Finding Bugs in Concurrent Software

CMSC 433

Fall 2006

Options

- Static analysis
- Normal testing
  - hard to set up tests, results not reproducible
- stepwise controlled testing
- Dynamic data race detection

One Place Buffer

```
public class OnePlaceBuffer<T> {
    T value;
    synchronized public void put(@NonNull T v) {
          if (v == null) throw new NullPointerException();
          while (value != null) wait();
          value = v;
          notifyAll();
    }
    synchronized public @NonNull T get() {
          while (value == null) wait();
          T result = value;
          value = null;
          notifyAll();
          return result;
          }
    }
```

Testing

- At a minimum, we want full branch coverage
  - we probably want to test situations while while loops execute 2+ times as well

Multithreaded test case

```
class MyTest extends MultithreadedTestCase {
      OnePlaceBuffer<String> buf;
       public initialize() {
          buf = new OnePlaceBuffer<String>();
          }
       public void thread1() {
          buf.put(“a”);
          }
       public void thread2() {
          assertEquals(“a”, buf.get());
          }
      }
```

```
class MyTest extends MultithreadedTestCase {
        OnePlaceBuffer<String> buf;
        public void initialize() {
            buf = new OnePlaceBuffer<String>();
          }
         public void thread1() {
            buf.put(“a”);
            }
         public void thread2() {
            Thread.sleep(1000);
            buf.put(“b”);
            }
         public void thread3() {
            Thread.sleep(2000);
             assertEquals(“a”, buf.get());
          }
         public void thread4() {
            Thread.sleep(3000);
             assertEquals(“b”, buf.get());
          }}
```

What to do?

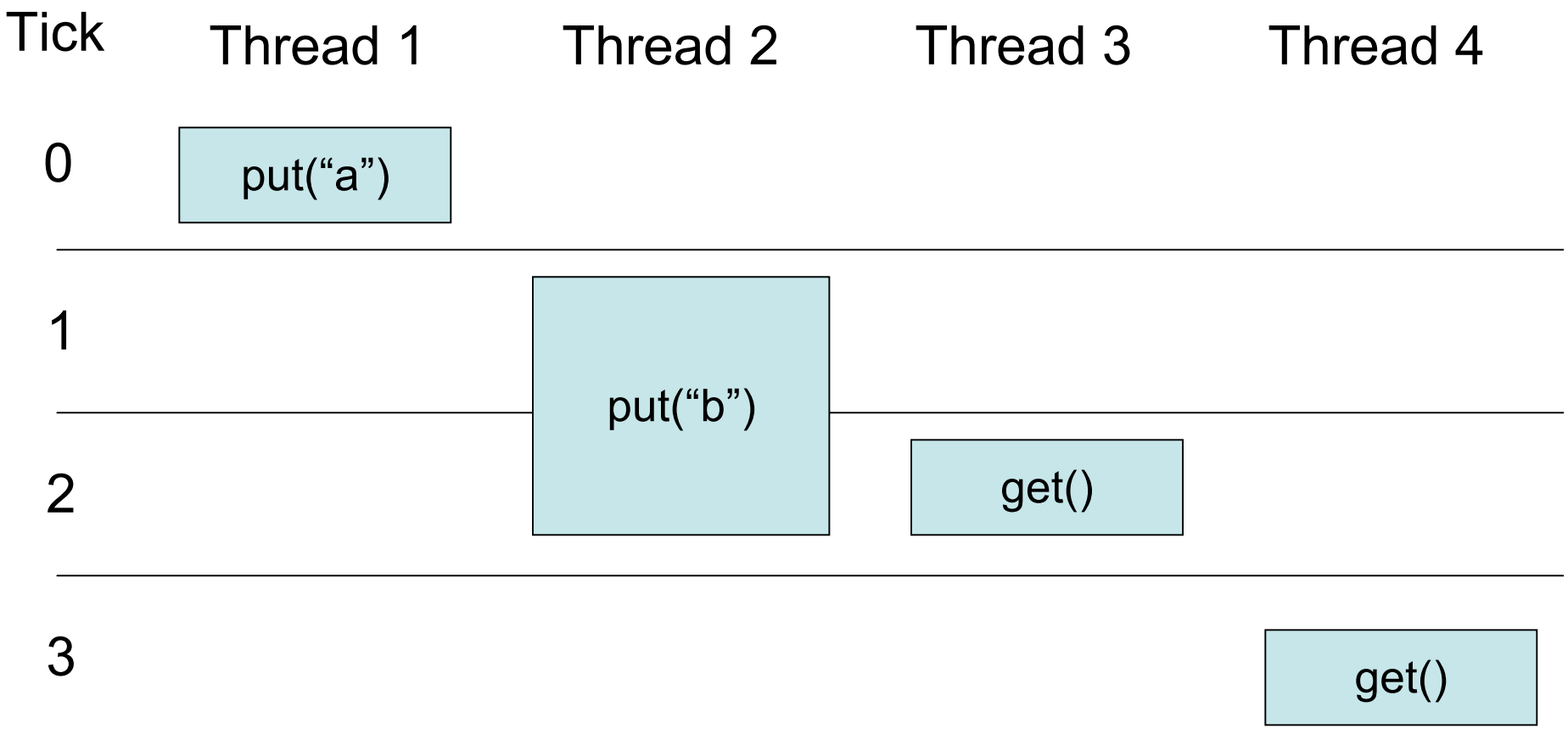
- Making code depend upon sleep and timing is yucky
  - doesn’t scale
  - doesn’t work in debugger
  - doesn’t always work
- Introduce new idea
  - a metronome timer for thread test cases

