Programming Assignment 4

Karthik Kunchala

1 Classes

- **Semaphore:**
  - Class representing a semaphore, responsible for managing access to shared resources.
  - Methods: `wait()`, `signal()`, `initialize()`.
- **ReaderWriterManager:**
  - Class managing the reader-writer problem.
  - Contains semaphores for synchronization (`mutex`, `writeblock`, `rMutex`, `wMutex`, `rTry`) and shared data (`shared_data`).
  - Methods: `writer()`, `reader()`, `calculateAverage()`, `main()`.

2 Methods and Functions

- **writer(id, kw, randCSTime, randRemTime, logFile, entryTimes):**
  - Represents a writer thread.
  - Acquires lock for writing, updates shared data, releases lock, and logs entry/exit times.
  - Parameters: `id` (thread id), `kw` (number of critical sections), `randCSTime` (random critical section time), `randRemTime` (random remainder time), `logFile` (file for logging), `entryTimes` (vector to store entry times).
- **reader(id, kr, randCSTime, randRemTime, logFile, entryTimes):**
  - Represents a reader thread.
  - Acquires lock for reading, reads shared data, releases lock, and logs entry/exit times.
  - Parameters: `id` (thread id), `kr` (number of critical sections), `randCSTime` (random critical section time), `randRemTime` (random remainder time), `logFile` (file for logging), `entryTimes` (vector to store entry times).
- **calculateAverage(times, algorithm):**

- Calculates the average time for a thread to gain entry to the critical section.
- Parameters: `times` (vector of entry times), `algorithm` (name of the algorithm).

- **main():**
  - Entry point of the program.
  - Initializes semaphores, creates writer and reader threads, waits for threads to complete, calculates and logs average times.

3 Concurrency and Synchronization

- Writers and readers access shared data (`shared_data`) using semaphores (`mutex`, `writeblock`, `rMutex`, `wMutex`, `rTry`) to ensure mutual exclusion.
- Writers have preference over readers to prevent starvation using the `rTry` semaphore.
- Entry times of threads are logged for analysis and performance evaluation.
- Threads sleep for random intervals to simulate real-world scenarios.

4 File Handling

- Log files (`RW-log.txt`, `Average time.txt`) are opened and closed in the `main()` function.
- Entry times and average times are logged to respective files for analysis.

5 Error Handling

- Minimal error handling is included in the provided code.
- It's assumed that file I/O operations and semaphore initialization succeed without errors.

6 Data Structures

- **Shared Data:** Data accessed by reader and writer threads.
- **Semaphores:**
  - `mutex`: Semaphore controlling access to critical sections.
  - `writeblock`: Semaphore for exclusive writer access.

- `rMutex`: Mutex for reader count (`rC`).
- `wMutex`: Mutex for writer count (`wC`).
- `rTry`: Semaphore for writer preference.
- `queue`: Semaphore for implementing the fair solution.

7 Critical Section Execution

- **Reader Thread:**
  - Requests entry and reads shared data.
- **Writer Thread:**
  - Requests entry and writes to shared data.

8 Output

- Log all events (requests, entries, and exits) to a file.
- Calculate and output average entry times for readers and writers.

