Semaphores

CSCI 201
Principles of Software Development

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

USC Viterbi
School of Engineering

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

java.util.concurrent.Semaphore	
+Semaphore(numberOfPermits: int)	Creates a semaphore with the specified number of permits. The fairness policy is false.
+Semaphore(numberOfPermits: int, fair: boolean)	Creates a semaphore with the specified number of permits and the fairness policy.
+acquire(): void	Acquires a permit from this semaphore. If no permit is available, the thread is blocked until one is available.
+release(): void	Releases a permit back to the semaphore.

Semaphore Example #1

```
1   import java.util.concurrent.Semaphore;
2
3   public class SemaphoreTest {
4     public static void main(String [] args) {
5       for (int i=0; i < 100; i++) {
6         MyThread mt = new MyThread(i);
7         mt.start();
8       }
9     }
10  }
11
12  class MyThread extends Thread {
13    private static Semaphore semaphore = new Semaphore(1);
14    private int num;
15    public MyThread(int num) {
16      this.num = num;
17    }
18    public void run() {
19      try {
20        semaphore.acquire();
21        System.out.println("Thread " + num + " starting run");
22        Thread.sleep(1000);
23        System.out.println("Thread " + num + " finishing run");
24      } catch (InterruptedException ie) {
25        System.out.println("MyThread.run IE: " + ie.getMessage());
26      } finally {
27        semaphore.release();
28      }
29    }
30  }
```

```
Thread 1 starting run
Thread 1 finishing run
Thread 0 starting run
Thread 0 finishing run
Thread 2 starting run
Thread 2 finishing run
Thread 7 starting run
Thread 7 finishing run
Thread 6 starting run
Thread 6 finishing run
Thread 3 starting run
Thread 3 finishing run
Thread 8 starting run
Thread 8 finishing run
Thread 5 starting run
Thread 5 finishing run
Thread 9 starting run
Thread 9 finishing run
```

```
Thread 0 starting run
Thread 0 finishing run
Thread 3 starting run
Thread 3 finishing run
Thread 4 starting run
Thread 4 finishing run
Thread 6 starting run
Thread 6 finishing run
Thread 9 starting run
Thread 9 finishing run
Thread 5 starting run
Thread 5 finishing run
Thread 10 starting run
Thread 10 finishing run
Thread 7 starting run
Thread 7 finishing run
Thread 11 starting run
Thread 11 finishing run
```

Semaphore Example #2

```
1   import java.util.concurrent.Semaphore;
2
3   public class SemaphoreTest {
4     public static void main(String [] args) {
5       for (int i=0; i < 100; i++) {
6         MyThread mt = new MyThread(i);
7         mt.start();
8       }
9     }
10  }
11
12  class MyThread extends Thread {
13    private static Semaphore semaphore = new Semaphore(2);
14    private int num;
15    public MyThread(int num) {
16      this.num = num;
17    }
18    public void run() {
19      try {
20        semaphore.acquire();
21        System.out.println("Thread " + num + " starting run");
22        Thread.sleep(1000);
23        System.out.println("Thread " + num + " finishing run");
24      } catch (InterruptedException ie) {
25        System.out.println("MyThread.run IE: " + ie.getMessage());
26      } finally {
27        semaphore.release();
28      }
29    }
30  }
```

```
Thread 0 starting run
Thread 1 starting run
Thread 0 finishing run
Thread 5 starting run
Thread 1 finishing run
Thread 6 starting run
Thread 5 finishing run
Thread 7 starting run
Thread 6 finishing run
Thread 8 starting run
Thread 7 finishing run
Thread 12 starting run
Thread 8 finishing run
Thread 10 starting run
Thread 12 finishing run
Thread 13 starting run
Thread 10 finishing run
Thread 14 starting run
```

```
Thread 1 starting run
Thread 0 starting run
Thread 0 finishing run
Thread 1 finishing run
Thread 3 starting run
Thread 4 starting run
Thread 3 finishing run
Thread 4 finishing run
Thread 5 starting run
Thread 2 starting run
Thread 2 finishing run
Thread 5 finishing run
Thread 11 starting run
Thread 10 starting run
Thread 11 finishing run
Thread 10 finishing run
Thread 14 starting run
Thread 13 starting run
```