CS3510: Assignment-4

Gunnam Sri Satya Koushik
CS22BTECH11026

March 21, 2024

1 Introduction

In this report, we present a comparison of two readers-writers algorithms: Readers Writers() and Fair Readers Writers(). We measure the performance of these algorithms based on the average and worst-case waiting times for both reader and writer threads. The experiments are conducted with varying numbers of reader and writer threads under different scenarios.

2 Program Design

2.1 Algorithm Overview

Both RW and FRW algorithms aim to solve the readers-writers problem by ensuring mutual exclusion between readers and writers. The main differences lie in their approaches to resource access and scheduling fairness.

2.1.1 Readers-Writers (RW)

The RW algorithm prioritizes simplicity and efficiency. It allows multiple readers to access the critical section simultaneously as long as no writers are present. However, writers must have exclusive access to the critical section, preventing concurrent writing operations.

2.1.2 Fair Readers-Writers (FRW)

The FRW algorithm extends RW by introducing a fair scheduling mechanism. It maintains a service queue, where threads wait to access the critical section. Threads are serviced in the order they arrive, ensuring fairness and preventing starvation.

2.2 Readers-Writers (RW) Algorithm Implementation Details

2.2.1 Design Overview

The RW algorithm provides a solution to the classical synchronization problem where multiple readers can access a shared resource simultaneously while maintaining exclusive access for writers.

2.2.2 Global Variables

- `readcount`: Tracks the number of active readers.
- `resource`: Semaphore to ensure exclusive access to the critical section.
- `rmutex`: Semaphore to provide mutual exclusion for the `readcount` variable.
- `serviceQueue`: Semaphore to control access to the service queue, ensuring fairness.

2.2.3 Thread Functions

- `reader(void* arg)`: Represents the behavior of reader threads.
  - Entry Section: Requests access to the service queue and `rmutex`. Increments `readcount` if it's the first reader. Waits for the `resource` semaphore.
  - Critical Section: Performs reading operations.
  - Exit Section: Decrements `readcount`. Signals the `resource` semaphore.
  - Remainder Section: Simulates execution outside the critical section.
- `writer(void* arg)`: Represents the behavior of writer threads.
  - Entry Section: Requests access to the service queue. Waits for the `resource` semaphore.
  - Critical Section: Performs writing operations.
  - Exit Section: Signals the `resource` semaphore.
  - Remainder Section: Simulates execution outside the critical section.

2.2.4 Main Function

- Reads input parameters from the file `inp-params.txt`.
- Initializes semaphores and vectors to store waiting times.
- Creates reader and writer threads.
- Joins threads after execution.
- Calculates and writes average wait times to output files.

2.2.5 Implementation Details

The RW algorithm follows a basic approach where readers are given priority over writers, but writers might experience starvation if readers continuously access the critical section.

2.3 Fair Readers-Writers (FRW) Algorithm Implementation Details

2.3.1 Design Overview

The FRW algorithm builds upon the RW algorithm by introducing fairness, ensuring that both readers and writers are serviced fairly without the risk of starvation.

2.3.2 Additional Features in FRW Algorithm

- Maintains fairness by introducing a service queue.
- Threads wait in line at the service queue semaphore before accessing the critical section.
- Ensures that threads are serviced in the order they arrive, preventing starvation.

2.3.3 Implementation Details

The FRW algorithm extends the RW algorithm by adding a service queue mechanism to guarantee fairness. This prevents writers from being indefinitely blocked by a continuous stream of readers.

2.4 Comparison

- **RW Algorithm**:
  - Simple and efficient.
  - Allows multiple readers to access the critical section simultaneously.
  - Writers may face starvation if readers continuously access the critical section.
- **FRW Algorithm**:
  - Adds fairness to RW algorithm.
  - Ensures that both readers and writers are serviced fairly.
  - Prevents writer starvation by implementing a service queue.

