution of the program.

4.2 Other Functions

The program utilizes functions such as reader, writer, readerfair, and writerfair to manage the behavior of threads in different versions of the reader-writer problem.reader and writer are used in writer preference reader writer problem while readerfair and writerfair are used in fair reader writer problem.as name suggests,when threads are created,it starts to execute from here depending on which type of thread,reader or writer,it is.

4.3 global variables used

- int nw, nr, kw, kr: These variables represent the number of writer threads (nw), reader threads (nr), iterations for writer threads (kw), and iterations for reader threads (kr). They are used to control the number of threads and iterations in both versions of the reader-writer problem.
- int ww, onr;: ww tracks the number of active writers, while onr tracks the number of active readers. They are used for synchronization purposes to ensure mutual exclusion between readers and writers in the writer preference version.
- semaphores writechange, sread, swrite, readchange, printing;: These semaphores are used for synchronization between threads. writechange: Controls access to the writing process. sread: Ensures mutual exclusion among readers. swrite: Ensures mutual exclusion among writers. readchange: Synchronizes changes to the reader count. printing: Synchronizes output to the output files, ensuring thread-safe writing.
- ofstream output1("RWlog.output"); and ofstream output2("fairRWlog.output");: These ofstream objects are used to write output logs to files for the writer preference and fair versions of the reader-writer problem, respectively.
- double reader1, writer1, reader2, writer2, maxr1, maxw1, maxr2, maxw2;: These variables store statistics related to time measurements for readers and writers in both versions of the problem, such as average access times and worst-case entry times.
- average1,average2,lambda1,lambda2,dist1,dist2:set up for random exponential function.
- programstart1,programstart2 declare starting points of 2 techniques.

4.4 reader,writer,readerfair,writerfair

These functions are functions where created threads starts to work from.reader thread from writer preference reader writer problem starts from reader while writer starts from writer.reader thread from fair reader writer problem calls function readerfair while writer calls and starts from writer fair.

- variables used
  - pthreadt *thread: Pointer to a pthreadt structure that represents the thread.
  - pthreadt id: Variable to store the identifier of the current thread.
  - temp:temporary store time to access critical section after request.
  - time elapsed:time required to access critical section after request.
- Short introduction to techniques
  - writer preference reader writer problem:In the writer preference reader-writer problem, priority is given to writer threads over reader threads. This means that when a writer thread requests access to the critical section, no new reader threads are allowed to enter until the writer has finished writing. This approach ensures that writers are not starved and that the shared resource is not read while it is being modified.
  - Fair reader writer problem:In the fair reader-writer problem, fairness is ensured between reader and writer threads. This means that both readers and writers are given equal opportunities to access the critical section. Fairness is typically achieved by implementing a solution that prevents starvation of either readers or writers. This approach ensures that neither readers nor writers are indefinitely blocked from accessing the critical section, thereby improving overall system fairness and performance.

4.5 reader and writer

Each reader thread from writer preference reader writer problem starts its execution for problem from reader function and writer from writer function.it stores its thread id in id variable.each reader thread loops and acceses critical section for kr number of times while writer thread kw number of times.each time a thread is printing something about its status of execution,which is request,entry or exit from

critical section,it aquires printing semaphore.reader thread will access its critical section only if there is no writer thread in waiting or in critical section.every reader thread accesing critical section and exiting it updates onr variable,first reader thread entering critical section calls sem wait to not let writer thread to enter critical section directly,and last exiting reader thread calls sem post to let writer thread enter critical section.writer thread aquires sread semaphore which will not allow future requesting reader threads to access its critical section unless no writer thread is in its critical section or in waiting.variable ww helps to track number of waiting and executing writer threads.if a writer thread is first writer thread then only it aquires sread semaphore.swrite makes sure that there is only one writer thread accessing its critical section,a thread aquires swrite while entering in critical section,finally,if there is no waiting thread,before exiting critical section a writer thread calls sem post to wake up all sleeping reading threads which were waiting for writer thread completing their task,this is done by calling sem post for sread that many times.if a writer thread request for critical section,then it waits till all reads which are currently in critical section to exit it,it wakes up onlyy when onr becomes zero..

