May 18, 2021

CSN-232 Starve Free Reader Writer Problem Assignment #6

Utkarsh Mishra

17112085

1 Introduction

A classical Reader-Writer Problem is a situation where a data structure can be read and modified simultaneously by concurrent threads. Only one Writer is allowed access the critical area at a moment. When no Writer is active any number of Readers can access the critical area. There are at least three variations of the problems, which deal with situations in which many concurrent threads of execution try to access the same shared resource at one time. The basic reader–writers problem was first formulated and solved by Courtois et al in 1971.

2 The First Readers- Writers Problem

This is also called readers-preference as the preference is given to the readers. It operates as follows: Before entering the critical section, every new reader must go through the entry section. However, there may only be a single reader in the entry section at a time. This is done to avoid race conditions on the readers (in this context, a race condition is a condition in which two or more threads are waking up simultaneously and trying to enter the critical section; without further constraint, the behavior is non deterministic. E.g. two readers increment the read count at the same time, and both try to lock the resource, causing one reader to block). To accomplish this, every reader which enters the will lock the for themselves until they are done with it. At this point the readers are not locking the resource. They are only locking the entry section so no other reader can enter it while they are in it. Once the reader is done executing the entry section, it will unlock it by signalling the mutex. Signalling it is equivalent to: mutex.V() in the above code. Same is valid for the . There can be no more than a single reader in the exit section at a time, therefore, every reader must claim and lock the Exit section for themselves before using it. Once the first reader is in the entry section, it will lock the resource. Doing this will prevent any writers from accessing it. Subsequent readers can just utilize the locked (from writers) resource. The reader to finish last (indicated by the read count variable) must unlock the resource, thus making it available to writers.

In this solution, every writer must claim the resource individually. This means that a stream of readers can subsequently lock all potential writers out and starve them. This is so, because after the first reader locks the resource, no writer can lock it, before it gets released.

3 The Second Readers–Writers Problem

The first solution is suboptimal, because it is possible that a reader R1 might have the lock, a writer W be waiting for the lock, and then a reader R2 requests access. It would be unfair for R2 to jump in immediately, ahead of W; if that happened often enough, W would starve. Instead, W should start as soon as possible. This is the motivation for the second readers–writers problem, in which the constraint is added that no

writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is also called writers-preference.

In this solution, preference is given to the writers. This is accomplished by forcing every reader to lock and release the readtry semaphore individually. The writers on the other hand don't need to lock it individually. Only the first writer will lock the readtry and then all subsequent writers can simply use the resource as it gets freed by the previous writer. The very last writer must release the readtry semaphore, thus opening the gate for readers to try reading.

No reader can engage in the entry section if the readtry semaphore has been set by a writer previously. The reader must wait for the last writer to unlock the resource and readtry semaphores. On the other hand, if a particular reader has locked the readtry semaphore, this will indicate to any potential concurrent writer that there is a reader in the entry section. So the writer will wait for the reader to release the readtry and then the writer will immediately lock it for itself and all subsequent writers. However, the writer will not be able to access the resource until the current reader has released the resource, which only occurs after the reader is finished with the resource in the critical section.

The resource semaphore can be locked by both the writer and the reader in their entry section. They are only able to do so after first locking the readtry semaphore, which can only be done by one of them at a time.

If there are no writers wishing to get to the resource, as indicated to the reader by the status of the readtry semaphore, then the readers will not lock the resource. This is done to allow a writer to immediately take control over the resource as soon as the current reader is finished reading.

4 The Third Readers- Writers Problem

The solution implied from both the above problems statements can result in starvation — the first one may starve writers in the queue, and the second one may starve readers. Therefore, the third readers–writers problem is sometimes proposed, which adds the constraint that no thread shall be allowed to starve; that is, the operation of obtaining a lock on the shared data will always terminate in a bounded amount of time.

4.1 Starve Free Readers Writers Solution

The classical solution of the problem results in starvation of either reader or writer. In this document, I propose a solution for the above problem with the following assumptions: i.e. Semaphore preserves the first in first out(FIFO) order when locking and releasing the processes(Semaphore uses a FIFO queue to maintain the list of blocked processes).

4.1.1 Initialization

In this solution 3 semaphores are used. turn semaphore is used to specific whose chance is it next to enter the critical section. The process holding this semaphore gets the next chance to enter the critical section. rwt semaphore is the semaphore required to access the critical section. r mutex semaphore is required to change the read count variable which maintain the number of active reader

4.2 Correctness of Solution

4.2.1 Mutual Exclusion

The rwt semaphore ensures that only a single writer can access the critical section at any moment of time thus ensuring mutual exclusion between the writers and also when the first reader try to access the critical section it has to acquire the rwt mutex lock to access the critical section thus ensuring mutual exclusion between the readers and writers.

4.2.2 Bounded Waiting

Before accessing the critical section any reader or writer have to first acquire the turn semaphore which uses a FIFO queue for the blocked processes. Thus as the queue uses a FIFO policy, every process has to wait

for a finite amount of time before it can access the critical section thus meeting the requirement of bounded waiting.

4.2.3 Progress Requirement

The code is structured so that there are no chances for deadlock and also the readers and writers takes a finite amount of time to pass through the critical section and also at the end of each reader writer code they release the semaphore for other processes to enter into critical section.