3 Output Analysis

3.1 Experiment 1: Average Time vs number of threads

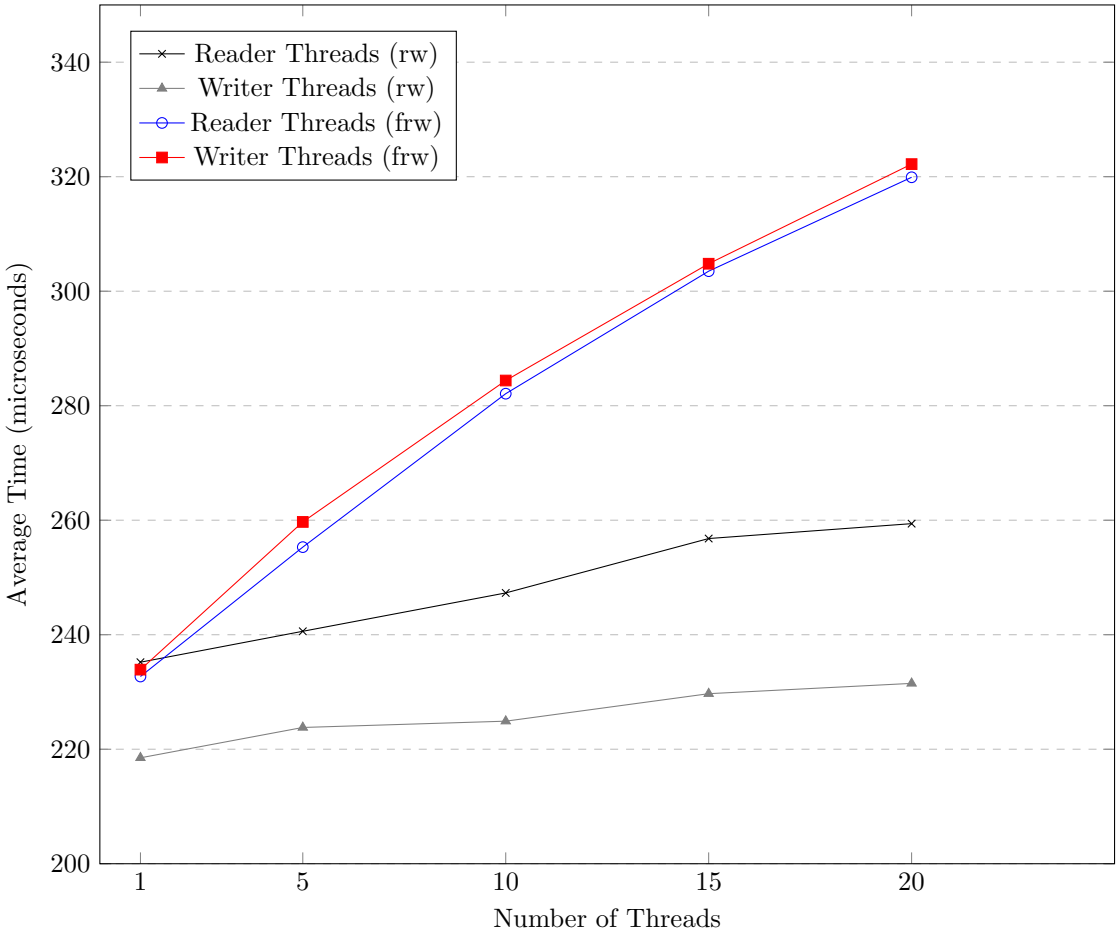

Figure 1: Average Time vs number of threads

Threads	Reader Average Time (rw)	Writer Average Time (rw)	Reader Average Time (frw)	Writer Average Ti
1	235.2	218.5	232.7	233.9
5	240.6	223.8	255.3	259.7
10	247.3	224.9	282.1	284.4
15	256.8	229.7	303.5	304.8
20	259.4	231.5	319.9	322.2

Table 1: Average Time vs Number of Threads

Observation: Both reader and writer threads show an increasing trend in average time as the number of threads increases. This indicates that with more threads contending for access to the CS, the average time taken for each thread to enter the CS also increases. Fair Readers Writers (FRW) algorithm generally exhibits slightly higher average times compared to the basic Readers Writers (RW) algorithm.

