Programming Assignment-4

Kandi Prudhvi
CS22BTECH11031

March 18, 2024

1 Introduction

This report describes the aim to solve the Readers-Writers problem (writer preference) and Fair Readers-Writers problem using Semaphores as discussed in C++.

1.1 FILE INPUT

The program reads integers nw,nr,kw,kr,mUCS, mURem from an input file named inp-params.txt and stored them in variables nw,nr,kw,kr,mucs,mrem respectively.nr is the number of reader threads. nw is the number of writer threads.kr is the number of times each reader thread tries to enter the CS.kw is the number of times each writer thread tries to enter the CS.

1.2 Low Level Design

The program reads the N,K,rowInc values from input file and store them in variables n,k,rowinc respectively. Then it reads the matrix A from input file and store it in 2D vector named v. There are four algorithms and each algorithm is written in different codes. So, main function will be same for for all four codes. Creates a vector of threads (threads) to perform matrix multiplication using the tas or cas or bcas or atom function. Each thread is assigned a chunk of rows to process concurrently.The main function waits for all threads to finish their computations by using join on each thread.

1: **Writer Preference**:
Initialization: The program initializes various parameters and synchronization primitives such as semaphores to control access to shared resources.
Thread Functions:
Reader Function: The reader function is responsible for handling the behavior of threads that aim to read from the shared resource. When a reader thread is invoked, it first requests permission to access the critical section by waiting in line using semaphores. Importantly, the reader function coordinates with other threads to ensure that writers are given precedence over readers. If a writer is currently accessing the critical section, the reader thread patiently waits its turn. Once granted access, the reader enters the critical section, reads the required data, and subsequently exits the critical section, relinquishing control. Throughout this process, synchronization mechanisms such as semaphores are employed to prevent conflicts and maintain thread safety.
Writer Function: Contrary to readers, writer threads are tasked with modifying the shared resource. Upon invocation, a writer thread seeks exclusive access to the critical section to perform its write operation. It makes a request for entry, effectively signaling its intention to write. Similar to the reader function, writers respect the priority of other writers and ensure that no reader or additional writer enters the critical section concurrently. Once inside the critical section, the writer executes its write operation, potentially altering the shared data. Subsequently, it exits the critical section, allowing other threads to access the resource. Just like with reader threads, synchronization primitives like semaphores are employed to enforce mutual exclusion and manage access to the shared resource effectively.
Main Function:
File Input: Reads parameters like the number of reader and writer threads from an input file.
Thread Creation: Creates and starts reader and writer threads.
Thread Joining: Waits for all threads to finish execution.
Statistics Calculation: Computes average and worst-case waiting times for readers and writers based on their entry and request times.

Output: Writes statistics to an output file for analysis.
Semaphore Usage: Semaphores are utilized to enforce synchronization among threads, ensuring that readers and writers coordinate their access to shared resources appropriately.

2: **Fair Readers writers**:
Initialization: The program initializes various parameters and synchronization primitives such as semaphores to control access to shared resources.
Thread Functions:
Reader Function: The reader function represents the behavior of a reader thread in the fair readers writers problem. When a reader thread is executed, it undergoes several steps. First, it records the time at which it requests access to the critical section. Then, it enters a queue to await access, ensuring fairness among threads. Afterward, it increments the read count and acquires the lock if it's the first reader, allowing multiple readers to access the critical section simultaneously. Once inside the critical section, the reader simulates reading operations. It logs the entry and exit times of the critical section before decrementing the read count and releasing the lock if it's the last reader, ensuring fairness. Finally, the thread simulates execution in the remainder section before completing its iteration.
Writer Function: On the other hand, the writer function represents the behavior of a writer thread. Similarly, it begins by recording the time at which it requests access to the critical section and enters a queue to await access. However, unlike readers, writers require exclusive access to the shared resource. Therefore, after entering the queue, a writer requests exclusive access to the shared resource by acquiring a semaphore lock. Once granted access, the writer enters the critical section and simulates writing operations. It logs the entry and exit times of the critical section before releasing exclusive access to the shared resource, ensuring fairness among threads. Similar to the reader function, the writer also simulates execution in the remainder section before completing its iteration.
Main Function:
File Input: Reads parameters like the number of reader and writer threads from an input file.
Thread Creation: Creates and starts reader and writer threads.
Thread Joining: Waits for all threads to finish execution.
Statistics Calculation: Computes average and worst-case waiting times for readers and writers based on their entry and request times.
Output: Writes statistics to an output file for analysis.
Semaphore Usage: Semaphores are utilized to enforce synchronization among threads, ensuring that readers and writers coordinate their access to shared resources appropriately.

