Lecture 3
Basics of Concurrent Testing
System Testing

• Used to confirm functionality of systems

• Some types of testing
  – Functional
    *Does the system deliver the required features?*
  – Performance
    *Does system execute in a sufficiently timely manner?*
  – Stress
    *How does the system respond to unexpected operating conditions (failures, etc.)?*
Software Testing

• Approaches for testing software aspects of systems

• Some types of software testing
  – Unit
    Checks individual code units (e.g. classes)
  – Integration
    Checks collections of units (e.g. components)
  – Acceptance / validation
    Checks entire software system
How Much Testing?

• “Program testing can be used to show the presence of bugs, but never to show their absence!”
  – Edsger Dijkstra (1970), 1972 Turing Award winner

• When do you stop testing? Traditionally:
  when tests meet **coverage criteria**
  – “White box”: statement coverage, branch coverage, etc.
  – “Black box”: requirements coverage
Practical Aspects of Testing

• High cost
  – Rule of thumb: 50% of software project budgets
  – Largely manual

• Essential in safety-, business-critical settings

• Often seen as “boring”
  – See “largely manual”
  – Sometimes outsourced
Testing and Concurrency

• A general testing scenario
  – Devise tests
  – Run tests
  – Use failures to identify, correct bugs

• The hardest part traditionally (single-threaded applications): devising tests
  – Automated test running environments exist
  – Debuggers can be used to replay buggy tests, since applications are deterministic

• Concurrency: all parts are hard, due to nondeterminism
  – The same test can be passed, failed
  – Replaying a buggy test run is difficult
Why the Difficulties

• Testing requires executing system
• Multiple threads need processor time
• Scheduler handles distribution of processing resources among threads

• Scheduling is outside of testers’ control
  – Implication: many possible interleavings of thread actions
  – Testing needs to consider these, in addition to traditional coverage notions
Interleavings

• Given sequences $s_1, s_2$, an *interleaving* of $s_1, s_2$ is a sequence containing all the elements of $s_1, s_2$ and respecting the relative orders within $s_1, s_2$

• Example
  - Let $s_1 = a.b, s_2 = c.d$ be (two-element) sequences
  - Then
    • $a.b.c.d, a.c.b.d, c.a.d.b$ are some interleavings of $s_1, s_2$
    • $b.c.a.d$ is not an interleaving because order between $a, b$ in $s_1$ is not preserved
How Many Interleavings?

• Question

Suppose sequence $s_1$ has $n_1$ elements, $s_2$ has $n_2$. How many interleavings of $s_1$, $s_2$ are possible?

• Answer

– “$n_1 + n_2$ choose $n_1$”, i.e.

– \( \binom{n_1 + n_2}{n_1} \), i.e.

– \( \frac{(n_1 + n_2)!}{n_1!n_2!} \)
Why?

• Each interleaving of \( s_1, s_2 \) has \( n_1+n_2 \) elements
• How many ways are there of creating interleavings this long from \( s_1, s_2 \)?
  – Once the positions for \( s_1 \)'s elements are fixed, the positions of \( s_2 \) are completely determined
  – An interleaving has \( n_1+n_2 \) positions
  – Number of ways of picking the positions for \( s_1 \)'s elements is \( \binom{n_1 + n_2}{n_1} \)
Example

Recall $s_1 = a.b$, $s_2 = c.d$
- $n_1 = n_2 = 2$
- $\binom{n_1 + n_2}{n_1} = \binom{4}{2} = \frac{4!}{2!2!} = 6$, so 6 possible interleavings
- What are they?
  - a.b.c.d
  - a.c.b.d
  - a.c.d.b
  - c.d.a.b
  - c.a.d.b
  - c.a.b.d
Back to Concurrent Testing

• Threads are sequences of actions
• Different interleavings of actions can produce different results
• So
  – Nondeterminism, and
  – From previous discussion, lots of it
Example

• Recall simple “increment race”
  – Two threads, each executing following
    myShared = shared;
    myShared++;
    Shared = myShared
  – Exact action sequence depends on platform, but here is one
    read shared
    write myShared
    increment myShared
    read myShared
    write shared
  – How many interleavings?
    • $n_1 = n_2 = 5$
    • ${10 \choose 5} = \frac{10!}{5!5!} = 252$
    • So 252 different outcomes, possibly!
Testing and Interleavings

• The same test can yield different results, depending on interleavings
• Not all interleavings yield distinct results, necessarily
• During testing
  – Exercise multiple interleavings per test
  – If a bad interleaving is detected, try to recreate it using directives to scheduler
Multiple Interleavings?

- In test cases, rerun same test inside a loop!
  - Number of iterations of loop is a “judgment call” based on expected complexity of program
  - After each iteration, determine success, keep count of failures
  - Be sure to reset state of program before each iteration
- E.g. “TestRace.java” JUnit test from Lecture 2

```java
private int numRuns = 10; ...
public void test() throws InterruptedException {
    ...
    for (int i=0; i < numRuns; i++) {
        ...
        IncThread.resetShared();
        Thread t1 = new Thread (new IncThread ("t1"));
        Thread t2 = new Thread (new IncThread ("t2"));
        ...
        passes = (IncThread.getShared() == 2) ? (passes + 1) : passes;
    }
```
Recreating Bad Interleavings

• Interleavings result from scheduling decisions
• Scheduler is not under tester control
• However, Java Thread class provides some methods that can influence scheduling
  – static void sleep (int millis)
    Block for millis milliseconds
  – static void yield ()
    “Hint” to scheduler that thread can give up processor
• You can insert these statements in your code to coax threads to give up processor
  – Remember to remove these!
  – yield() is not guaranteed to do anything, so be warned