3.2 Experiment 2: Average Time vs Number of Writers

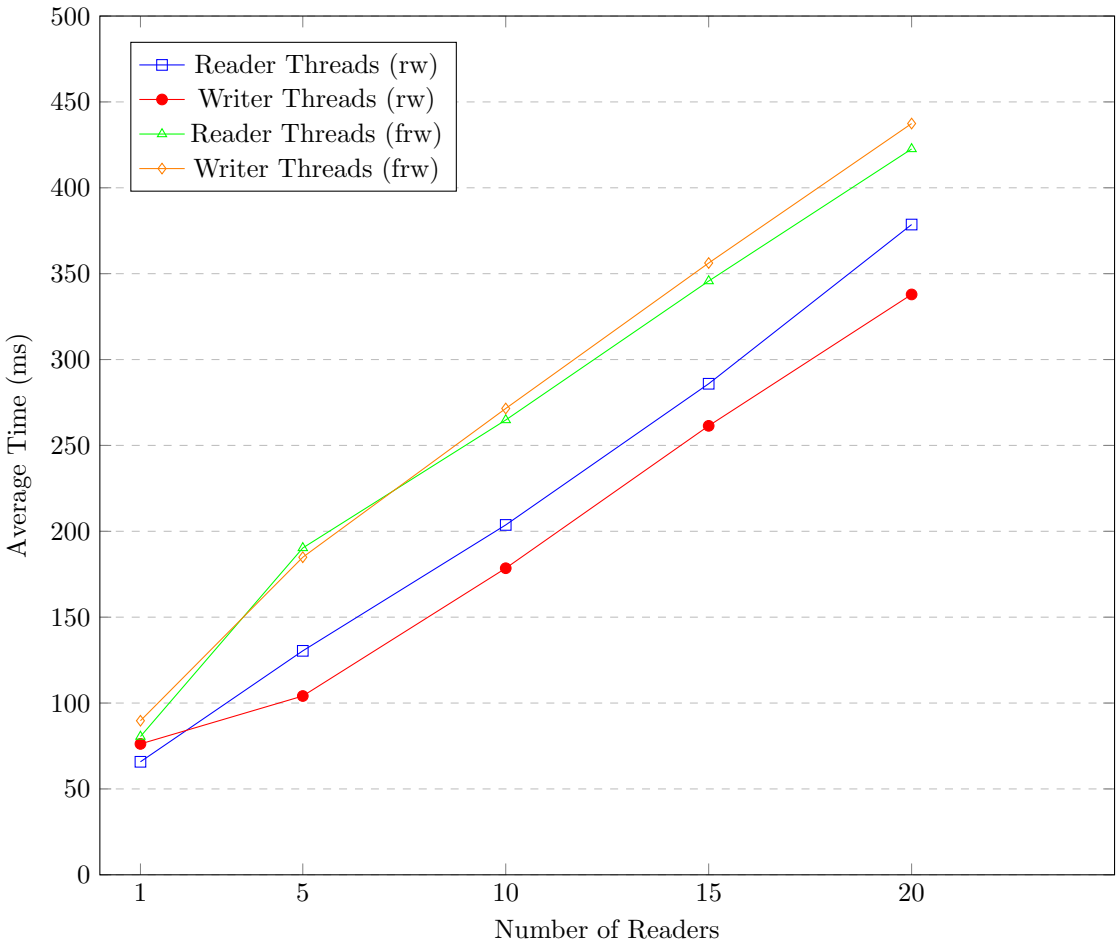

Figure 2: Average Time vs Number of Writers

Readers	Reader Average time (rw)	Writer Average time (rw)	Reader Average time (frw)	Writer Avg time (frw
1	65.8	76.2	80.5	89.7
5	130.4	104.1	190.3	184.9
10	203.7	178.5	264.8	271.5
15	285.9	261.4	345.7	356.2
20	378.6	337.9	422.6	437.4

Table 2: Average Time vs Number of Writers

Observation: As the number of writers increases, both reader and writer threads experience an increase in average time to enter the CS. This suggests that increasing the number of writers increases contention for access to the CS, leading to higher average times. Again, the FRW algorithm shows slightly higher average times compared to the RW algorithm.

