Thomas Bivins

3/26/2022

**Project 3 Report**

***Introduction***

The overall purpose of this project is to implement a non-starving and fair approach to the Readers/Writers Problem. The importance of solving this task goes more in depth than just the scope of this project and involves implementing a solution that also solves the classical issue of multiple threads running concurrently while also accessing a shared resource in a fair and organized manner. The main key to solving this problem is ensuring mutual exclusion to areas of the code where it is important that only one thread is accessing critical data at a time.

The Readers/Writers problem involves two different thread classes: readers and writers. For the reader threads, any number of readers can be accessing the shared resource at the same time. For the writer threads, only one writer may have access to the shared resource at any given time and no readers may be present. This is to avoid multiple writers changing the resource at the same time and prevent multiple readers from reading differing values.

***Implementation***

The basic implementation of this problem involves a “lock” structure that provides mutual exclusion to critical sections of code. The classic solution to this problem involves 2 locks where one lock is common to both the reader thread and writer thread, and another lock that is specific only to the writer. This implementation however suffers from a problem; the writer can become starved where any number of readers can enter and exit the critical section without ever giving the writing thread access to the resource.

To prevent this from happening, an additional layer of protection must be introduced through another lock mechanism. This lock would also be common to both the readers and the writers and will work by signaling the writer when a reader is completed so that it may access the shared resource. Doing this will also stall any readers that may be present and force them to wait for the writer to complete their task.

Text

Description automatically generated

Figure : Basic structure of lock mechanism and helper functions

Basic implementation of each thread involves waiting for the acquisition of and releasing multiple mutex locks for both the reader and writer threads. The basic reader thread appears as follows:

READER:

Wait for lock

If readers == 1:

Wait for writeLock

Post lock

\*\*Critical Section\*\*

Wait for lock

If readers == 0:

Post writeLock

Post lock

END

While similar in its handling of the critical section, the writer thread is simpler:

WRITER:

Wait for writeLock

\*\*Critical Section\*\*

Post writeLock

END

Text

Description automatically generated

Figure : Figure 2: Lock function implementations with added solutions to fix starvation

Text

Description automatically generated

Figure : Sample output to Reader/Writers problem, where the writer thread is not starved

***Solution***

With the addition of an extra locking mechanism to control the facilitation of the writer’s access to the critical section, the program becomes much fairer to all processes and ensures access to all shared resources. While a fairer solution is apparent by adding an additional mutex lock, the scheduler ultimately determines what threads run when. An estimated 8 hours was put into this project.