1.3 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. Here, we vary the number of reader threads nr from 1 to 20 in increments of 5 on the X-axis. All the other parameters are fixed: Number of writer threads, nw = 10, kr = kw = 10. The Y-axis will have time in milliseconds and will measure the average time taken to enter CS for each reader and writer thread.

Reader threads(nr)	Avg Time by W readers(in millisec)	Avg Time by fair readers(in millisec)	Avg Time by W writers (in millisec)	Avg Time by fair writers(in millisec)
1	0.65	0.399	0.505	0.02023
5	0.408	0.553	0.179	0.547
10	0.619	0.724	0.437	0.633
15	0.738	1.096	0.464	0.792
20	0.628	0.817	0.424	0.684

Table 1: Avg Time taken for 2 algorithms

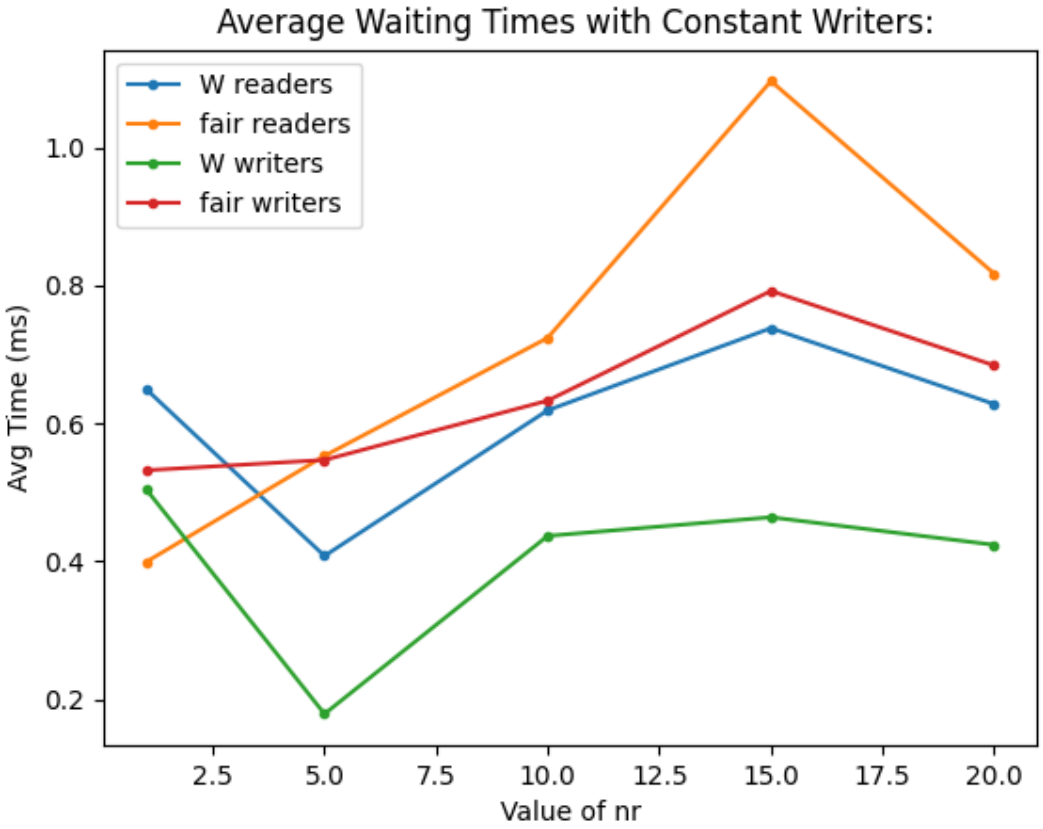

Figure 1: Average Waiting Times with Constant Writers

1.4 Experiment 2: Average Waiting Times with Constant Readers:

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