9 Experimental Results

9.1 Average Waiting Times with Constant Writers

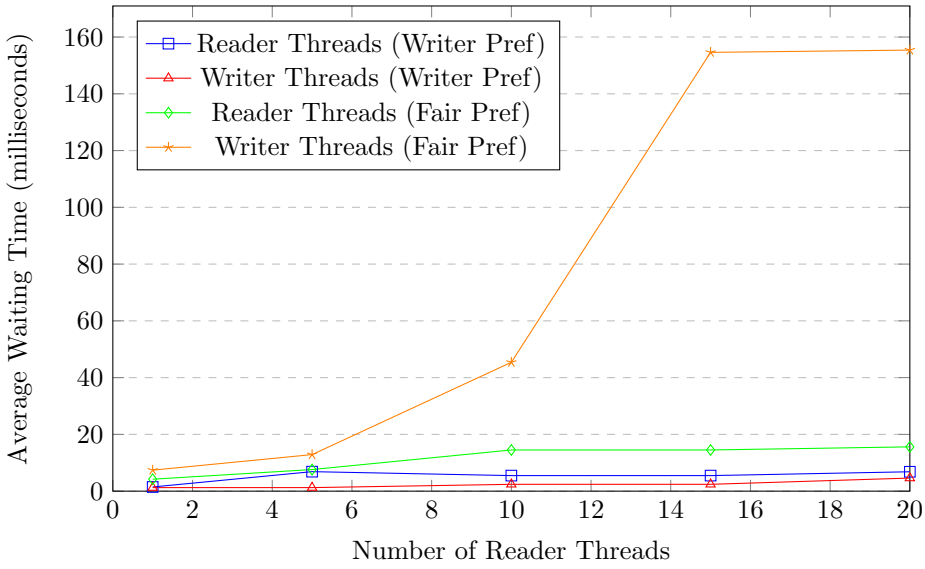

Figure 1: Average Waiting Times with Constant Writers

- In the writer preference solution, readers tend to experience longer average wait times compared to writers. This discrepancy arises because writers are given higher priority, resulting in quicker access to the critical section.
- On the other hand, the fair solution ensures that both readers and writers encounter similar average wait times. This equality stems from the implementation of a fair access policy, where both readers and writers have comparable chances of accessing the critical section.
- However, the fair solution typically exhibits higher average wait times overall when compared to the writer preference solution. This is due to the fair solution's utilization of a queue mechanism, which introduces additional waiting time for threads.
- In the fair solution, both readers and writers face nearly identical worst-case wait times. This uniformity arises from the equitable allocation of access to the critical section, ensuring fairness in thread execution.
- With an increase in the number of readers, the fair solution experiences a notable escalation in overall waiting times. This occurs because a larger pool of readers leads to lengthier queues, subsequently prolonging the wait times for both readers and writers.

9.2 Average Waiting Times with Constant Readers

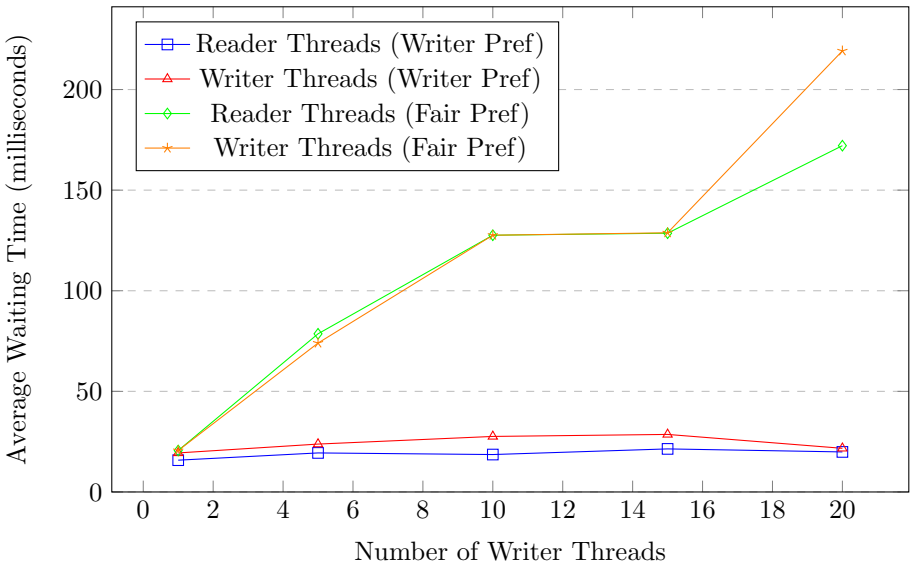

Figure 2: Average Waiting Times with Constant Readers

- In the second experiment, it is observed that the fair readers and writers exhibit nearly identical average wait times, aligning with expectations based on the fairness principle inherent in the solution.
- Upon closer examination, it becomes evident that all four curves converge as the number of writers increases. This convergence indicates that as writers become more numerous, readers experience heightened wait times due to increased contention for access to the critical section.
- In the writer preference solution, where writers are afforded priority, readers endure prolonged wait times as they compete for access with writers. Consequently, readers exhibit higher average waiting times compared to writers in the writer preference scenario.
- Overall, the average waiting time in the writer preference solution escalates due to the amplified volume of resource requests, resulting in heightened waiting periods for both readers and writers.
- Conversely, in the fair solution, irrespective of whether the number of readers or writers increases, the average waiting time consistently rises. This trend stems from the fair allocation of access to the critical section, which invariably leads to increased waiting times as contention for resources intensifies.

