Assignment 3 Reader/Writer Problem - Ahmad Sibai

Introduction:

The problem this project aims to solve 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 performed on a data structure that will change it, 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 so you can have multiple at the same time. While with writer threads, only one writer can access the critical section at once. 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.

Description/Solution:

To solve this problem, I decided to use a universal semaphore so that when a reader or writer would attempt to access the critical section to read or write, they would need to acquire it if it was not locked. It enabled the readers to access a data structure while also giving writers the availability to claim the lock if it was not already obtained so that they can enter the critical section and edit it, this also meant that if one writer had the lock, no other readers or writers would be able to access the critical section. Using this solution in my code enabled me to avoid any starvation from occurring which meant that I could have multiple readers accessing the file and reading from it, but the minute a writer would require access, he would be the only one allowed to acquire the lock and therefore the only one at that time accessing the critical section.

Pseudocode:

Readerwriter.h:

```
Initialized struct

Initialized different functions involving the semaphore

Initialized read and write threads
```

Main.c:

```
    Initialized necessary variables

    Opened scenarios.txt

    Began to read from scenarios.txt

        If character is a r, the read thread was called

        Else if character is a w, the write thread was called

Close file and exit threads
```

Readerwriter.c:

```
Function to initialize rw lock structure

Function so read can acquire lock

Function so read can release lock

Function readThread

    Acquire the universal lock
    Acquire the reader lock
    Release the universal lock
    Access critical section
    Release reader lock
End of read thread
Function writeThread
    Acquire the universal lock
    Acquire the writer lock
    Access critical section
    Release writer lock
    Release the universal lock
End of write thread
Function reading_writing which wastes time to act like critical section is being accessed
```

Conclusion:

To conclude, this solution was perfect for solving the problem as threads were able to communicate with each other by having the same semaphore, the universal one. Ultimately, this prevents starvation in that it enables multiple reads to access the critical section at the same time to read, but when a writer wants to overwrite or edit a file, then they are the only ones accessing it at that time so that readers aren't reading data that is being changed. I spent 10-12 hours working on this project spread out over a few days and I was able to get ideas on how to solve this from the textbook.