writer threads(nw)	Avg Time by W readers(in millisec)	Avg Time by fair readers(in millisec)	Avg Time by W writers (in millisec)	Avg Time by fair writers(in millisec)
1	0.36	0.55	0.264	0.317
5	0.626	0.909	0.362	0.719
10	0.659	0.863	0.505	0.857
15	1.074	1.008	0.824	1.004
20	0.672	1.092	0.314	1.119

Table 2: Avg Time taken for 2 algorithms

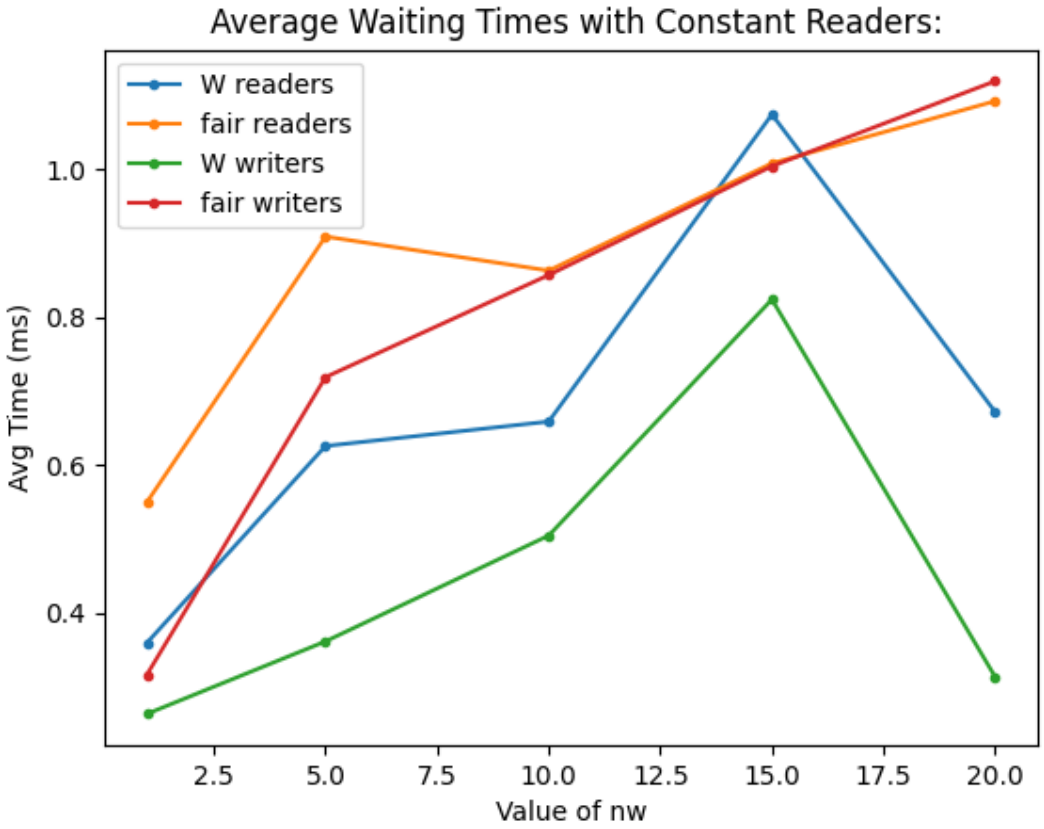

Figure 2: Average Waiting Times with Constant Readers

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

This graph will be similar to the graph in Step 1. In this graph, you measure the worst-case (instead of average) time taken to enter the CS by reader and writer threads with a constant number of writers. Here we vary the number of reader threads nr from 1 to 20 in the increments 5 on the X-axis. All the other parameters are fixed: Number of writer threads, nw = 10, kr = kw = 10. 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.

