CSCI_4100_OS_10_10_2018.md

Midterm talk

- Generally everything is fair game
- There will be no coding q's on the exam
- Matching and Multi-choice ONLY
- There are ALOT of them

Quiz talk

- Paraphrasing becomes too much

RWLock operations

- start read - block caller until safe to read, then acquires the lock
- done read - used by the reader to release the lock
- start write - block caller until safe to write
- done write - used by writer to release the lock

1. Use a lock to guard the internal state of the readers writers LOCK

    1. The internal state? better to think of invariance
    2. A Writer can only write if there are no readers reading

    - Leads to 2 peices of info.

        - Number of active reader threads
        - Number of active writer threads (Never should go above 1)

    - Not only do we keep track of active reader and writer threads.

    - We keep track of waiting reader and writer threads

        - Number of waiting reader threads
        - Number of waiting writer threads

    - These lock are about **mutual exclusion** NOT sequencing

2. All operations start by aquiring the lock

3. Id and add all contion variables

    - readGo - safe to read
    - writeGo - safe to read

4. Add Loops to wait in startRead and startWrite

5. Add code to signal/broadcast

---------------- startRead ----------------

```
void RWLock::startRead() {
  lock_aquire();

  waitingReaders++; // starting to wait

  // wait here until safe
  while( !safeToRead() ){
    readGo.wait();
  }

  waitingReaders--; // done waiting, tell PC one less waiter
  activeReaders++; // tell we have 1 more readers

  lock_release(); // RELEASE ; data is consistent
}
```

---------------- doneRead ----------------

```
void RWLock::doneRead() {
  lock_aquire();

  activeReaders--; // simple, one less readers

  // if safe to write, tell the writer it can start
  if ( safeToWrite() ) {
    writeGo.signal(); // broadcast out that it is now safe to write
  }

  lock_release();
}
```

---------------- startWrite ----------------

```
void RWLock::startWrite() {
  lock_aquire();

  waitingWriters++; // one more writer waiting

  // wait until safe
  while( !safeToWrite() ){
    writeGo.wait();
  }

  waitingWriters--; // done waiting

  activeWriters++; // now actively writing
  lock_release();
}
```

---------------- doneWrite ----------------

```
void RWLock::doneWrite() {
  lock_aquire();

  activeWriters--; // simple, one less readers

  if ( safeToWrite() ) {
    // here is the important part
    writeGo.signal(); // signal out that it is now safe to write MORE
  }
  if ( safeToRead() ) {
    // then tell all readers it is safe to read
    readGo.broadcast(); // broadcast out that it is now safe to read
  }

  lock_release();
}
```

---------------- safeToRead ----------------

```
bool safeToRead () {
  return activeWriters == 0 &&
         waitingReaders == 0;
}
```

---------------- safeToWrite ----------------

```
bool safeToWrite () {
  return activeReaders == 0 &&
         activeWriters == 0;
}
```