CS3523: Programming Assignment-4

Manohar Maripe
CS22BTECH11036

March 18, 2024

1 Introduction

This report presents the design, implementation, and comparative analysis of two synchronization algorithms for solving the Readers-Writers problem: Writer Preference and Fair Readers-Writers. The algorithms were implemented in C++ using semaphores, and their performance was evaluated based on average and worst-case waiting times for accessing the critical section.

The Readers-Writers problem involves multiple threads concurrently accessing a shared resource, with different constraints for readers and writers. Two synchronization algorithms were implemented: Writer Preference and Fair Readers-Writers. This report aims to compare their performance in terms of average and worst-case waiting times.

2 Code Overview

The provided C++ codes simulates a multi-threaded system with fair scheduling and Writers preference. The program begins by reading parameters from an input file, including the number of reader and writer threads, the number of iterations for each, and timing parameters for the CS and remainder sections. Using semaphores for synchronization, reader and writer threads are spawned and execute concurrently. Within their respective loops, each thread requests access to the CS, enters the CS upon approval, simulates reading or writing, and exits the CS. Additionally, threads simulate the remainder section after accessing the CS. Timing data for thread requests and entries are recorded for subsequent analysis. Upon completion, the program generates an output file containing statistics on average and worst-case times for reader and writer threads accessing the CS and an output file containing request, entry, exit times for all reader and writer threads.

2.1 Header Inclusions

- `<fstream>`: Provides facilities for file input and output.
- `<thread>`: Provides classes and functions for working with threads.
- `<ctime>`: Defines various functions to manipulate date and time information.
- `<sys/time.h>`: Defines structures for time-related operations.
- `<pthread.h>`: POSIX threads library for multithreading support.
- `<sched.h>`: Provides functions for setting scheduling parameters for processes and threads.
- `<atomic>`: Atomic operations for thread synchronization.

2.2 File Input

The code begins by reading input parameters from a file named `inp-params.txt`. This part of the implementation ensures flexibility by allowing users to provide various values for all parameters. The file format is assumed to contain the parameters (nw, nr, kw, kr, mu-cs, mu-rem).

2.3 Main Function

The `main` function is the entry point of the program. It reads input parameters from the file `"inp-params.txt"`, initializes the variables with the values in input file, measures execution time, performs reader-writer problem using threads, and writes the result to the output file `"out.txt"` according to the algorithm implemented.

2.4 Thread Synchronization:

2.5 Time Calculation

2.5.1 getCurrentTime() Function

The `getCurrentTime()` function retrieves the current system time with microsecond precision. It utilizes the following components:

- **std::chrono**: The `std::chrono` library is used to manage time points and durations. It provides facilities for converting between different time representations.
- **std::strftime()**: This function formats the time obtained from `std::chrono` into a string with the specified format. It includes hours, minutes, seconds, and microseconds.

2.5.2 stringTolonglong() Function

The `stringTolonglong()` function converts a time string into a long long integer representing microseconds since the epoch. It involves the following steps:

- **Tokenization**: The input time string is tokenized based on the delimiter (':') to extract hours, minutes, seconds, and microseconds.
- **Conversion**: Each token is converted into a long long integer and combined to calculate the total time in microseconds.
- These functions are crucial for capturing timestamps at various stages of thread execution and calculating time durations for analysis.
- The program utilizes the above functions to calculate the time taken for each thread. After each threads completes their total execution, the total average time taken by the each thread is calculated by taking the difference between the entry time and request time in each iteration, summing up the difference got in each iteration and dividing by number of iterations of each thread (either kw or kr).
- **For Average case:** After calculating average of each thread, we find average in between readers threads and writers threads.
- **For Worst case:** After calculating average of each thread, we find worst or maximum value in between readers threads and writers threads.

2.6 File Output

The C++ code generates two output files: `output_[algorithm].txt` contains the request, entry, exit times of each thread and `time_[algorithm].txt` contains the average time for each thread, average time, worst case time for all reader and writer threads.

3 Graph Analysis

3.1 Experiment 1: Average Waiting Times with Constant Writers:

- In this graph, you measure the average time to enter the CS by reader and writer threads with a constant number of writers.
- The Y-axis will have time in milliseconds and will measure the average time taken to enter CS for each reader and writer thread.

