Lecture 8: Reader/Writer Locks

Goal: walk through an example synchronization problem, found in many operating system kernels, that can illustrate the various aspects of locks and condition variables.

Illustrate method for writing correct synchronization code.

Problem statement:

Shared data, accessed by multiple threads. Very common in databases (e.g., at Amazon, many more queries about books than purchases of books, so data for how many books are left could be protected by a reader /writer lock). But also found in operating systems: linux is converting to use RCU locks in the kernel, which are a kind of reader/writer lock.

Two classes of threads:
Readers -- never modify shared data
Writers -- read and modify shared data

Using a single lock on the data would be overly restrictive. Want:
many readers at same time
only one writer at same time

Constraints:

0. At most one writer can access data at same time – safety

1. Readers can access data when no writers (Condition okToRead) – progress

2. Writers can access data when no readers or writers (Condition okToWrite) – progress

3. Bounded waiting for writers, if each writer's use of the data is bounded -- progress

4. Only one thread manipulates state variables at a time. – safety

Basic structure of solution

Reader

wait until no writers
access database
check out -- wake up waiting writer

Writer
wait until no readers or writers
access database
check out -- wake up waiting readers or writer

State variables:
# of active readers -- AR = 0
# of active writers -- AW = 0
# of waiting readers -- WR = 0
# of waiting writers -- WW = 0

Condition okToRead = NIL
Condition okToWrite = NIL
Lock lock = FREE

Recall: **Condition variable**: a queue of threads waiting for something **inside** a critical section

Condition variables support three operations:

Wait() -- release lock, go to sleep, re-acquire lock
Releasing lock and going to sleep is atomic

Signal() -- wake up a waiter, if any

Broadcast() -- wake up all waiters

Code:

```
Reader() {
  // first check self into system
  lock.Acquire();
```

```
  while ((AW + WW) > 0) {   // check if safe to read
                                          // if any writers, wait
    WR++;
    okToRead.Wait(&lock);
    WR--;
    }
  AR++;
  lock.Release();

  Access DB

  // check self  out of system
  lock.Acquire();
  AR--;
  if (AR == 0 && WW > 0)//if no other readers still
                                          // active, wake up writer
    okToWrite.Signal(&lock);
  lock.Release();
}

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

  Access DB

  // check out
  lock.Acquire();
```

```
    AW--;
    if (WW > 0)   // give priority to other writers
      okToWrite.Signal(&lock);
    else if (WR > 0)
      okToRead.Broadcast(&lock);
    lock.Release();
  }
```

Questions:

1. Can readers starve?

2. Why does checkRead need a while?