Reader threads(nr)	Avg Time by W readers(in millisec)	Avg Time by fair readers(in millisec)	Avg Time by W writers (in millisec)	Avg Time by fair writers(in millisec)
1	0.544	0.434	0.579	0.63
5	0.759	0.585	0.596	0.721
10	1.216	1.256	0.863	1.199
15	1.172	1.223	0.535	1.215
20	1.235	1.118	0.745	1.226

Table 3: Avg Time taken for 2 algorithms

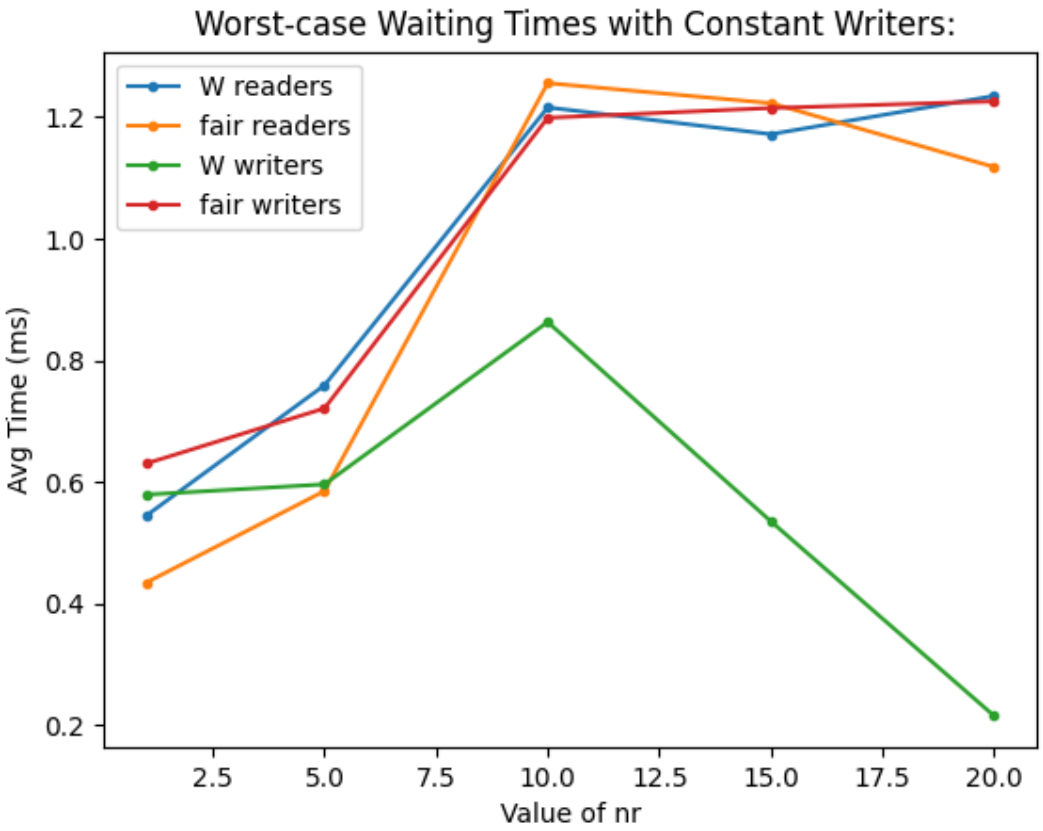

Figure 3: Worst-case Waiting Times with Constant Writers

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

This graph will be similar to the graph in Step 2. In this graph, you measure the worst-case time taken to enter the CS by reader and writer threads with a constant number of readers. Here, we vary the number of

writer threads nw from 1 to 20 in increments of 5 on the X-axis. All the other parameters are fixed: Number of reader threads, nr = 10, kr = kw = 10. The Y-axis will have time in milliseconds and will measure the worst-case time taken to enter CS for each reader and writer thread.

