REPORT ON solutions to Readers-Writers (writer preference) and Fair Readers-Writers problems using Semaphores

ESLAVATH NAVEEN NAIK
CS22BTECH11021

March 18, 2024

1 Report on 1st code(RW)

2 Introduction

The Readers-Writers problem is a classic synchronization problem concerning multiple threads that want to read and write a shared resource concurrently. This report discusses an implementation of the Readers-Writers problem in C++ using semaphores and threads.

3 Implementation Overview

The implementation consists of C++ code utilizing various standard libraries for file I/O, threading, semaphores, and time manipulation. The program reads input parameters from a file, including the number of readers and writers, the number of iterations each thread will execute, and the mean times for critical section (CS) operations and remainder section.

4 Synchronization Mechanisms

The implementation uses semaphores to ensure mutual exclusion and coordination between threads:

- `log_file_lock`: Ensures mutual exclusion for writing to the log file.
- `write_mutex`: Controls access for writers to the critical section.
- `read_mutex`: Controls access for readers to the critical section.
- `resource`: Ensures exclusive access to the shared resource.
- `read_try`: Allows readers to attempt to access the resource without blocking writers.

5 File I/O and Logging

The program logs the activities of each thread to a log file, including timestamps for critical section requests, entries, and exits.

6 Average Time Calculation

After all threads complete execution, the program calculates the average time for both writers and readers to gain entry to the critical section. It writes these averages to a separate file named "AverageTime.txt".

7 Conclusion

This implementation effectively demonstrates how to solve the Readers-Writers problem using semaphores and threads in C++. By ensuring proper synchronization and mutual exclusion, it provides concurrent access to shared resources for both readers and writers while maintaining consistency and integrity.

8 Recommendations for Improvement

- Implement error handling for file I/O operations and semaphore initialization.
- Enhance logging functionality to include more detailed information or debugging data.
- Experiment with different strategies for delaying thread execution to better simulate real-world scenarios.
- Consider optimizing the code for better performance or scalability in larger-scale systems.

9 Report on 2nd Code(fair)

10 Introduction

The provided code implements a multi-threaded program to simulate the behavior of writers and readers accessing a critical section (CS) protected by semaphores. The goal is to analyze the fairness of access for both writers and readers under varying system parameters.

11 Code Overview

The code consists of several components:

- Includes necessary libraries for threading, file I/O, semaphores, and time manipulation.
- Defines global variables and functions for managing semaphores, generating timestamps, and calculating system time.
- Implements writer and reader functions to simulate access to the critical section.
- In the main function, system parameters are initialized from an input file, semaphores are initialized, and threads are created to execute writer and reader functions.

12 Semaphore Usage

Semaphores are utilized to manage access to the critical section:

- `write_mutex` ensures mutual exclusion among writer threads.
- `read_mutex` ensures mutual exclusion when updating the `read_count` variable.
- `resource` controls access to the critical section itself.
- `log_file_lock` provides mutual exclusion for writing to the log file.
- `read_try` allows only one reader to attempt to access the resource at a time.

13 Thread Functions

- **Writer Function**: Simulates a writer thread attempting to enter the critical section, performing a write operation, and exiting. Delays are introduced to simulate computation and wait times.
- **Reader Function**: Simulates a reader thread attempting to enter the critical section, performing a read operation, and exiting. Similar delays are introduced.

14 Main Function

- Initializes system parameters from an input file.
- Initializes semaphores.
- Creates threads for writers and readers.
- Collects data on entry times for analysis.

15 Results

After running the simulation, the average entry times for writers and readers are calculated:

Average entry time for writers: [$average\_writer\_time$] milliseconds.
Average entry time for readers: [$average\_reader\_time$] milliseconds.

16 Conclusion

The analysis provides insights into the fairness of access to the critical section for both writers and readers under varying system conditions. Further optimizations or adjustments to parameters could potentially improve fairness or performance.

