Thread Safety with Phaser, StampedLock and VarHandle

Dr Heinz M. Kabutz

Last Updated: 2020-11-07

Javaspecialists.eu
java training

Tweet

Heinz Kabutz
@heinzkabutz

For the easily amused

In #Java, which of these mechanisms of waiting is compatible with fibers from Project Loom?

wait();	24%
Thread.sleep(...);	18%
Thread.onSpinWait();	39%
for(;;);	18%

98 votes · 21 hours left

Phaser

Phasers

- Allows threads to coordinate by phases
  - More flexible than CountDownLatch and CyclicBarrier
- Registration
  - Number of parties *registered* may vary over time
    - Same as *count* in CountDownLatch, *parties* in CyclicBarrier
    - A party can register/deregister itself at any time
- ManagedBlocker
  - Can be used in the ForkJoinPool

Demo of Cojoining Approaches

github.com/kabutz/modern-synchronizers

branch talks-20-11-07_DevoxxUkraine

Javaspecialists.eu
java training

ManagedBlocker

- ForkJoinPool makes more threads when blocked
  - ForkJoinPool is configured with desired parallelism
- Uses in the JDK
  - Java 7: Phaser
  - Java 8: CompletableFuture
  - Java 9: Process, SubmissionPublisher
  - Java 14: AbstractQueuedSynchronizer
    - ReentrantLock, ReentrantReadWriteLock, CountDownLatch, Semaphore
  - Loom: LinkedTransferQueue, SynchronousQueue, SelectorImpl

For All You Wonderful Programmers

- **Surprise for those listening to me live today**
  - **https://tinyurl.com/dvx2020**
- **Coupon will expire once talk is over, so get it now**
  - **Lifetime access to course**

StampedLock

Javaspecialists.eu
java training

What is StampedLock?

- Java 8 synchronizer
- Allows optimistic reads
  - ReentrantReadWriteLock only has pessimistic reads
- Not reentrant
  - This is not a feature
- Use to enforce invariants across multiple fields
  - For simple classes, synchronized/volatile is easier and faster
- Can split locking and unlocking between threads

Pessimistic Exclusive Lock (write)

```
public class StampedLock {
  long writeLock() // never returns 0, might block

  // new write stamp if successful; otherwise 0
  long tryConvertToWriteLock(long stamp)

  void unlockWrite(long stamp) // needs write stamp


// and a bunch of other methods left out for brevity
```

Pessimistic Non-Exclusive Lock (read)

```
public class StampedLock { // continued ...
  long readLock() // never returns 0, might block

  // new read stamp if successful; otherwise 0
  long tryConvertToReadLock(long stamp)

  void unlockRead(long stamp) // needs read stamp

  void unlock(long stamp) // unlocks read or write
```

Optimistic Non-Exclusive Read (No Lock)

```
public class StampedLock { // continued ...
  // could return 0 if a write stamp has been issued
  long tryOptimisticRead()

  // return true if stamp was non-zero and no write
  // lock has been requested by another thread since
  // the call to tryOptimisticRead()
  boolean validate(long stamp)
```

Code Idiom for Optimistic Read

```
public double optimisticRead() {
  long stamp = sl.tryOptimisticRead();
  double currentState1 = state1,
         currentState2 = state2, ... etc.;
  if (!sl.validate(stamp)) {
    stamp = sl.readLock();
    try {
      currentState1 = state1;
      currentState2 = state2, ... etc.;
    } finally {
      sl.unlockRead(stamp);
    }
  }
  return process(currentState1, currentState2);
}
```

Code Idiom for Optimistic Read

```
public double optimisticRead() {
  long stamp = sl.tryOptimisticRead();
  double currentState1 = state1,
         currentState2 = state2, ... et
  if (!sl.validate(stamp)) {
    stamp = sl.readLock();
    try {
      currentState1 = state1;
      currentState2 = state2, ... etc.;
    } finally {
      sl.unlockRead(stamp);
    }
  }
  return process(currentState1, currentState2);
}
```