9.3 Worst-case Waiting Times with Constant Writers

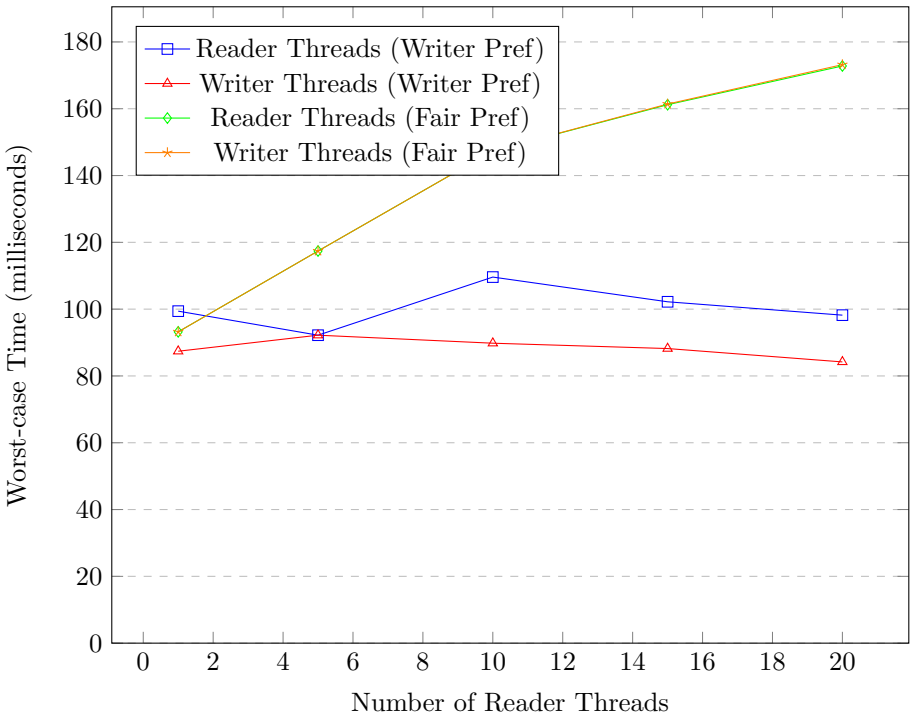

Figure 3: Worst-case Waiting Times with Constant Writers

- In the third experiment utilizing the writer preference solution, it is observed that the worst-case time for readers exceeds that of writers. This disparity arises from the prioritization of writers over readers, resulting in readers experiencing prolonged waiting periods as resources are allocated preferentially to writers. Consequently, when a writer thread is executing and both readers and writers await access to the resource, the resource is allocated to the writer, exacerbating the worst-case waiting time for readers.

- Conversely, in the fair solution, both readers and writers exhibit nearly identical worst-case waiting times due to the equitable distribution of resource access inherent in the fairness principle. With an increase in the number of readers, the waiting time escalates as both readers and writers contend for access, resulting in prolonged wait times as threads await their turn in the queue.

9.4 Worst-case Waiting Times with Constant Readers

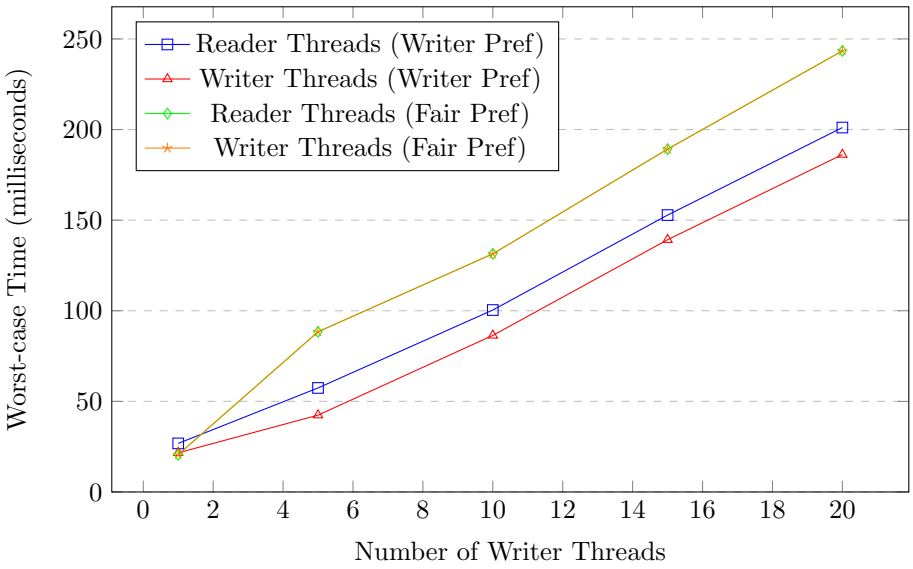

Figure 4: Worst-case Waiting Times with Constant Readers

- In the fourth experiment, when utilizing the fair solution, the observations from experiment 3 remain consistent. The fairness of this solution ensures that an increase in the number of readers or writers leads to a proportional increase in the worst-case waiting time. This phenomenon arises due to the equal treatment of all threads, resulting in a fair but inevitably lengthier waiting period as demand for resource access rises.

- Conversely, in the writer preference solution, there is an observed increase in waiting time. This increase primarily affects readers, who encounter extended waiting periods as writers are granted priority access to the resource. As a result, readers experience higher average waiting times compared to writers. Overall, the average waiting time for the writer preference solution rises due to an increase in resource requests, leading to prolonged waiting periods for both readers and writers.

10 Conclusion

The experiments conducted to address the reader-writer problem provided valuable insights into the performance and behavior of two distinct solutions: writer preference and fair access. In the writer preference solution, readers often experience longer waiting times compared to writers due to the prioritization of writer access to the critical section. Conversely, the fair solution, employing a first-come-first-serve queue, ensures equitable access for both readers and writers, albeit with higher overall waiting times. As thread counts increase, the fair solution exhibits proportionate increases in waiting times, while the writer preference solution sees exacerbated waiting times for readers, maintaining its prioritization of writers. Ultimately, the choice between these solutions depends on the system's requirements, balancing efficiency and fairness considerations.