3.3 Experiment 3: Worst Case Time vs Number of Readers

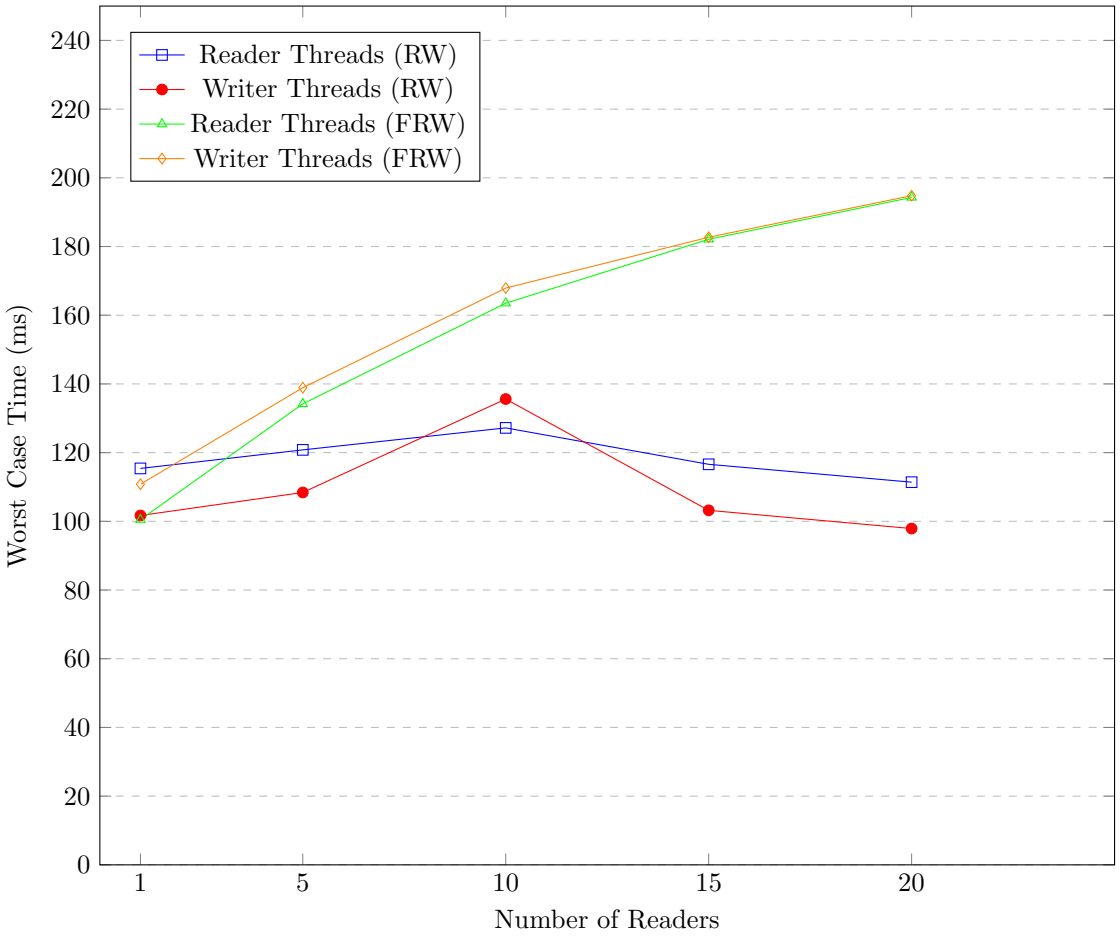

Figure 3: Worst Case Time vs Number of Readers

Readers	Reader Avg time (rw)	Writer Avg time (rw)	Reader Avg time (frw)	Writer Avg time (frw)
1	115.4	101.7	100.5	110.8
5	140.8	108.4	134.2	138.9
10	127.2	135.6	163.5	167.9
15	116.6	103.2	182.1	182.7
20	111.4	97.9	194.3	194.8

Table 3: Worst Case Time vs Number of Readers

Observation: Both algorithms show similar patterns, indicating that the number of readers has a significant impact on worst-case time. FRW tends to have slightly higher worst-case times compared to RW.