We get a stamp to use for the optimistic read

Code Idiom for Optimistic Read

```
public double optimisticRead() {
  long stamp = sl.tryOptimisticRead();
  double currentState1 = state1,
         currentState2 = state2, ... etc.;
  if (!sl.validate(stamp)) {
    stamp = sl.readLock();
    try {
      currentState1 = state1;
      currentState2 = state2, ... etc.;
    } finally {
      sl.unlockRead(stamp);
    }
  }
  return process(currentState1, currentState2);
}
```

We read field values into local fields

Code Idiom for Optimistic Read

```
public double optimisticRead() {
  long stamp = sl.tryOptimisticRead();
  double currentState1 = state1,
         currentState2 = state2, ... etc.;
  if (!sl.validate(stamp)) {
    stamp = sl.readLock();
    try {
      currentState1 = state1
      currentState2 = state2
    } finally {
      sl.unlockRead(stamp);
    }
  }
  return process(currentState1, currentState2);
}
```

Next we validate that no write locks have been issued in the meanwhile

Code Idiom for Optimistic Read

```
public double optimisticRead() {
  long stamp = sl.tryOptimisti
  double currentState1 = state
         currentState2 = state
  if (!sl.validate(stamp)) {
    stamp = sl.readLock();
    try {
      currentState1 = state1;
      currentState2 = state2, ... etc.;
    } finally {
      sl.unlockRead(stamp);
    }
  }
  return process(currentState1,
}
```

If they have, then we don't know if our state is clean

Thus we acquire a pessimistic read lock and read the state into local fields

Code Idiom for Optimistic Read

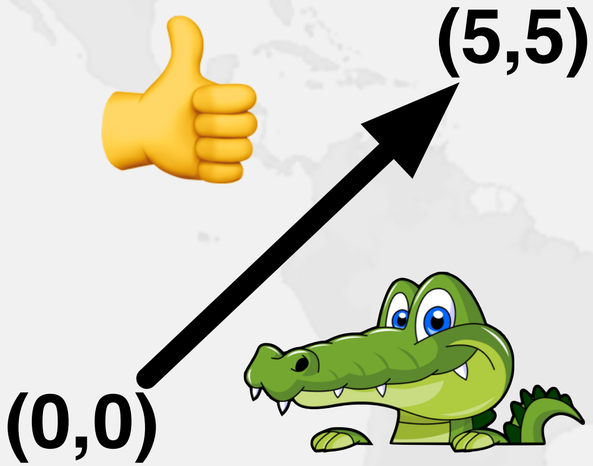
```
public double optimisticRead() {
  long stamp = sl.tryOptimisticRead();
  double currentState1 = state1,
         currentState2 = state2, ... etc.;
  if (!sl.validate(stamp)) {
    stamp = sl.readLock();
    try {
      currentState1 = state1;
      currentState2 = state2, ... etc.;
    } finally {
      sl.unlockRead(stamp);
    }
  }
  return process(currentState1, currentState2);
}
```

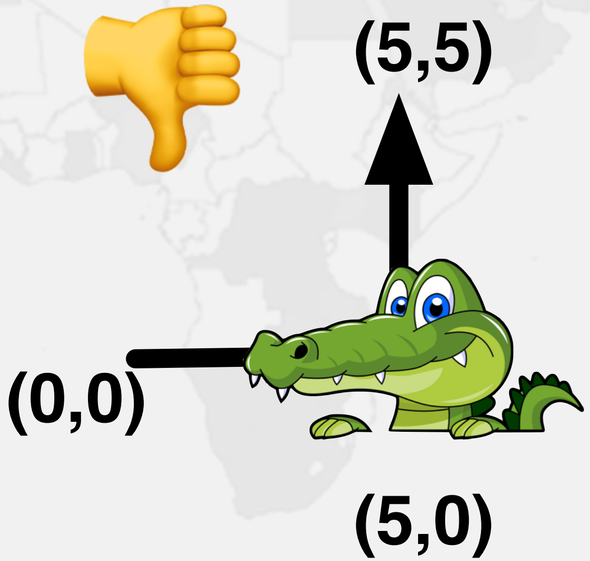
Sifis the Cretan Crocodile (RIP)

