SE 552: Lecture 4

Overview

Memory model

Safety and liveness

Synchronization

Deadlock

Memory model

What is SMP? What is the memory model for SMP?

Java uses the SMP memory model. This has some weird results!

What is the possible output from this program?

```
int a = 0; int b = 0;
Thread setT = new Thread () { public void run () { a = 1; b = 1; } }
Thread getT = new Thread () { public void run () { print (b); print (a); } }
setT.start (); getT.start ();
```

0,0? 1,1? 0,1? 1,0?

Memory model

The SMP memory model allows *stale* objects to be read.

What is a stale object? How do stale objects arise?

How can a Java programmer avoid stale objects?

(Note, in common with Lea's footnote on p.95, we assume that immutable objects are never stale.)

Safety and Liveness

What is safety? What is liveness?

Is safety without liveness useful?

Is liveness without safety useful?

Why is safety difficult for multi-threaded programs?

Why is liveness difficult for multi-threaded programs?

What about immutable multi-threaded programs?

Synchronization

One strategy for mutable code:

- Synchronize all methods
- Use protection to hide fields
- No infinite loops or unbounded recursion
- All fields are initialized to obey invariants
- All methods obey invariants (if the invariant is true when the method is called, the invariant is true when the method returns).

How could we apply this strategy to UnsafeBuffer?

Does this strategy ensure safety?

Does this strategy ensure liveness?

Synchronization of iterators

Imagine we were to add a method to the Buffer class:

```
Iterator iterator ()
```

where:

```
interface Iterator {
  boolean hasNext ();
  Object next ();
}
```

How would we implement this?

How could we make it thread-safe?
(Hints: client-side locking, snapshots, fast-fail iterators).

Deadlock

The main problem with using locks is *deadlock*.

What is deadlock?

What is livelock? (Aka resource starvation.)

Deadlock

What is the Dining Philosophers problem?

How can the Dining Philosophers be coded in Java?

(Here is a sample implementation: DeadlockingPhilosopher.)

How can we avoid deadlock in the dining philosophers?
(Hint: resource ordering, or tokens.)

Summary

Java uses the SMP memory model, which has some surprising consequences!

Locks are excellent at ensuring safety, but not at ensuring liveness.

Locks often result in deadlock, for example the Dining Philosophers.

Two strategies for eliminating deadlock are resource ordering and tokens.

More on tokens next week!

Reading: Lea Chapter 2, especially 2.3 and 2.5