Ahmed El Maliki

Project 3 COP4600

Reader Writer Locks

Introduction:

Reader-writer problem is one of those long-lasting problems of computer science when it comes to concurrency. It relates to a file or data that is being shared between multiple processes/threads. The problem is intuitively simple, when a writer is writing in the shared memory/critical section no other writer or reader is allowed in. When a reader is reading, multiple readers are allowed, but no writer is allowed. That means exclusive privileges to the writers must be in place.

Now, let's introduce another constraint which is starvation.

We need to design a program that does not starve either readers or writers.

The most likely to happen due to the conditions of writers being only able to enter when no thread is inside the Critical section is most likely that the writers will get starved by the readers

Solution:

To solve this problem, the code from the book was used. This code only addresses privilege access, but not starvation, meaning if there are more readers than writers, the critical section might be used for a various amount of time by readers only, and writers will be left for last.

The code from the book uses 2 semaphores to control access privileges, sem_t lock is used to prevent threads from modifying the number of readers currently present in the shared memory, sem_t writelock is there to prevent any writing thread from entering the critical section if another thread is present there.

In our solution we introduce another semaphore to prevent starvation it is called sem_t extracontrol.

This semaphore basically prevents writer starvation, this semaphore is added to both readers and writers, but only gets unlocked when the writer is done writing.

To elaborate more, if a writer arrives and a reader or multiple readers are already in the critical section, it will lock the sem_t extracontrol and be on hold since the critical section is not empty. Now, let's say next thread is a read, that thread will not be able to require the read lock as it must wait for extracontrol which was locked by the previous thread which is writer. Once the Critical section is empty, the readers will signal the writelock and lock allowing only writer to enter critical section since the extracontrol has not been signaled yet, now once the writer enters and executes it will release both writelock and extracontrol allowing whatever next thread to enter.

Pseudo Code:

Semaphores:

Initialize semaphores to 1 and #readers

Lock<-1

Writelock<-1

extracontrol<-1

readers<-0

```
   reader{
       aquireRead()
       //critical section

       // done with critical section
       releaseRead()

}


  writer() {

       AquireWrite()
       //critical section

       // done with critical section
       releaseWrite()


}


aquireRead() {
       sem_wait(extracontrol)
       sem_wait(lock)
       readers++

       if (readers == 1)
              sem_wait(writelock)
       sem_post(lock)
       sem_post(extracontrol)

}
```

```
aquireWrite() {
	sem_wait(extracontrol)
	sem_wait(writelock)


}


releaseRead() {
	sem_wait(lock)
	readers--
	if (readers == 0)
		sem_post(writelock)



	sem_post(lock)

}


releaseWrite() {
	sem_post(writelock)
	sem_post(extracontrol)


}
```

Discussion and Conclusions:

When a writer leaves the critical section and signals extracontrol and write, the scheduler might have other threads already scheduled and there is no guarantee on which thread is getting executed next. The next writer might have to wait on readers or not, it is the wild west with the scheduler at that point.

The approach on how to solve this problem is heavily dependent on the type of frequent operations and which priority we would want to give to operations. Some application would prefer reader priority, some would prefer writer priority, some require both which is the hardest to implement in the real world.

Overall, I spent around 16 hours working on this project.

Reference:

NesoAcedemy

Textbook from the class