- Here, we vary the number of reader threads nr from 1 to 20 in increments of 5 on the X-axis.
- We Fix the number of writer threads, nw = 10, kr = kw = 10.
- Specifically, the graph will have four curves:
- (a) Average time the reader threads take to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().
- (b) Average time the writer threads take to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers()

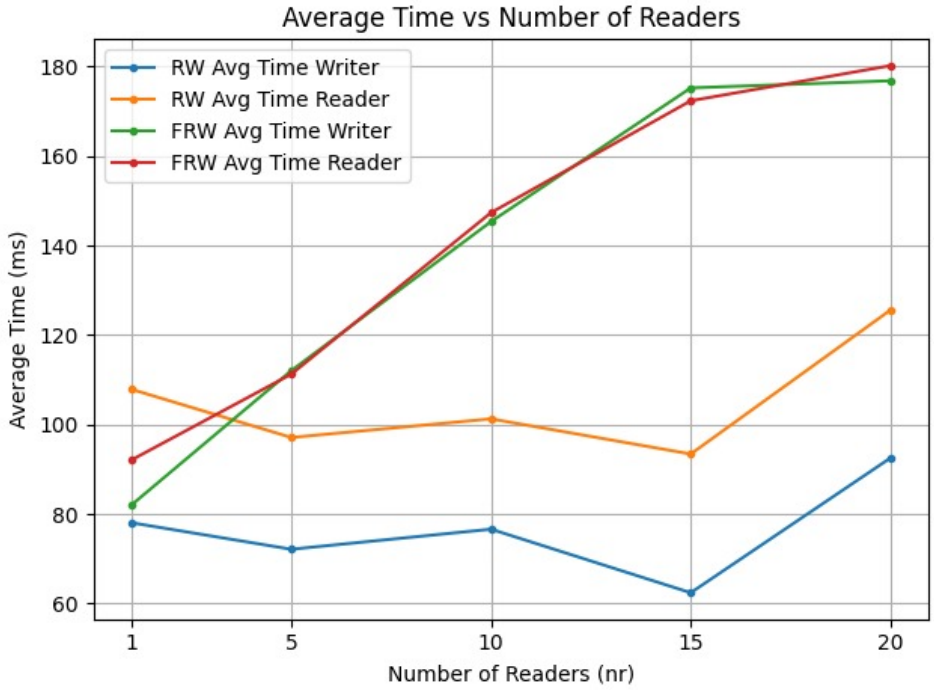

Figure 1: Time vs. Size(N)

Reader Threads (nr)	Avg.time for W readers	Avg.time for Fair readers	Avg.time for W writers	Avg.time for Fair writers
1	91.340	81.92	70.272	77.824
5	89.333	103.219	69.142	102.43
10	92.345	154.009	73.054	142.540
15	94.203	147.456	69.992	138.444
20	91.736	187.12	72.301	176.323

Table 1: Total Average Time with nr in milli seconds

3.2 Experiment 2: Average Waiting Times with Constant Readers:

- In this graph, you measure the average time to enter the CS by reader and writer threads with a constant number of readers.
- The Y-axis will have time in milliseconds and will measure the average time taken to enter CS for each reader and writer thread.
- Here, we vary the number of reader threads nw from 1 to 20 in increments of 5 on the X-axis.
- We Fix the number of writer threads, nr = 10, kr = kw = 10.

- Specifically, the graph will have four curves:
- (a) Average time the reader threads take to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().
- (b) Average time the writer threads take to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().