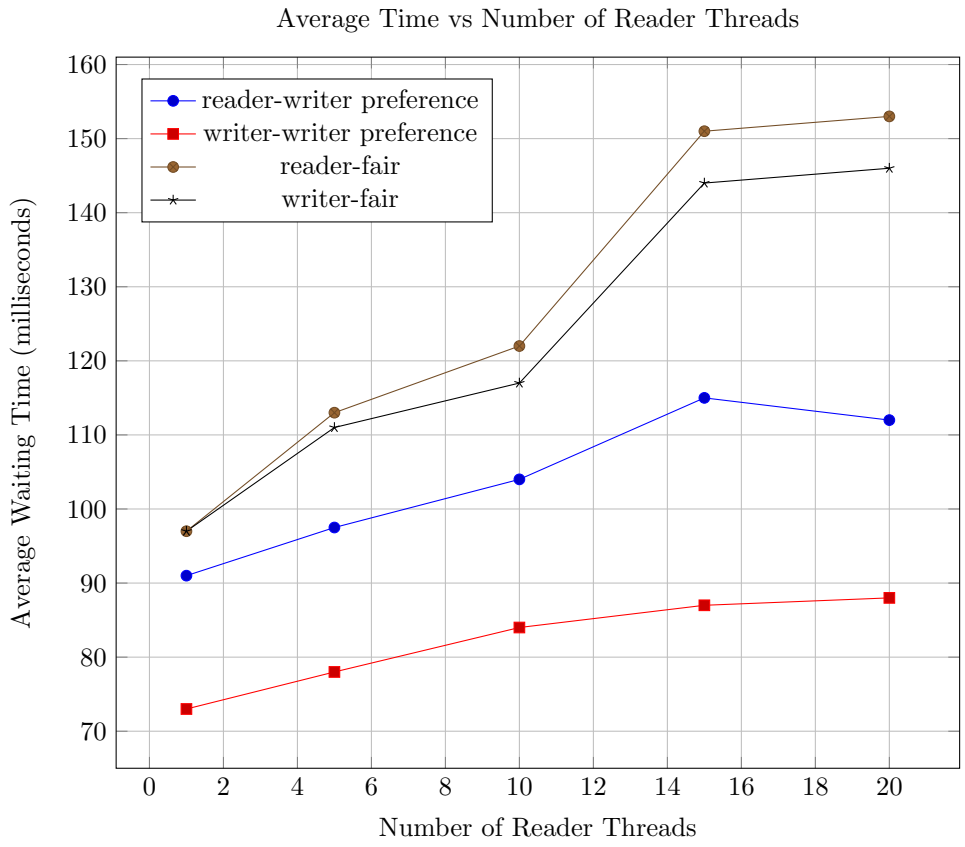
4.6 readerfair and writerfair

Both reader and writer threads use the common semaphore to ensure mutual exclusion, allowing only one type of thread to access shared variables at a time.reader can get access only if no writer is in its critical section or requested before it,else than previously requested writer would acquire common semaphore before it and it has to wait for it to complete its critical section access.fwriter ensures that there is only one writer in critical section,every time writer enters critical section,it aquires it and releases only while exiting critical section.when there is no working reader thread in critical section,that is a reader thread is first one to access critical section,then it aquires fwriter,to not let any writer to access critical section,last exiting reader thread releases it to make sure that writer thread can access its critical section and there is no working reader thread in critical section currently.common is not acquired throughout the critical access as we want to allow multiple reader threads to access critical section simultaneously.

4.7 Some important points

- while updating or checking values corresponding to current status about number of working threads in critical section or printing in output file,semaphore is used to ensure mutual exclusion.
- usleep function is used to simulate that thread is performing some work.exponential random number is given as input to it.
- time mentioned in log file is respect to start of that algorithm
- worst case time is compared every time for each access to update it if needed.
- time to access critical section is summed after every access so that average could be found after completion of algorithm.

