Readers/Writers Problem

Readers/Writers

- Shared database (for example, bank balances, or airline seats)
- Two classes of users:
  - Readers
    - never modify database
  - Writers
    - read and modify database
- Using a single lock on the database would be overly restrictive.
- Want:
  - many readers at a time
  - only one writer at a time

Readers/Writers

- **Constraints**
  - 1. Readers can access database when no writers
    - (Condition okToRead)
  - 2. Writers can access database when no readers or writers
    - (Condition okToWrite)
  - 3. Only one thread manipulates state variables at a time.
- The naïve approach is to allow only one thread to access the database at a time
- But multiple readers can access at a time

- State variables:
    - # of active readers -- AR = 0
    - # of active writers -- AW = 0
    - # of waiting readers -- WR = 0
    - # of waiting writers -- WW = 0
    - Condition okToRead
    - Condition okToWrite
    - Lock lock
- Acquire lock as soon as you enter the critical section
- Release lock before leaving the critical section
- If you realize that you cannot proceed inside the critical code
    - Wait

- Reader
  - Initial status is waiting
  - wait until no writers
  - Change status to active reader
  - access database
  - No more active
  - check out -- wake up waiting writer
- Writer
  - Initial status is waiting writer
  - wait until no readers or writers
  - Change status to active writer
  - access database
  - No more active
  - check out -- wake up waiting readers or writer

```
Reader() {
    lock.Acquire();
    WR++;
    while (AW > 0) { // check if safe to read
                             // if any writers, wait
                okToRead.Wait(&lock);}
    WR--;
    AR++;
    lock.Release();
    Access DB
    lock.Acquire();
    AR--;
          if (AR == 0)//if no other readers still
                        // active, wake up writer
                okToWrite.Signal(&lock);
    lock.Release();
}
```

```
Writer() { // symmetrical
lock.Acquire();
WW++;
while ((AW + AR) > 0)// check if safe to write
    // if any readers or writers,wait
    okToWrite->Wait(&lock);
WW--;
AW++;
lock.Release();
Access DB
// check out
lock.Acquire();
AW--;
    okToRead->Broadcast(&lock);
    okToWrite->Signal(&lock);
lock.Release();
}
```

Readers/Writers

- **Constraints**
  - 1. Readers can access database when no writers
    - (Condition okToRead)
  - 2. Writers can access database when no readers or writers
    - (Condition okToWrite)
  - 3. Only one thread manipulates state variables at a time.
  - Waiting/Active Writers should be given priority over the readers

```
Reader() {
    lock.Acquire();
    WR++;
    while (AW > 0) { // check if safe to read
                            // if any writers, wait
            okToRead.Wait(&lock);}
    WR--;
    AR++;
    lock.Release();
    Access DB
    lock.Acquire();
    AR--;
    if (AR == 0)//if no other readers still
                       // active, wake up writer
            okToWrite.Signal(&lock);
    lock.Release();
}
```

What if there’s always an AR
Waiting Writer will starve
So give priority to Waiting Writer

```
Reader() {
    lock.Acquire();
    WR++;
    while (AW > 0 || WW > 0) {
                             // check if safe to read
                           // if any writers, wait
                okToRead.Wait(&lock);}
    WR--;
    AR++;
    lock.Release();
    Access DB
    lock.Acquire();
    AR--;
    if (AR == 0 && WW > 0)
                okToWrite.Signal(&lock);
    lock.Release();
}
```

```
Writer() { // symmetrical
lock.Acquire();
WW++;
while ((AW + AR) > 0)// check if safe to write
    // if any readers or writers,wait
    okToWrite->Wait(&lock);
WW--; AW++;
lock.Release();
Access DB
// check outs
lock.Acquire();
AW--;
If(WW > 0)
    okToWrite->Signal(&lock);
else if (WR > 0)
    okToRead->Broadcast(&lock);
lock.Release();
}
```