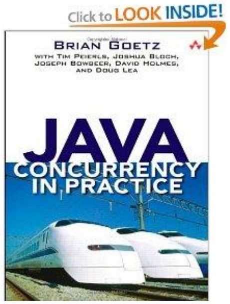
Introduction to Concurrency

4010-441
Principles of Concurrent System Design

Logistics (On myCourses)

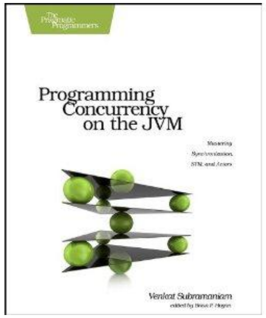


Texts

- *Java Concurrency in Practice*, Brian Goetz, et. al.
- *Programming Concurrency on the JVM*, Venkat Subramaniam.

Rough Calendar

- 4 weeks on controlling shared, mutable state + 1 exam + project
- 2 weeks on actors + 1 exam + project
- 1.5 weeks on (software) transactional memory
- 1.5 weeks on classic concurrency problems
- 1 week on other approaches to concurrency



Grading

- (5%) Preparation & Presentation
- (15%) Activities (2-3 persons for a week or less)
- (30%) Two projects (15% lowest each)
- (30%) Two in-class exams (15% each)
- (20%) Final exam in finals week

Projects

- 4-5 persons for 2+ weeks
- I'll pick the teams
- You'll have an opportunity to assess the contributions of your teammates via Peer2Peer.
- Individual grade is team grade, adjusted as appropriate based on peer evaluations.

Activities

- 2-3 persons for <= 1 week
- I'll pick the teams - sub-teams of projects where feasible
- Always traded on a 10 point basis

Preparation & Participation

- Do the readings before the specified due date
- I'll consider participation in answering questions and engaging in discussions.
- In participating in discussions, I will ***never*** penalize you for a “wrong” answer (unless you're flippant).
- Woody Allen: 80% of success is showing up.

Now on to the good stuff!

What Is "Concurrency?"

What Is "Concurrency?"

What Is a Process?

What Is "Concurrency?“

What Is a Process?

What Is a Thread?

What Is “Concurrency?”

Two events are said to be concurrent if they occur within the same time interval. Two or more tasks executing over the same time interval are said to execute concurrently. In concurrent programming, there are two basic units of execution: *processes* and *threads*.

What Is a Process?

What Is a Thread?

What Is "Concurrency?“

Two events are said to be concurrent if they occur within the same time interval. Two or more tasks executing over the same time interval are said to execute concurrently. In concurrent programming, there are two basic units of execution: *processes* and *threads*.

What Is a Process?

A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.

What Is a Thread?

What Is "Concurrency?“

Two events are said to be concurrent if they occur within the same time interval. Two or more tasks executing over the same time interval are said to execute concurrently. In concurrent programming, there are two basic units of execution: *processes* and *threads*.

What Is a Process?

A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.

What Is a Thread?

Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.

The Promises of Concurrency

- Original (OS centric processes)
  - Better resource utilization.
  - Fairness among multiple users with multiple computations.

The Promises of Concurrency

- Original (OS centric processes)
  - Better resource utilization.
  - Fairness among multiple users with multiple computations.
- Current (process centric threads)
  - Exploiting multiple processors
  - Moore's Law running out of steam (multi-core).
  - Modeling: Divide & conquer on loosely related tasks.
  - Simplify handling asynchronicity (e.g., mouse events)
  - Throughput (even on single CPU systems)
  - Responsiveness

The Perils of Concurrency

- Safety: Nothing bad happens
  - Incorrect behavior in context of concurrency
  - Race conditions
  - Memory barrier (caching)
  - Overly optimistic compiler optimizations

The Perils of Concurrency

- Safety: Nothing bad happens
  - Incorrect behavior in context of concurrency
  - Race conditions
  - Memory barrier (caching)
  - Overly optimistic compiler optimizations
- Liveness: Good things eventually happen
  - One or more threads cannot make progress
  - Deadlock

The Perils of Concurrency

- Safety: Nothing bad happens
  - Incorrect behavior in context of concurrency
  - Race conditions
  - Memory barrier (caching)
  - Overly optimistic compiler optimizations
- Liveness: Good things eventually happen
  - One or more threads cannot make progress
  - Deadlock
- Fairness: Let's share, boys and girls
  - Starvation
  - Livelock

The Perils of Concurrency

- Safety: Nothing bad happens
  - Incorrect behavior in context of concurrency
  - Race conditions
  - Memory barrier (caching)
  - Overly optimistic compiler optimizations
- Liveness: Good things eventually happen
  - One or more threads cannot make progress
  - Deadlock
- Fairness: Let's share, boys and girls
  - Starvation
  - Livelock
- Performance
  - TANSTAAFL (There Ain't No Such Thing As A Free Lunch)
  - Context switching overhead
  - Disabled compiler optimizations

The Perils of Concurrency

- Safety: Nothing bad happens
  - Incorrect behavior in context of concurrency
  - Race conditions
  - Memory barrier (caching)
  - Overly optimistic compiler optimizations
- Liveness: Good things eventually happen
  - One or more threads cannot make progress
  - Deadlock
- Fairness: Let's share, boys and girls
  - Starvation
  - Livelock
- Performance
  - TANSTAAFL
  - Context switching overhead
  - Disabled compiler optimizations
- Testing, hair-pulling, and Heisenbugs

The Perils of Concurrency

- **1985-1987 -- Therac-25 medical accelerator.** A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.
- What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured.

(Source: “History's Worst Software Bugs”, Wired)

The Ultimate Culprit - Shared, Mutable State

- Most of your development has been in imperative languages.
- The fundamental operation is assignment to change state.
  - Assignable variables are mutable.
  - May be exposed as public (bad karma).
  - May be exposed via interface methods (medium warm karma).
  - Things get tricky very fast when > 1 thread can invoke a mutating function.

The Ultimate Culprit - Shared, Mutable State

- Most of your development has been in imperative languages.
- The fundamental operation is assignment to change state.
  - Assignable variables are mutable.
  - May be exposed as public (bad karma).
  - May be exposed via interface methods (medium warm karma).
  - Things get tricky very fast when > 1 thread can invoke a mutating function.

```
public class Counter {
   private int count = 0 ;

   public void increment() {
      count = count + 1 ;
   }

   public int getCount() {
      return count ;
   }
}
```

If we call **increment()** 10,000 times and then call **getCount()**, what value is returned?

The Ultimate Culprit - Shared, Mutable State

- Most of your development has been in imperative languages.
- The fundamental operation is assignment to change state.
  - Assignable variables are mutable.
  - May be exposed as public (bad karma).
  - May be exposed via interface methods (medium warm karma).
  - Things get tricky very fast when > 1 thread can invoke a mutating function.
- Three basic approaches:
  - Make things immutable.
  - Hide shared state behind sequential access.
  - Provide mechanisms to support controlled access to shared, mutable state.

Other Issues

- Thread management
  - How many threads at one time?
  - Allocation of tasks to threads.
  - Thread scheduling.
- Higher level constructs
  - Fork / join
  - Callables & Futures
- Distributed state management
  - State consistency
  - Decision consensus