17 Graph Observations

Table 1: Average Waiting Times with Constant Writers

nr	F Reader	F Writer	W Reader	W Writer
1	0	90.814	0	56.210
5	74.960	87.70	76.347	71.150
10	109.899	106.169	76.868	62.211
15	145.157	144.689	79.53	55.677
20	141.563	123.345	94	72.247

Table 2: Average Waiting Times with Constant Readers

nw	F Reader	F Writer	W Reader	W Writer
1	11.753	0	14.939	0
5	54.852	55.227	43.17	26.469
10	125.841	131.154	89.106	65.421
15	132.038	135.091	152.870	129.74
20	200.21	216.231	170.879	151.444

Table 3: Worst-case Waiting Times with Constant Writers

nr	F Reader	F Writer	W Reader	W Writer
1	0	253.179	0	330.379
5	201.669	309.539	968.480	134.581
10	370.268	346.829	923.328	229.371
15	445.189	392.252	1011.52	220.628
20	645.219	433.071	1155.4	302.811

Table 4: Worst-case Waiting Times (ms) with Constant Readers

nw	W Reader	W Writer	F Reader	F Writer
1	97.139	0	68.559	0
5	520.338	85.986	203.081	164.259
10	1206.48	212.090	290.247	295.444
15	1461.20	359.301	520.180	540.418
20	1780.28	585.201	820.168	932.391

