Parallel & Concurrent Programming

CSCI 334
Stephen Freund
Basic Question for Us

How can programming languages make concurrent and distributed programming easier?

Shared-Memory Deterministic Parallelism

do i=1, n
    z(i) = x(i) + y(i)
endo

z(1) = 1
do i=2, n
    z(i) = z(i-1)*2
endo
Occam cobegin/end

cobegin
   x = x + 1  ||  y = y + 1
end

MATLAB parfor Loop

clear A
for i = 1:8
   A(i) = i;
end

clear A
parfor i = 1:8
   A(i) = i;
end

MATLAB parfor Loop

clear A
parfor i = 1:8
   A(i) = A(i-1)+1;
end

Fork-Join Parallelism

- Define class Worker extending Thread
  - override public void run() method
- Create object of class Worker
- Invoke o.start()

Thread States (more to come...)

- New
- Runnable
- Running
- Scheduled
- Blocked
- Done
- join
- join complete
- start
- descheduled

Self control
JVM Scheduler
External control

Why might it be bad to have many more Threads than processors???

SumArray Speedup vs. Num. Threads

Account Info

www.bank.com
Non-Deterministic Concurrency

- Concurrency Control
  - mutual exclusion
  - monitors
  - signals
  - transactions

- Communication Abstractions
  - message passing
  - Actors

Race Condition Demo

Concurrency and Race Conditions

```java
int bal = 0;

Thread 1
  t1 = bal
  bal = t1 + 10

Thread 2
  t2 = bal
  bal = t2 - 10

bal == 0
```

Concurrency and Race Conditions

```java
int bal = 0;

Thread 1
  t1 = bal
  bal = t1 + 10

Thread 2
  t2 = bal
  bal = t2 - 10

bal == -10
```

Concurrency and Race Conditions

```java
Lock m = new Lock();
int bal = 0;

Thread 1
  synchronized(m) {
    t1 = bal
    bal = t1 + 10
  }

Thread 2
  synchronized(m) {
    t2 = bal
    bal = t2 - 10
  }

```

Account Monitor [Hoare]

```java
class Account {
  private int balance;

  public synchronized void add(int n) {
    balance += n;
  }

  public synchronized String toString() {
    return "balance = " + balance;
  }
}
```

Thread States

- New
- Runnable
- Running
- Done
- Blocked
- stop or interrupted
- run() ends
- stop or interrupted
- join
- acquire
- sleep
- join complete
- lock released
- notifyAll
- sleep done
- start
- scheduled
deschedued

Self control
JVM Scheduler
External control

Producer-Consumer Buffers

- Buffer with finite size
  - Producers add values to it
  - Consumers remove values from it
- Used "everywhere"
  - buffer messages on network, OS events, events in simulation, messages between threads...

Using Buffers

class Example {
    public static void main(String[] args) {
        Buffer<Character> buffer = new Buffer<Character>(5);
        Producer prod = new Producer(buffer);
        Consumer cons1 = new Consumer(buffer);
        Consumer cons2 = new Consumer(buffer);
        prod.start();
        cons1.start();
        cons2.start();
    }
}

Producers

class Producer extends Thread {
    private final Buffer<Character> buffer;
    public Producer(Buffer<Character> b) {
        buffer = b;
    }
    public void run() {
        while (moreData()) {
            char c = next();
            buffer.insert(c);
        }
    }
}

Consumers

class Consumer extends Thread {
    private final Buffer<Character> buffer;
    public Consumer(Buffer<Character> b) {
        buffer = b;
    }
    public void run() {
        while (true) {
            char c = buffer.delete();
            System.out.print(c);
        }
    }
}
public class Buffer<T> {
    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
}

b.insert("A");
s = b.delete();
public class Buffer<T> {
    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;

    public synchronized void insert(T t) throws InterruptedException {
        while (elementCount == elementData.length) wait();
        end = (end + 1) % elementData.length;
        elementData[end] = t;
        elementCount++;
        notifyAll();
    }

    public synchronized T delete() throws InterruptedException {
        while (elementCount == 0) wait();
        T elem = elementData[start];
        start = (start + 1) % elementData.length;
        elementCount--;
        notifyAll();
        return elem;
    }
}

Safe Buffer Ops

Consumers With Handler

class Consumer extends Thread {
    private final Buffer<Character> buffer;
    public Consumer(Buffer<Character> b) {
        buffer = b;
    }

    public void run() {
        try {
            while (true) {
                char c = buffer.delete();
                System.out.print(c);
            }
        } catch (InterruptedException e) {
            // thread interrupted, so stop loop
        }
    }
}

Interrupting Threads

class Example {
    public static void main(String[] args) {
        Buffer<String> buffer = new Buffer<String>(5);
        Producer prod = new Producer(buffer);
        Consumer cons = new Consumer(buffer);
        prod.start();
        cons.start();
        try {
            prod.join();
            cons.interrupt();
        } catch (InterruptedException e) {
            System.out.println("...");
        }
    }
}

Account Monitor, redux

class Account {
    int balance;
    synchronized void add(int n) {
        balance += n;
    }
    synchronized void transfer(Account other, int n) {
        balance -= n;
        other.add(n);
    }
}

Deadlock

Thread 1  Thread 2
a.transfer(b, n)  b.transfer(a, n)

Account  Monitor

java.lang.StringBuffer

class Account {
    int balance;
    synchronized void add(int n) {
        balance += n;
    }
    synchronized void transfer(Account other, int n) {
        balance -= n;
        other.add(n);
    }
}

public final class StringBuffer {
    int count;
    char chars[];
    synchronized int length() { return count; }
    synchronized void clear() { ... }
    synchronized StringBuffer append(StringBuffer sb) {
        int len = sb.length();
        ... sb.getChars(0, len, value, count);
        ... }
}
Atomicity Demo

```
public final class StringBuffer {
  int count;
  char chars[];
  atomic int length() { return count; }
  atomic int getChars(...){ ...}
  atomic StringBuffer append(StringBuffer sb) {
    int len = sb.length();
    ...sb.getChars(0, len, value, count);
    ...
  }
}
```

java.util.StringBuffer


Atomic As a Language Feature

```
class Account {
  int balance;
  atomic void add(int n) {
    balance += n;
  }
  atomic void transfer(Account other, int n) {
    balance -= n;
    other.add(n);
  }
}
```

Pessimistic Atomicity

```
class Account {
  int balance;
  void add(int n) {
    synchronized(global_lock) {
      balance += n;
    }
  }
  void transfer(Account other, int n) {
    synchronized(global_lock) {
      balance -= n;
      other.add(n);
    }
  }
}
```

What's good? What's bad? Alternatives?