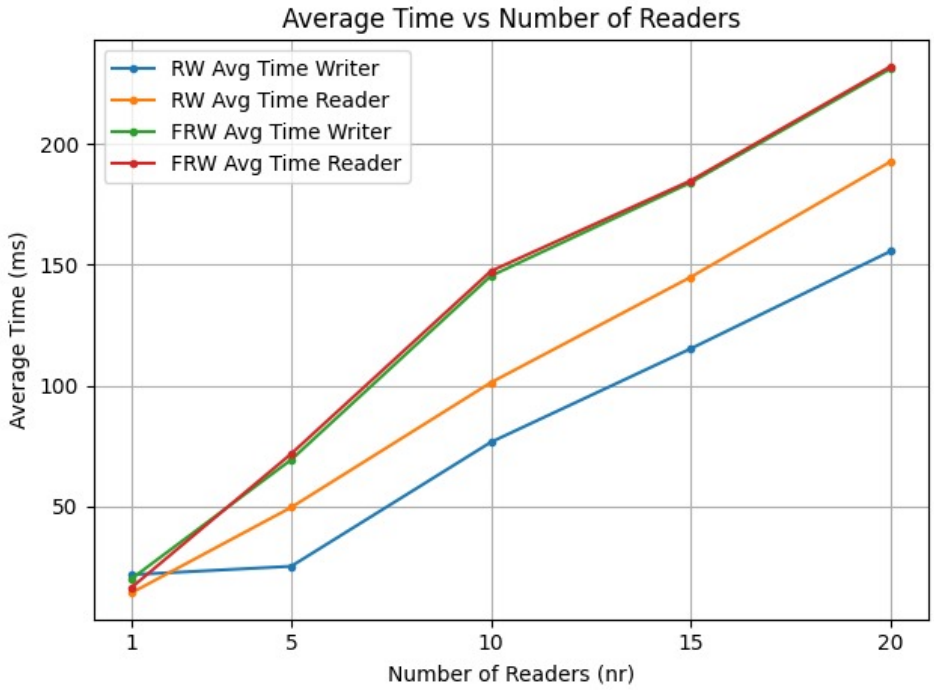

Figure 2: Time vs. rowInc.

Writer Threads (nw)	**Avg.time for W readers**	**Avg.time for Fair readers**	**Avg.time for W writers**	**Avg.time for Fair writers**
1	20.133	19.660	18.565	24.527
5	49.443	78.434	31.244	80.343
10	91.345	144.349	71.054	139.40
15	92.223	189.424	69.345	185.45
20	191.436	235.65	164.341	229.454

Table 2: Total Average Time with nw in milli seconds

3.3 Experiment 3: Worst-case Waiting Times with Constant Writers:

- The Y-axis will have time in milli-seconds and measure the worst-case time taken to enter CS by the reader and writer threads
- Here, we vary the number of reader threads nr from 1 to 20 in the increments 5 on the X-axis
- We fix all the other parameters: Number of writer threads, nw = 10, kr = kw = 10.
- Specifically, the graph will have four curves:
- (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 taken by the writer threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().

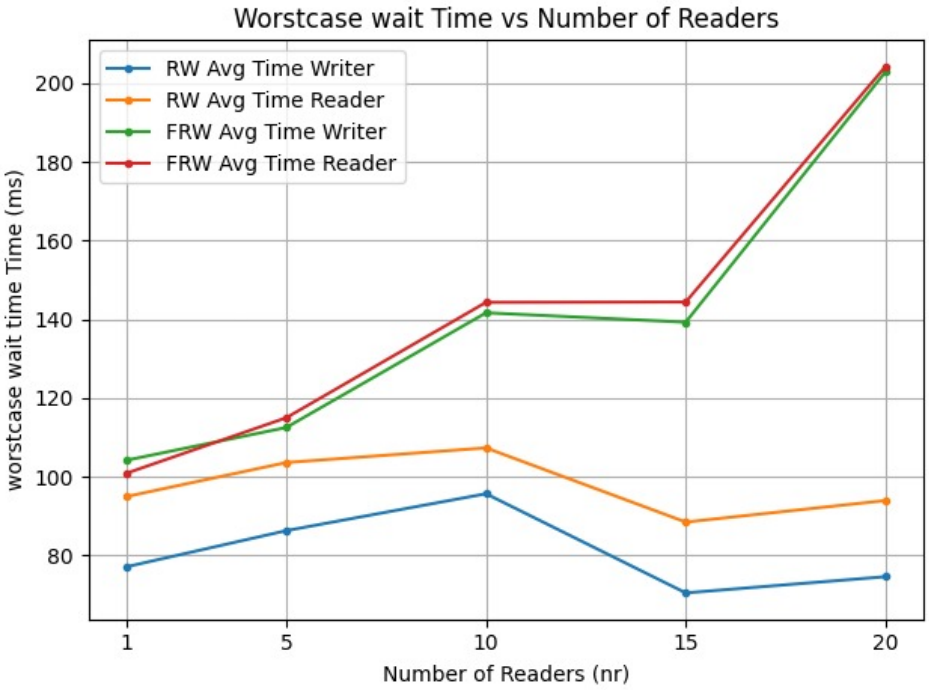

Figure 3: Time vs. Number of Threads (K).

