Synchronization, Part 7: The Reader Writer Problem

Alex Kizer edited this page on Mar 17 · 1 revision

What is the Reader Writer Problem?

Imagine you had a key-value map data structure which is used by many threads. Multiple threads should be able to look up (read) values at the same time provided the data structure is not being written to. The writers are not so gregarious - to avoid data corruption, only one thread at a time may modify (`write`) the data structure (and no readers may be reading at that time).

The is an example of the *Reader Writer Problem*. Namely how can we efficiently synchronize multiple readers and writers such that multiple readers can read together but a writer gets exclusive access?

An incorrect attempt is shown below ("lock" is a shorthand for `pthread_mutex_lock`):

```
read()
  lock(m)
  // do read stuff
  unlock(m)
```

```
write()
  lock(m)
  // do write stuff
  unlock(m)
```

At least our first attempt does not suffer from data corruption (readers must wait while a writer is writing and vice versa)! However readers must also wait for other readers. So let's try another implementation..

Attempt #2:

```
read() {
  while(writing) {/*spin*/}
  reading = 1
  // do read stuff
  reading = 0
```

```
write() {
  while(reading || writing) {/*spin*/}
  writing = 1
  // do write stuff
  writing = 0
```

Our second attempt suffers from a race condition - imagine if two threads both called `read` and `write` (or both called write) at the same time. Both threads would be able to proceed! Secondly, we can have multiple readers and multiple writers, so lets keep track of the total number of readers or writers. Which brings us to attempt #3,

```
read() {
  lock(&m)
  while (writers) {
    pthread_cond_wait(&cv,&m)
```

```
write() {
  lock(&m)
  while (readers || writers) {
    pthread_cond_wait(&cv,&m)
```

```
    }
    readers++
    // do read stuff
    readers--
    pthread_cond_signal(&cv)
    unlock(&m)
```

```
    }
    writers++
    // do write stuff
    writers--
    pthread_cond_signal(&cv)
    unlock(&m)
```

This solution might appear to work when lightly tested however it suffers from several drawbacks - can you see them? We will discuss these in a future section.