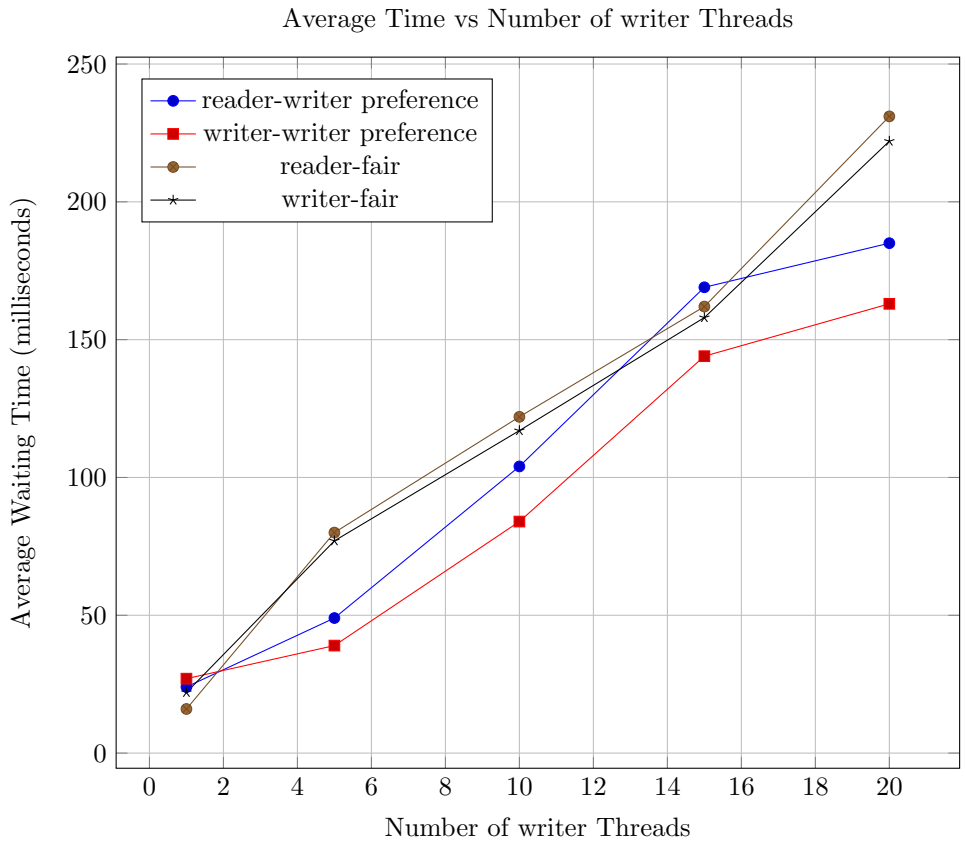
5 Graph



Number of Reader Threads	reader-wp	writer-wp	reader-fair	writer-fair
1	91	73	97	97
5	97.5	78	113	111
10	104	84	122	117
15	115	87	151	144
20	112	88	153	146

Table 1: Average Waiting Time (milliseconds) for Different Numbers of Reader Threads

