Lecture 32: Concurrency 1.5

CS 62 Spring 2018

Alexandra Papoutsaki & William Devanny

Based on slides from Papoutsaki, Bruce, and Grossman

Concurrency vs Parallelism

Parallelism

Multiple tasks running at literally the exact same time

Concurrency

Multiple tasks running during overlapping time periods

One person cooking in a kitchen vs restaurant full of chefs

Sharing resources

If tasks are independent, no need to share

fork-join with simple parallel algorithms

If tasks use same memory?

we need to coordinate access!

Efficiently coordinating and synchronizing access to resources is very difficult

The challenge of concurrency

Concurrency introduces non-determinism!

The JVM can jump between separate threads and execute their lines of code in any order

The scheduling of threads can affect what each thread sees and knows about any shared resources

Race condition - undesirable behavior of a program where the output of a program is dependent on the sequencing or timing of uncontrollable events

Debugging and testing are also very difficult

Bank account example

Simple bank account class:

getBalance, deposit, withdraw

Two threads: one that withdraws and one that deposits

What is going to happen?

Bank account failure

```
withdraw(int dollars) {                 deposit(int dollars) {
1       int b = balance;
                                        2       if(dollars < 0)
                                                    throw ...;
                                        3       int b = balance;
                                        4       balance = b + dollars;
5       if(b < dollars)
            throw ...;
6       balance = b - dollars;
}                                       }
```

On both sides, b is the original balance

When line 6 executes, the deposit is overwritten!

Critical section - segment of code that accesses shared resource

Other concurrency examples

I/O is expensive, have threads use a shared cache of recent files

- What if two threads open a new file and add it to the cache at the same time?
- What if one thread tries to access a file at the same time another tries to remove it to make room in the cache?

Multi-stage task, have one thread for each stage and use a queue to hand-off results from one stage to the next

- What if enqueuer and dequeuer adjust a circular linked list at the same time?

Sharing is caring

Very common to have multiple threads accessing the same resource in an unpredictable order

Program correctness requires that synchronization be used to avoid having multiple threads in a critical section

If multiple threads can enter critical sections at the same time, the code has a race condition

Finding race conditions is very hard

- Might occur very rarely, so testing is difficult
- Have to be very thoughtful while programming

Bad bank account fix

Try to use a boolean to prevent double execution

Unfortunately only pushes off the problem

- What if both threads check the value of blocked at the same time, and then both set it to be true?

Bad bank account race condition

```
withdraw(int dollars) {

2      while(blocked) {}
3      blocked = true;
4      int b = balance;
10     if(b < dollars)
           throw ...;
11     balance = b - dollars;
12     blocked = false;
}
```

```
deposit(int dollars) {
1      while(blocked) {}
5      blocked = true;
6      if(dollars < 0)
           throw ...;
7      int b = balance;
8      balance = b + dollars;
9      blocked = false;
}
```

Mutual exclusion techniques

Several common tools exist

Synchronize blocks

- Block of code that can only be entered by one thread
- Synchronized bank account example
- Can synchronize on any object or
  make a whole method synchronized

Mutual exclusion locks (mutexes/locks)

- lock(), unlock()
- lock() blocks until the object is unlocked
- Locked bank account example

Deadlock

	Thread 1:		Thread 2:
1	lock1.lock()		
		2	lock2.lock()
3	lock2.lock()		
		4	lock1.lock()

Both threads stuck waiting for a lock that the other thread needs to unlock