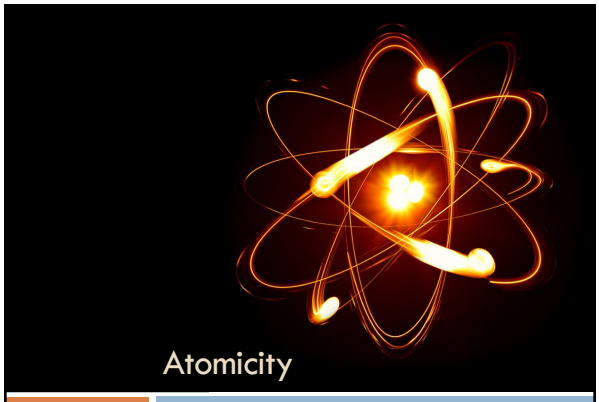


Atomicity

CS 2110 – Fall 2017

Parallel Programming Thus Far

- Parallel programs can be faster and more efficient
- Problem: race conditions
- Solution: synchronization

Are there more efficient ways to ensure the correctness of parallel programs?

Selling Widgets

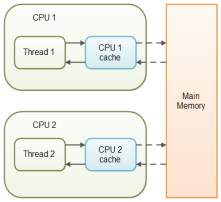
```
public class WidgetStore{
    private int numWidgets;

    /** produce widgets */
    public void produce(){…}

    /** sell all the widgets */
    public void sell(){…}

    /** diplay widget if there
    /*  are any available */
    public void display(){…}
}
```

sell() display()

display() might continue displaying widgets after all the widgets are sold!!!

Caching

- Data is stored in caches: small, fast storage units
- Only written to main memory occasionally
- Huge efficiency gains!
- Each CPU has its own cache
- Each thread maintains its own cache entries
- Huge concurrency headaches!



keyword volatile

```
public class WidgetStore{
    private volatile int numWidgets;

    /** produce widgets */
    public void produce(){…}

    /** sell all the widgets */
    public void sell(){…}

    /** diplay widget if there
    /*  are any available */
    public void display(){…}
}
```

Variables declared as volatile will not be stored in the cache. All writes will write directly to main memory. All reads will read directly from main memory.

Handling Writes

int numWidgets = 0;

Thread 1(produce)

```
numWidgets++;
```

Thread 2 (sell)

```
numWidgets--;
```

What is the value of x?

Can be either -1, 0, or 1!

Handling Writes

volatile int numWidgets = 0;

Thread 1 (produce)

```
numWidgets++;
```

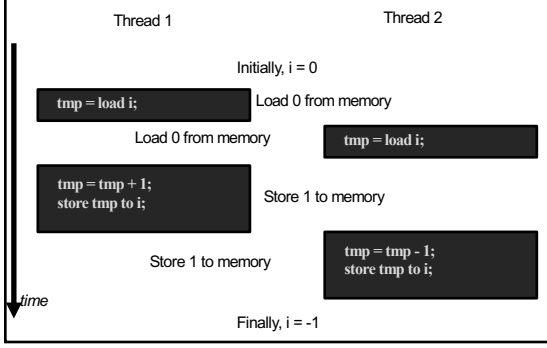
Thread 2 (sell)

```
numWidgets--;
```

What is the value of x?

Can be either -1, 0, or 1!

The Problem with Writes…



Concurrent Writes

Solution 1: synchronized

```
private int numWidgets;

public void produce(){
    ...
    synchronized(this){
        numWidgets++;
    }
    ...
}
```

- It works
- But locks can be slow

Solution 2: atomic values

```
private AtomicInteger numWidgets;

public void produce(){
    ...
    synchronized(this){
        numWidgets++;
    }
    ...
}
```

- Less powerful
- More efficient

Atomic Values

- Package java.util.concurrent.atomic defines a toolkit of classes that implement atomic values
  - atomic values support lock-free, thread-safe programming on single variables
  - class AtomicInteger, AtomicReference<E>, …
- Atomic values extend the idea of volatile
  - method get(): reads current value like volatile
  - method set(newValue): writes value like volatile
  - implements new atomic operations

Compare and Set (CAS)

- boolean compareAndSet(expectedValue, newValue)
  - If value doesn't equal expectedValue, return false
  - if equal, store newValue in value and return true
  - executes as a single atomic action!
  - supported by many processors – as **hardware instructions**
  - does not use locks!

```
AtomicInteger n = new AtomicInteger(5);
n.compareAndSet(3, 6); // return false – no change
n.compareAndSet(5, 7); // returns true – now is 7
```

Incrementing with CAS

```
/** Increment n by one. Other threads use n too. */
public static void increment(AtomicInteger n) {
       int i = n.get();
       while (!n.compareAndSet(i, i+1)) {
              i = n.get();
       }
}

// AtomicInteger has increment methods that do this
public int incrementAndGet()
public int addAndGet(int delta)
public int updateAndGet(InUnaryOperator updateFunction)
```

Locks with CAS

```
public class WidgetStore{
    private int numWidgets;

    /** produce widgets */
    public synchronized void produce(){…}
}
```

```
public class WidgetStore{
    private int numWidgets;
    private boolean lock;

    /** produce widgets */
    public synchronized void produce(){
        while(!lock.compareAndSet(false, true)){}
        …
        lock = false;
}
```

Lock-Free Data Structures

- Usable by many concurrent threads
- using only atomic actions – no locks!
- compare and swap is your best friend
- but it only atomically updates one variable at a time!

Let's look at one!

- Lock-free binary search tree [Ellen et al., 2010] http://www.cs.vu.nl//~tcs/cm/cds/ellen.pdf

More Concurrency

- Concurrency is actually an OS-level concern
  - Different platforms have different concurrency APIs
- Programming languages provide abstractions
- There are lots of techniques for write concurrent programs
  - lock (e.g., synchronized), mutex
  - atomic operations
  - semaphores
  - condition variables
  - transactional memory