Reader Threads (nr)	**Worst.time for W readers**	**Worst.time for Fair readers**	**Worst.time for W writers**	**Worst.time for Fair writers**
1	91.344	73.34234	80.5345	90.3434
5	91.737	122.3453	81.4343	122.34
10	91.56	140.459	78.4673	131.4340
15	92.34	180.345	76.754	172.435
20	96.324	196.43	83.434	188.34

Table 3: Worst Time with varying nr in milliseconds

3.4 Experiment 4: Worst-case Waiting Times with Constant Readers:

- The Y-axis will have time in milli-seconds and measure the worst-case time taken to enter CS by the reader and writer threads
- Here, we vary the number of reader threads nw from 1 to 20 in the increments 5 on the X-axis
- We fix all the other parameters: Number of writer threads, nr = 10, kr = kw = 10.
- Specifically, the graph will have four curves:
- (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 taken by the writer threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().

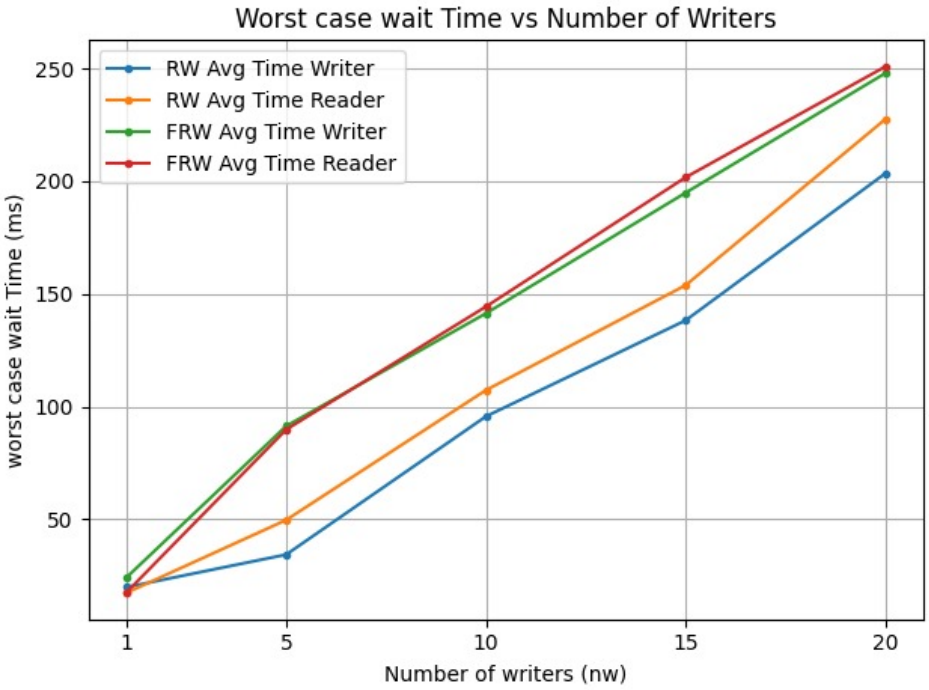

Figure 4: Time vs. Algorithms.

Writer Threads (nw)	**Worst.time for W readers**	**Worst.time for Fair readers**	**Worst.time for W writers**	**Worst.time for Fair writers**
1	19.949	24.324	14.444	24.576
5	49.994	81.233	37.324	73.334
10	91.56	140.459	78.4673	131.4340
15	155.67	212.424	136.76	213.45
20	193.554	245.56	176.3451	262.444

Table 4: Worst Time with varying nw in milliseconds

4 Conclusion

- Experiment 1 and 2:
- (a) Among the four threads, the writer thread in the writer-preference approach demonstrates superior efficiency. Its performance exceeds that of threads in the fair approach due to the writer thread's ability to wait only for its designated time. Conversely, in the fair approach, writer threads must wait longer to ensure fairness, allowing all preceding reader threads to finish.
- (b) In the fair approach, the waiting times for both reader and writer threads are nearly equal, as expected, since both are treated with equal priority.
- Experiment 3 and 4:
- (a) The maximum waiting times for reader threads in the writer-preference approach significantly exceed those of other threads because reader threads are vulnerable to starvation. Consequently, it's highly likely that certain reader threads experience starvation, thus increasing the maximum wait time.
- (b) In the fair approach, no thread experiences starvation, as both reader and writer threads are given equal priority. Consequently, the maximum waiting times remain consistent.
- (c) Writer threads in the writer-preference approach exhibit the shortest maximum wait time because they are prioritized over reader threads.