Reader threads(nr)	Avg Time by W readers(in millisec)	Avg Time by fair readers(in millisec)	Avg Time by W writers (in millisec)	Avg Time by fair writers(in millisec)
1	0.534	0.334	0.573	0.53
5	0.729	0.572	0.595	0.741
10	1.243	1.206	0.853	1.39
15	1.171	1.212	0.435	1.225
20	1.232	1.112	0.645	1.236

Table 4: Avg Time taken for 2 algorithms

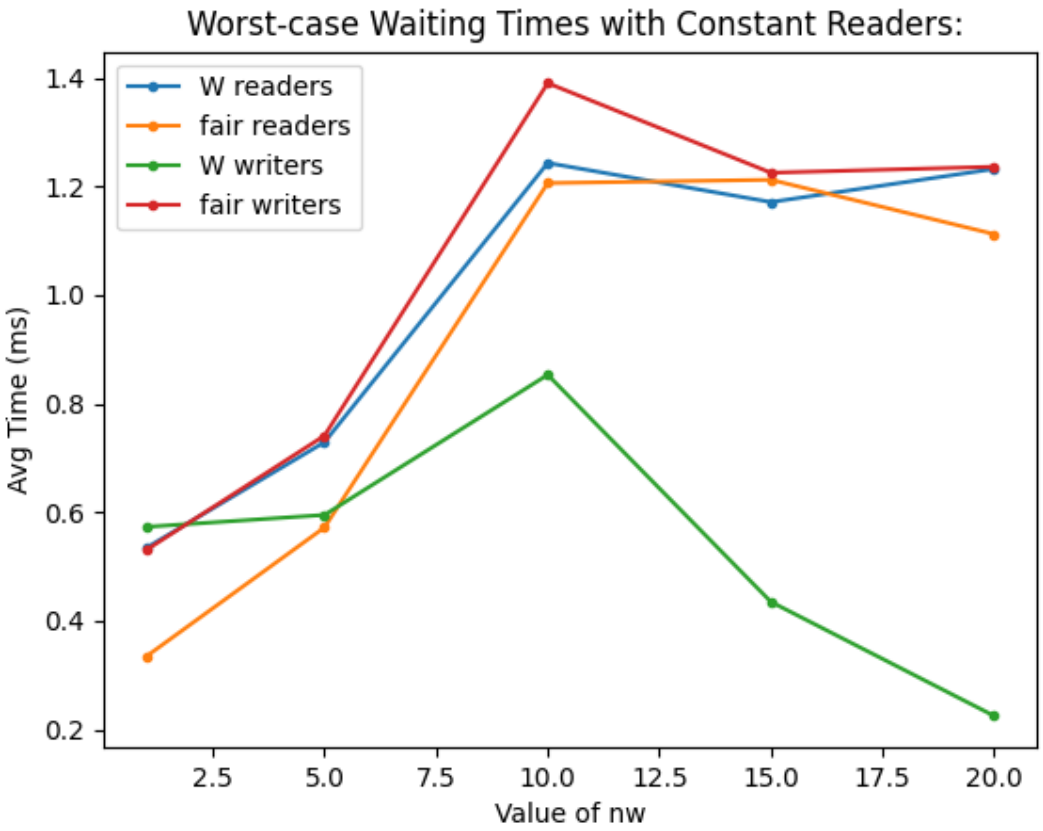

Figure 4: Worst-case Waiting Times with Constant Readers

1.7 Analysis of Graphs :

Average time taken increases as nr increases for all readers and writers in both algorithms.
Average time taken increases as nw increases for all readers and writers in both algorithms.
Time taken is less in the second case than the first case.
Worst time taken increases as nr increases for all readers and writers in both algorithms.
Worst time taken increases as nw increases for all readers and writers in both algorithms.
Time taken is less in the second case than the first case.