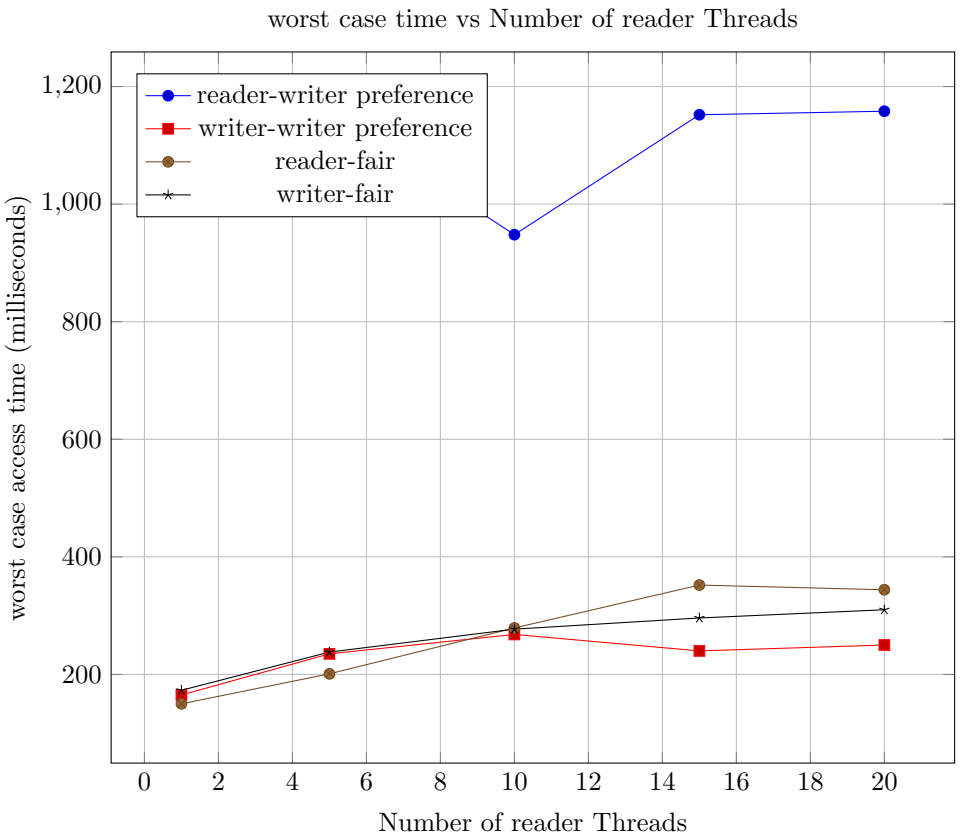
6 Graph



Number of Writer Threads	reader-writer preference	writer-writer preference	reader-fair	writer-f
1	24	27	16	22
5	49	39	80	77
10	104	84	122	117
15	169	144	162	158
20	185	163	231	222

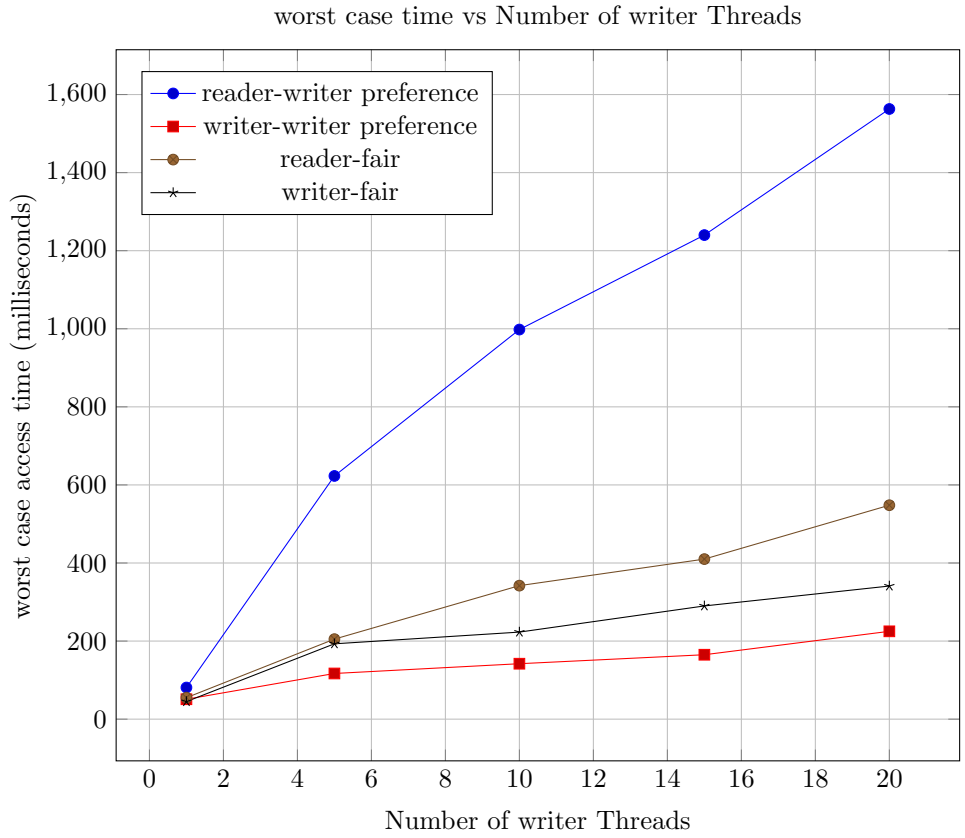
Table 2: Average Waiting Time (milliseconds) for Different Numbers of Writer Threads

