CMSC 433
Programming Language Paradigms and Technologies
William Pugh
March 9th
Midterm

- HTTP, GET vs. POST
- HTML forms and HTTP parameters
- SQL injection, XSS and CSRF
  - Referer HTTP header
- Parsing
  - grammars
  - actions
Midterm: concurrency

• what synchronization does:
  • mutual exclusion
  • atomicity - the most common mistakes involve atomicity
  • visibility
• synchronized/wait/notifyAll (or Locks and Conditions)
• AtomicX classes
concurrency

- Threads, executors, threadpools, Runnables, Futures
- Concurrent collections
  - difference between a ConcurrentHashMap and a synchronized HashMap
  - CopyOnWriteArrayList
  - concurrent Queues and BlockingConcurrentQueues
Concurrent abstractions

- CountDownLatch
- Semaphore
Project 5

• three approaches
  • subtle and elegant, using a ReadWriteLock and Locks
  • Straight forward application of wait and notifyAll
  • or Condition, signal and await
• dubious approach involving keeping a collection of component locks and iterating through them
  • avoid
P5 Warm up

• Implement our own ReadWriteLock
• Many similarities to CompoundLock
ReadWriteLocks

- Either one thread holds a write lock, or
- any number of threads can hold read locks
- We aren't going to worry about reentrant locks
The state of a read/write lock

- What is the state of a read/write lock?
  - abstractly, when it is between operations, what information would you need to know about a read/write lock

- How could that state be represented?
Operations on state

• Each operation on a read/write lock needs to atomically perform do one of the following things:
  • fail
  • wait
  • proceed
Quick failure

• Look at the state and decide that things are not in the right situation to proceed
• immediately return a value indicating failure
• without making changes to the state
waiting

- Look at the state and decide that things are not in the right situation to proceed
- wait until we are notified that things may have changed
- after we are notified, don't just assume things are in the right situation. Check. We may need to wait again
Proceed

• If things are in the right state, change the state of the object

• if this might allow another thread to proceed, notify them
Must be done atomically

- All of these inspections and changes must be done atomically
- have something to synchronize on so that a thread has exclusive access to the state of the object
- Make sure everyone that is inspecting the same state is synchronizing on the same object
wait and notifyAll

• call wait to wait to be notified
  • must be synchronized on the object you call wait on
  • thread gives up lock while waiting
• call notifyAll to wake up all threads waiting on same object
  • waking threads must reacquire lock to proceed
  • thread performing notifyAll must hold lock while notifying
class MyReadWriteLock implements RWLock {
    ... fields for state of MyReadWriteLock...
    // lock protecting those fields
    final Object lock = new Object();
    Lock getWriteLock() {
        return new SimpleLock() {
            public void lock() { ... }
            public boolean tryLock() { ... }
            public void unlock() { ... }
        }
    }

    Lock getReadLock() { ... }
}
In class pairing

• If you are done with the project, and/or are confident you understand how to implement your own read/write lock, raise your hand

• pair with someone who doesn't have their hand raised