18 Graphs

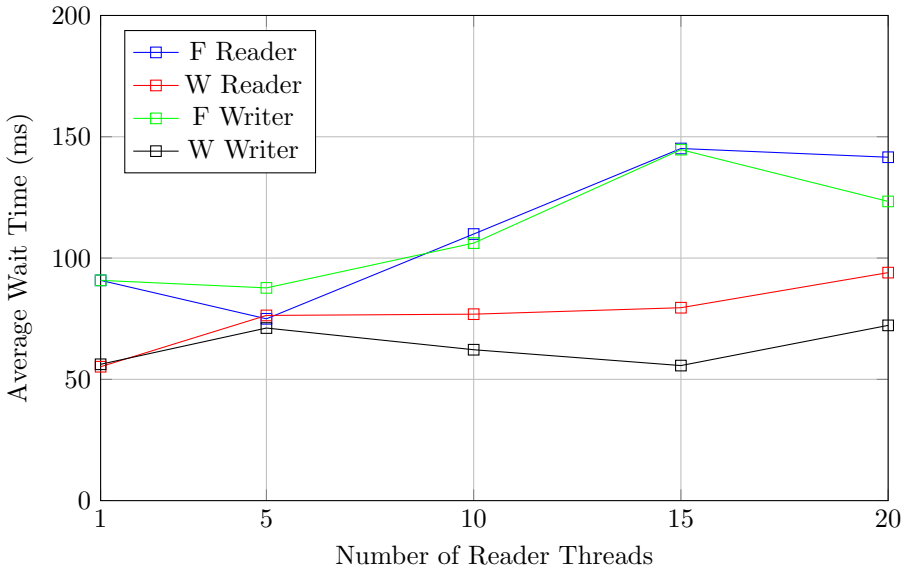

Figure 1: Average Waiting Times with Writers

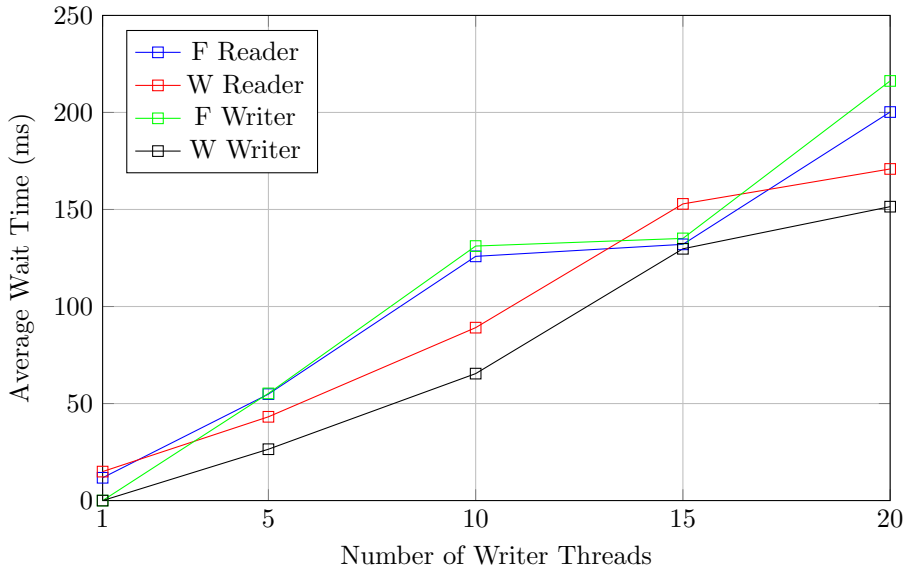

Figure 2: Average Waiting Times with Readers

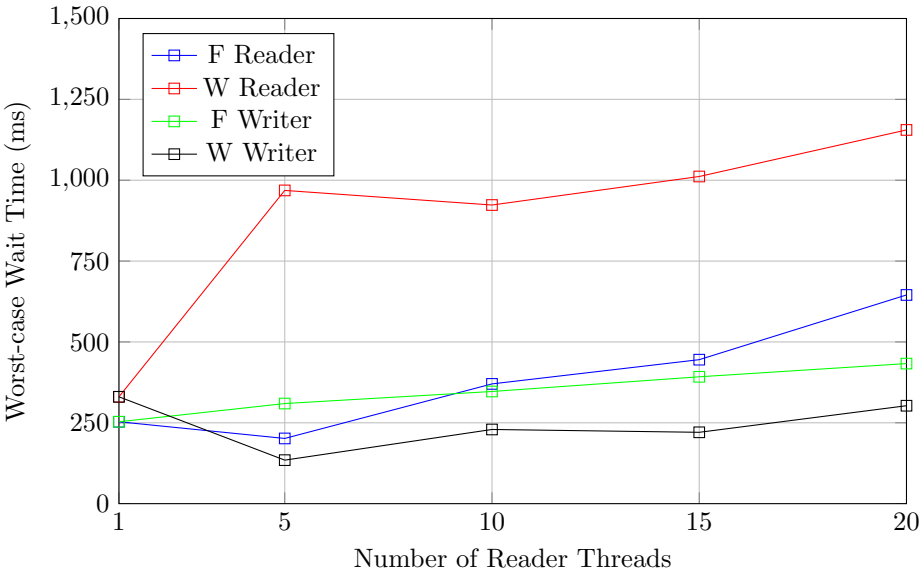

Figure 3: Worst-case Waiting Times with Writers

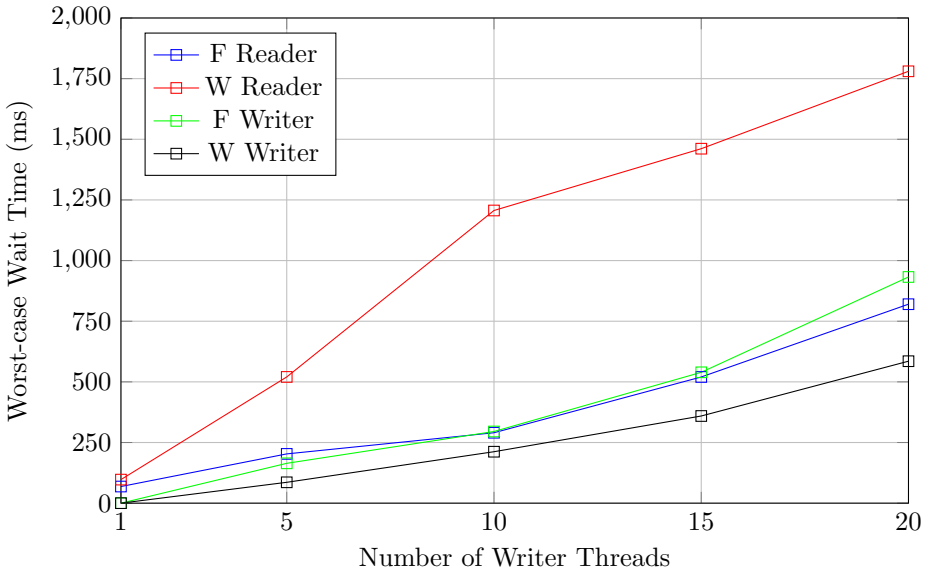

Figure 4: Worst-case Waiting Times with Readers

19 Graph Analysis

1. **Experiment 1 and 2:**

   (a) (a) Among the four threads, the writer thread excels in the writer preference solution. Its performance surpasses that of threads in the fair solution due to the writer preference solution's optimization, ensuring that a writer thread waits only as long as necessary. Conversely, in the fair solution, writer threads must wait longer to maintain fairness, allowing all reader threads that requested access to the critical section before them to complete first.

   (b) As anticipated, the wait times for reader and writer threads are nearly identical in the fair solution. This outcome aligns with expectations, given that both reader and writer threads are afforded equal preference in this scenario.

2. **Experiment 3 and 4:**

   (a) The worst-case waiting times of the reader thread in the writer preference solution are significantly higher than other threads due to the possibility of reader thread starvation. This increases the likelihood of certain reader threads being starved, thereby increasing the worst-case wait time.

   (b) In the fair solution, no thread experiences starvation as both reader and writer threads are given equal preference. Consequently, the worst-case waiting times are identical.

   (c) Writer threads in the writer preference solution enjoy the least worst-case wait time because of their prioritization over reader threads. This preference ensures minimal delays for writer threads, leading to reduced worst-case wait times.