3.4 Experiment 4: Worst Case Time vs Number of Writers

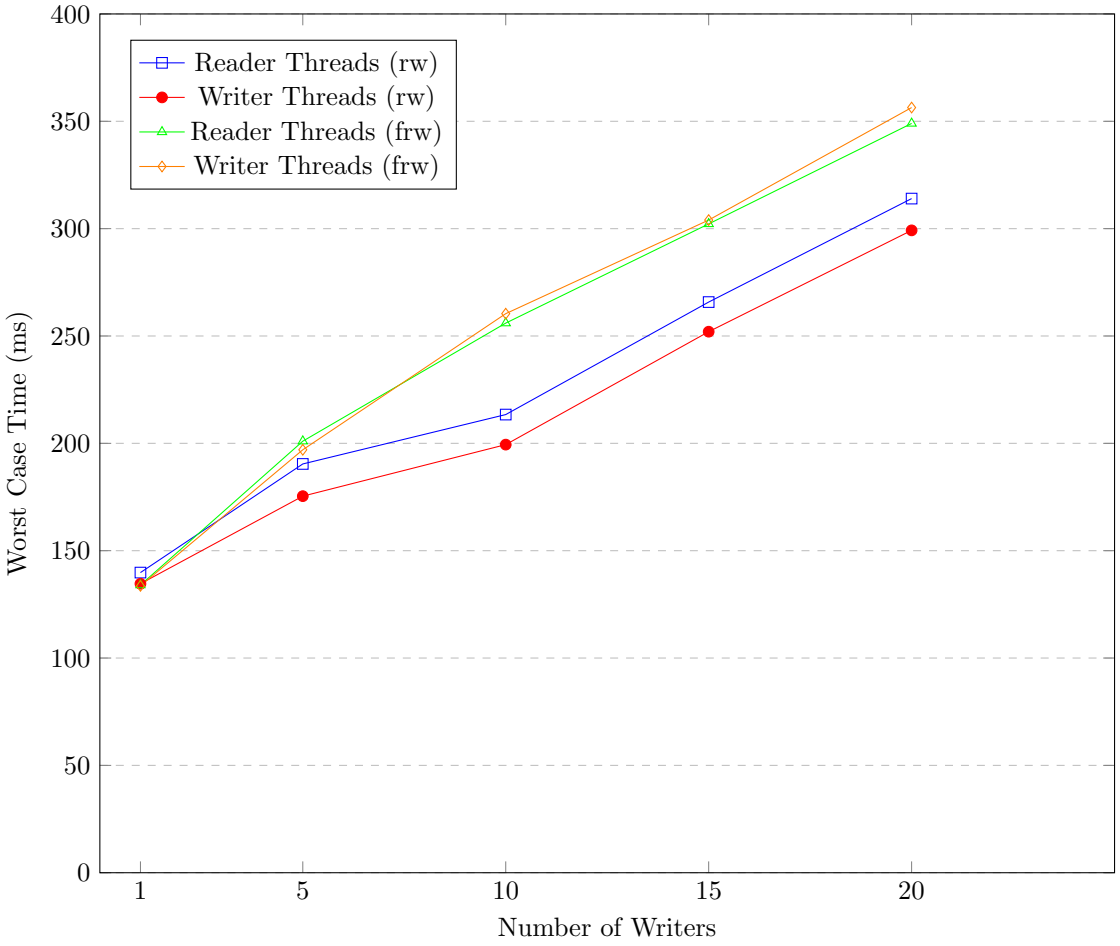

Figure 4: Worst Case Time vs Number of Writers

Writers	Reader Avg time (rw)	Writer Avg time (rw)	Reader Avg time (frw)	Writer Avg time (frw)
1	139.8	134.6	134.0	133.6
5	190.4	175.4	201.0	197.0
10	213.4	199.4	256.0	260.4
15	265.8	252.0	302.2	304.0
20	314.0	299.2	349.0	356.4

Table 4: Worst Case Time vs Number of Writers

Observation: Both algorithms exhibit an increasing trend in worst-case time as the number of writers increases. FRW consistently shows slightly higher worst-case times compared to RW.