Semaphores

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu
Outline

• Semaphores
Semaphores

- **Semaphores** can restrict the number of threads that access a shared resource
  - Unlike a lock or monitor, the number of permits available on a semaphore is specified at creation and can be more than one
  - A thread must acquire one of the permits of the semaphore before executing code managed by a semaphore
- **Semaphores** are acquired and released similarly to locks, but a specified number of threads can access a resource protected by a semaphore
  - A semaphore that only allows one thread to access the resource can behave like a mutually exclusive lock
  - In addition, a thread can release permits on a semaphores even without having them, allowing a thread to create permits on a semaphore
Semaphore Example #1

```java
import java.util.concurrent.Semaphore;

public class SemaphoreTest {
    public static void main(String [] args) {
        for (int i=0; i < 100; i++) {
            MyThread mt = new MyThread(i);
            mt.start();
        }
    }
}

class MyThread extends Thread {
    private static Semaphore semaphore = new Semaphore(1);
    private int num;
    public MyThread(int num) {
        this.num = num;
    }
    public void run() {
        try {
            semaphore.acquire();
            System.out.println("Thread " + num + " starting run");
            Thread.sleep(1000);
            System.out.println("Thread " + num + " finishing run");
        } catch (InterruptedException ie) {
            System.out.println("MyThread.run IE: " + ie.getMessage());
        } finally {
            semaphore.release();
        }
    }
}
```

```java
Thread 1 starting run
Thread 0 starting run
Thread 1 finishing run
Thread 0 finishing run
Thread 3 starting run
Thread 3 finishing run
Thread 2 starting run
Thread 4 starting run
Thread 2 finishing run
Thread 4 finishing run
Thread 7 starting run
Thread 6 starting run
Thread 7 finishing run
Thread 6 finishing run
Thread 9 starting run
Thread 9 finishing run
Thread 3 starting run
Thread 5 starting run
Thread 3 finishing run
Thread 5 finishing run
Thread 8 starting run
Thread 10 starting run
Thread 8 finishing run
Thread 10 finishing run
Thread 5 starting run
Thread 7 starting run
Thread 5 finishing run
Thread 7 finishing run
Thread 9 starting run
Thread 11 starting run
Thread 9 finishing run
Thread 11 finishing run
```
Semaphore Example #2

```java
import java.util.concurrent.Semaphore;

public class SemaphoreTest {
    public static void main(String[] args) {
        for (int i=0; i < 100; i++) {
            MyThread mt = new MyThread(i);
            mt.start();
        }
    }
}

class MyThread extends Thread {
    private static Semaphore semaphore = new Semaphore(2);
    private int num;
    public MyThread(int num) {
        this.num = num;
    }
    public void run() {
        try {
            semaphore.acquire();
            System.out.println("Thread " + num + " starting run");
            Thread.sleep(1000);
            System.out.println("Thread " + num + " finishing run");
        } catch (InterruptedException ie) {
            System.out.println("MyThread.run IE: " + ie.getMessage());
        } finally {
            semaphore.release();
        }
    }
}
```