- Poor critter was roaming around Crete
  - Pet grew too big
  - Or hungry
- Eventually died in our cold winter months

Introducing the Position Class

- When moving from (0,0) to (5,5), we want to travel in a diagonal line
  - Don't want to ever see our position at (0,5) or especially (5,0)





Refactoring Position

github.com/kabutz/modern-synchronizers
branch talks-20-10-24_JConfPeru

Newer Idiom for Optimistic Read

```
public double distanceFromOrigin() {
  long stamp = sl.tryOptimisticRead();
  try {
    retryHoldingLock: for (;; stamp = sl.readLock()) {
      if (stamp == 0L) continue retryHoldingLock;
      // possibly racy reads
      double currentState1 = state1;
      double currentState2 = state2; // etc.
      if (!sl.validate(stamp))
        continue retryHoldingLock;
      return process(currentState1, currentState2);
    }
  } finally {
    if (StampedLock.isReadLockStamp(stamp))
      sl.unlockRead(stamp);
  }
}
```

Truly Optimistic, Optimistic Read

```
public double distanceFromOrigin() {
  long stamp = sl.tryOptimisticRead();
  try {
    retryHoldingLock: for (;; stamp = sl.readLock()) {
      if (stamp == 0L) continue retryHoldingLock;
      // possibly racy reads
      double currentState1 = state1;
      double currentState2 = state2; // etc.
      if (!sl.validate(stamp))
        continue retryHoldingLock;
      return process(currentState1, currentState2);
    }
  } finally {
    if (StampedLock.isReadLockStamp(stamp))
      sl.unlockRead(stamp);
  }
}
```

Truly Optimistic, Optimistic Read

```
public double distanceFromOrigin() {
  long stamp = sl.tryOptimisticRead();
  try {
    retryHoldingLock: for (;; stamp = sl.readLock()) {
      // possibly racy reads
      double currentState1 = state1;
      double currentState2 = state2; // etc.
      if (!sl.validate(stamp))
        continue retryHoldingLock;
      return process(currentState1, currentState2);
    }
  } finally {
    if (StampedLock.isReadLockStamp(stamp))
      sl.unlockRead(stamp);
  }
}
```

Refactoring Position x 2

github.com/kabutz/modern-synchronizers
branch talks-20-10-24_JConfPeru

Javaspecialists.eu
java training

VarHandle

Making your application run even faster!

Java 9 VarHandles Instead of Unsafe

- VarHandles remove biggest temptation for Unsafe
  - As fast as Unsafe
  - Make sure VarHandle fields are static final
- Can read and write fields of class
  - getVolatile() / setVolatile()
  - getAcquire() / setRelease()
  - getOpaque() / setOpaque()
  - get() / set() - plain
  - compareAndSet(), returning boolean
  - compareAndExchange(), returning found value

Refactoring Position to VarHandle

github.com/kabutz/modern-synchronizers

branch talks-20-10-24_JConfPeru

Javaspecialists.eu
java training

compareAndExchange()

- Direct support for real compare-and-swap
  - Before it was compare-and-set
- Supported by Atomic classes and VarHandles
- Eliminates one volatile read - might be faster

```
public void move(int deltaX, int deltaY) {
  int[] current, next = new int[2], swapResult = xy;
  do {
    current = swapResult;
    next[0] = current[0] + deltaX;
    next[1] = current[1] + deltaY;
  }
  while ((swapResult = (int[]) XY.compareAndExchange(
      this, current, next)) != current);
}
```

Refactoring Position to VarHandle x 2

github.com/kabutz/modern-synchronizers

branch talks-20-10-24_JConfPeru

Javaspecialists.eu

java training

Question Time

- Remember: https://tinyurl.com/dvx2020
  - Coupon valid until talk ends
- Twitter: @heinzkabutz
- Newsletter: https://www.javaspecialists.eu
- Email: heinz@javaspecialists.eu