7 Graph



Number of Writer Threads	reader-writer preference	writer-writer preference	reader-fair	writer-f
1	1076	165	150	173
5	1155	235	201	238
10	948	268	279	277
15	1152	240	352	296
20	1158	250	344	310

Table 3: Worst Case Access Time (milliseconds) for Different Numbers of reader Threads



Number of Writer Threads	reader-writer preference	writer-writer preference	reader-fair	writer-f
1	81	51	55	45
5	623	117	205	193
10	998	142	342	223
15	1240	165	410	290
20	1563	225	548	341

Table 4: Worst Case Access Time (milliseconds) for Different Numbers of Writer Threads

8 output files

- log output for writer preference reader writer problem is obtained in "RWlog.output"
- log output for fair reader writer problem is obatined in"fairRWlog.output"
- average time by threads is obtained is "Average time.txt"

9 Results and analysis

9.1 Experiment 1:Avg time vs number of readers

- overall,average time increases with increase of number of reader threads.
- algorithm with writer preference has less growth compared to fair preference algorithm
- in particular algorithm,reader has more average access time compared to writer thread.
- in fair preference,we have high average time as threads have to wait longer to get access to critical section.

- in writer preference,writer has to wait for more number of reader threads to exit critical section to access critical section,hence average time is seen to increase.
- in writer preference,more readers have to wait as preference is given to writer,hence their average access time increases.

9.2 Experiment 2:Avg time vs number of writers

- Overall, there is increase in average access time to critical section.
- as preference is given to writers in writers preference,it has lower growth and average access time.readers has to wait for more writers to complete their access after request,hence it has higher average than writers.
- generally fair algorithm has higher average access time as we are not giving preference to writers,and also due to this,chunks of readers gets fragmented.
- compared to writer in writer preference,average waiting time for fair is higher as preference is not given to it.
- average waiting time for both readers and writers is almost same/similar in fair solution.

9.3 Experiment 3:Worst case time vs Number of readers

- there is exceptional difference in worst case time for reader in writer preference and others
- There is exceptional difference in worst case time for reader in writer preference as preference is given to writer and what sleep time we are giving in input does not make all writer threads to simultaneously sleep in usleep after critical section,due to this,if even one writer thread gets ready for next loop,readers can not enter critical section.
- worst case time for other 3 curves is seen to generally increase for increasing readers.
- worst time increases for both reader and writer increases in fair solution as competition for critical section increases for both and access is given to one who request earlier.

9.4 Experiment 4:Worst case time vs Number of writers

- As expected,generally worst case time is seen to increase with number of writer threads
- worst case time for reader in reader writer thread increases exceptionally as it has to wait for all writer threads to complete its all loops due to given sleep dimensions
- worst case time for writer in writer preference is lowest as preference is given to it and growth is seen as number of writers increases,a thread may have to wait more
- observed worst case time is not exceptional in case of fair solution as in it preference is not given to any type of thread.
- worst case time for reader is observed to be more than writer in fair solution.
- as number of writers increase,readers have to wait more for their mutually excluded access compared to writer thread,hence,growth observed is more for readers compared to writers in fair solution

9.5 General observations over all experiments

- Average and worst case time increases with increase in both reader and writer threads in both algorithms
- In the experiments comparing fair preference versus writer preference, it's observed that giving preference to writers generally results in lower growth rates of average access time
- Despite potentially higher average access times, the fair solution ensures fairness by not giving preference to any type of thread. This results in similar average waiting times for both readers and writers.

10 Conclusion

- we implemented two algorithms as solution to reader writer problem and obtained their outputs.we were successful to implement mutual exclusion in both algorithms.4 experiments were conducted to note down average and worst case time for reader and writer threads in both algorithms,graph was plotted for all and possible reasons for outcome were discussed.we conclude that the choice of thread preference significantly affects average and worst-case access times,with fair solutions generally exhibiting higher access times due to lack of preference.The experiments also highlight the importance of considering both reader and writer counts, as well as the design choices such as sleep dimensions,in analyzing concurrent access algorithms.