Chengbin Hu
Kimberly Bursum
Operating Systems COP4600 Fall 2014

Project 4: Readers/Writers Locks

Problem Statement:

Background: Synchronization is necessary when multiple threads concurrently execute on the same resources. Semaphores are the classic solution to isolate a program's critical section and provide mutual exclusion to avoid deadlock and race conditions. A typical use of Semaphores allows only one thread at a time to access a program's critical section. These locks restrict access to data structures used by Readers and Writers which allow a) multiple Reader threads with all Writer threads locked out or b) one Writer thread at a time and all Readers and other Writers locked out. The first Reader into the critical section acquires the Writelock and any number of subsequent Readers also may access the critical section triggering a counter as they come in. As the Readers leave the critical section the counter is decremented and, when all Readers are finished and out, one Writer may acquire the Writelock and access the critical section.

There are four classic Readers/Writers functions: 1. acquire_readlock, 2. release_readlock, 3. acquire_writelock, and 4. release_writelock. In the aquire_readlock function, a short-duration lock is put around the incrementation of a Reader counter variable, the counter variable is incremented, then a test causes the first Reader only to acquire the Writelock. This prohibits any Writers from entering the critical section. The short-duration lock is released allowing other Readers to acquire the short-duration lock and increment the Reader counter variable. In the release_readlock function, another short-duration lock is put around the decrementation of the Reader counter variable and a test causes the last Reader in the critical section to release the Writelock on his way out. The acquire_writelock and release_writelock functions are simply that.

Problem description: Because an unlimited number of Readers keep incrementing the Reader counter variable, as long as one Reader remains in the critical section no Writer may enter and Writer starvation may occur.

Solution:

Theory behind our solution: We flip the classic solution around to favor the writers. To solve the problem of Writer starvation our theory is to make the Readers yield to the Writers by counting the number of Writers and making Readers sleep until they are woken by the last Writer on his way out of the critical section. The Writers have the responsibility of waking up the Readers when they (the Writers) are done. While this solution might result in some Reader starvation, we believe it is a solid solution to the given task of preventing Writer starvation because in the general case there are many more Readers than Writers and Readers can still run concurrently in the critical section.

Details of our solution implementation: We create a Write counter variable we called “write”. We create a new condition variable we called “writejoin” and a semaphore short-duration lock called “writeonly”. In the acquire_readlock function we replace the semaphore with a mutex and a condition variable. We add a while loop that , as long as there is a Writer in the critical section, make the Readers wait in a conditional sleep until the condition “writejoin” is triggered (elsewhere) which will broadcast a signal for them to wake up and check the lock. After they wake up, the Readers continue, as in the classical function, to increment the Reader counter variable, give the Writelock to the first Reader and proceed to the critical section. As each Reader enters acquire_readlock it will check if there is a Writer waiting via the while loop and, if there is, it will yield to it and fall into conditional sleep. In release_readlock, we replace the classic functions with their mutex equivalents. In acquire_writelock, we use a new short-duration lock called “writeonly” around the incrementation of a new Writer counter variable “writer”. In release_writelock, we also use the new “writeonly” short-duration lock around the decrementation of the new Writer counter variable. Here we create the condition “writejoin”: if there are zero waiting Writers, then a broadcast can signal and wake up any waiting Readers. As Writers finish in the critical section they call release_writelock which triggers the condition which wakes up the Readers.

Demonstration of improved code showing no starvation: We created a test using groups of Readers and Writers with known run times. We have set up all the Writers to take approximately double the amount of time than the Readers take. We run the short-run-time Readers and the long-run-time Writers at the same time. The Writers finish BEFORE the Readers demonstrating that the Readers yielded and let the Writers run. We also run a group of quick Readers then a group of slow Writers then a group of quick Readers. The slow Writers still finish before most of the quick Readers showing that new Readers also have to wait for the Writers. There is no Writer starvation here.

Breakdown of Group Effort:
We both spent approximately 16 hours each working together and separately on all sections of the project. We both did problem analysis, research and theory of the solution. Chengbin did more of the coding and Kimberly did more of the report writing.

References: “Concurrent Control with Readers and Writers”, P.J. Courtois et al., Commun. of the ACM, vol. 14, No. 10., pg. 667-8, Oct. 1971.