Metronome timer

- Timer starts at zero
- Advances only when all threads are blocked
- Threads can wait until the timer reaches a particular value
- Threads can query the current value of the timer

One place buffer example

- Let’s figure out the timing/ticks for a test case where two threads try to put something before any thread tries to remove something


Tick
Thread 1
Thread 2
Thread 3
Thread 4
0
put(“a”)
1
put(“b”)
2
get()
3
get()

Running a test case

- TestFramework.runOnce(
  new MyMTTestCase())
- will run the test case once
- TestFramework.runManyTimes(
  new MyMTTestCase(),42)
- will run the test case 42 times

More details on setting up a multithreaded test case

- initialize() method is executed
- all void thread…() methods are run, each in a separate thread
- finish() method is executed

- Class extends junit Assert class, so all the assertXXX(…) methods are available

Testing with metronome

```
class MyTest extends
    MultithreadedTestCase {
OnePlaceBuffer<String> buf;

public void initialize() {
  buf = new OnePlaceBuffer<String>();
}

public void thread1() {
  buf.put("a");
  assertEquals(0,getTick());
}

public void thread2() {
  waitForTick(1);
  buf.put("b");
  assertEquals(2,getTick());
}
```

```
public void thread3() {
  waitForTick(2);
  assertEquals("a", buf.get());
  assertEquals(2,getTick());
}

public void thread4() {
  waitForTick(3);
  assertEquals("b", buf.get());
  assertEquals(3,getTick());
}}
```

Runtime tools

- You can use various tools to perform dynamic instrumentation and testing for concurrency:
  - data race detection
  - introduce additional thread interleaving

Preventing Data Races

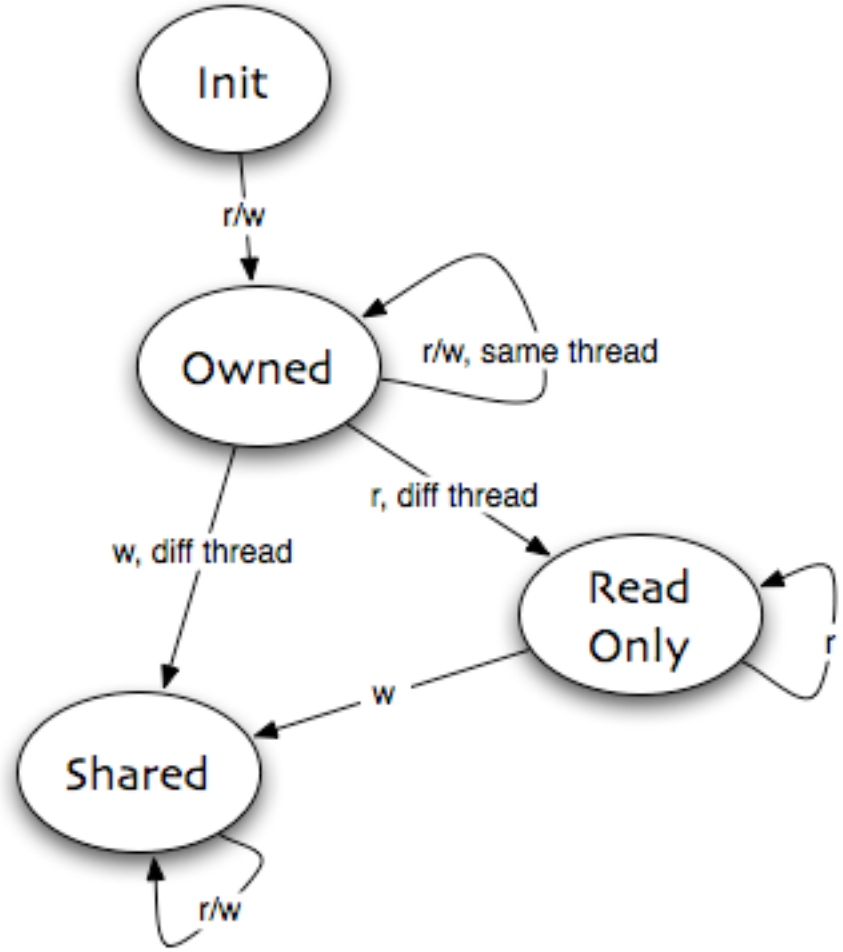
- One programming technique to prevent races:
  - Ensure that for every shared field x, there is some lock L such that no thread accesses x without holding lock L
- Note: This is not the *only* way to avoid races
  - There are fancier, more complicated techniques
  - But this is what you should do for your project

Detecting Races with checkSync

- Algorithm
  - Locks_held(t) = set of locks held by thread t
  - For each shared field x, C(x) := { all locks }
  - On each access to x by thread t,
    - C(x) := C(x) ∩ locks_held(t)
    - If C(x) = ∅ then issue a warning
  - From Savage et al, “Eraser: A Dynamic Race Detector for Multithreaded Programs,” TOCS 1997

An Improvement

- Unsynchronized *reads* of a shared location are OK
  - As long as no one *writes* to the field after it becomes shared
- Track state of each field
  - Only enforce locking protocol when it becomes shared and written



CheckSync

- added a new file to your project 2 distribution:
  - lib/checkSync.jar
- add `-javaagent:lib/checkSync.jar` to your JVM arguments to use checkSync

What checkSync does

- Instruments your code to apply the eraser protocol
- throws an exception if an access violates it
  - use `-javaagent:lib/checkSync.jar=keepGoing` to force it to keep going, printing error log to sync.log
  - use `=report` to get a report on all fields printed to sync.log