Data Structures
&
Algorithms

Subodh Kumar

(subodh@iitd.ac.in, Bharti 422)

Dept of Computer Sc. & Engg.

Parallelism

- Multiple Processes
  - with its own virtual address space
- Multiple threads within a process
  - share the same address space
  - but each has a separate execution stack
    - hence, separate local variables
  - share “heap”

Java Threads

```
Task task = new Task();
Thread thread = new Thread(task);
thread.start();
```

```
class Task implements Runnable{
    public void run() {
        ...
    }
}   public getItem() {
           synchronized
        return current++;
}
```

Two Threads Example

```
class Task implements Runnable{
    private static int value = 0;
    public synchronized void run() { value++; }
    int getValue()  { return value; }
}
```

```
public class TwoThreadExample {

  public static void main(String[] args) {
      Thread t1 = new Thread(new Task());
      Thread t2 = new Thread(new Task());
      t1.start();  t2.start();
       System.out.println(value);
   }
}
```

Two Threads Example

```
class Task implements Runnable{
    private static int value = 0;
    public synchronized void run() { value++; }
    int getValue()  { return value; }
}
```

```
public class TwoThreadExample {

  public static void main(String[] args) {
      Task t = new Task();
      Thread t1 = new Thread(t);
      Thread t2 = new Thread(t);
      t1.start();  t2.start();
      System.out.println(t.getvalue());
   }
}
```

Two Threads Example

```
class Task implements Runnable{
    private static int value = 0;
    public synchronized void run() { value++; }
    int getValue()  { return value; }
}
```

```
public class TwoThreadExample {

  public static void main(String[] args) {
      Task t = new Task());
      Thread t1 = new Thread(t);
      Thread t2 = new Thread(t);
      t1.start();  t2.start();
      t1.join(); t2.join();
      System.out.println(t.getvalue());
  }
}
```

Nested Synchronization

```
synchronized int method1(int value) {
  return value > 0?
       value * method2(value-1):1;
}
```

Re-entrant

```
synchronized int method2(int value) {
  return value < 0)?
       1:value + method1(value-1);
}
```

But what about two separate threads?

Deadlock

Class B

```
synchronized int method1(A a, int value) {
   return value > 0?
         v
}
```

Class A

```
synchroni
   return
         1
}
```

```
if(inputvalue % 0) { // Odd
      result = a.method1(b, inputvalue)
    }
} else {
      result = b.method2(a, inputvalue)
    }
}
```

Can deadlock with multiple threads

But what about two separate threads?

```
class Task implements Runnable{
    // private ObjectX, ObjectY
    public void run() {
      if(atest) { // Odd
         synchronized(objectX) {
            something();
            synchronized(objectY) {
               somethingElse();
            }
         }
      } else {
         synchronized(objectY) {
            synchronized(objectX) {
              anything();
            }
         }
      }
    }}
```

Deadlock

Memory Inconsistency

```
class Consumer implements Runnable{
    public void run() {
       while(! datavalid) {}
       consume();
     }
}
```

(Assume, `datavalid` is shared)

```
class Producer implements Runnable{
    public void run() {
       produce();;
       datavalid = true;
     }
}
```

Java Process

```
ProcessBuilder pb =
  new ProcessBuilder("command", "args");
pb.redirectOutput(
       ProcessBuilder.Redirect.INHERIT);
Process p = pb.start();
```

```
Runtime r = Runtime.getRuntime();

r.gc() .. r.freeMemory() ..
r.availableProcessors()
```