Introduction

The problem addressed in this report is called the Reader/Writer locks problem. The following problem was developed as different operations on data structures require different locking procedures. If data is only being read from a data structure, then many concurrent threads should be able to access the data at once. On the other hand, if an operation is being preformed on a data structure that will modify it in some way, then only one thread should have access to that critical section at any given time. With that said if one reader thread is in the critical section then other reader threads can join as well. The disadvantage with this algorithm is that writer threads can be easily starved as it would have to wait till there are no reader threads are in the critical section. My solution to this problem is addressed down below.

Solution

My solution to this problem involves creating a universal semaphore, where both reader and writers are required to claim before attempting to get reader and or writer locks. This universal lock still allows concurrent readers to access a data structure meanwhile also giving writers the ability to claim the universal lock thereby allowing current reader threads in the critical section to finish their job and all the while preventing more readers to enter the critical section until after the writer can receive their writer-lock and finish with their job. I will now show two examples of how this code preforms, or more specifically, I will show how this code allows writers to not starve and vice versa:

Original Code (From book)

```
Scenario Starts:
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Done 7
Reader Done 6
Reader Done 5
Reader Done 4
Reader Done 3
Reader Done 2
Reader Done 1
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
```

Scenario

rwrrrrwrr

My Code

```
Scenario Starts:
Reader Is Reading
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Done 4
Reader Done 3
Reader Done 2
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Is Reading
Reader Done 2
Reader Done 1
```

Original Code (From book)

```
Scenario Starts:
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Done 5
Reader Done 4
Reader Done 3
Reader Done 2
Reader Done 1
Writer Is Writing
Writer Done
```

Scenario

wwrrrrwr

My Code

```
Scenario Starts:
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Done 4
Reader Done 3
Reader Done 2
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Done 1
```

As seen in the above examples, the following algorithm does allow for a fair system where both readers and writers can get access to the critical section. All the while not limiting the number of concurrent readers able to access the critical section at once.

Pseudocode

```
Function Main()
        Open file
        Read characters from file
        If ( char == 'r')
                Create reader thread for function "reader"
        If ( char == 'w')
                Create writer thread for function "writer"
        Close file
        Stop threads
        Return
End Main

Function Reader()
        Acquire Universal lock
        Acquire reader lock
        Release Universal lock
        Access critical section
        Release reader lock

End Reader

Function Writer()
        Acquire Universal lock
        Acquire writer lock
        Access critical section
        Release writer lock
        Release Universal lock
End Writer
```

Conclusion

The result of this algorithm allows for a fair system where both reader and writer threads can access data when requested, however, with that said, this version of the reader-writer problem would lead to slower performance in theory. This is because fewer reader threads will be able to work concurrently due to writer threads continuously interrupting them so they can gain access to the critical section instead. This can be shown in the given example down below:

Original Code (From book)

```
Scenario Starts:
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Is Reading
Reader Done 7
Reader Done 6
Reader Done 5
Reader Done 4
Reader Done 3
Reader Done 2
Reader Done 1
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
```

Scenario

rwrwrwrwrwrwr

My Code

```
Scenario Starts:
Reader Is Reading
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Done 1
Writer Is Writing
Writer Done
Reader Is Reading
Reader Done 1
Writer Is Writing
Writer Done
Writer Is Writing
Writer Done
Reader Is Reading
Reader Is Reading
Reader Done 2
Reader Done 1
```

The code output on the left shows all the readers concurrently reading the critical sections data while the code on the right shows how the readers cannot achieve that due to numerous writer thread interruptions. Although the fair solution to the reader-writer problem is slower, it comes down to what your program requires, as not one solution solves all problems.