Semaphores

CSCI 201L

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

HTTP://WWW-SCF.USC.EDU/~CSCI201
Outline

- Semaphores
- Program
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 behaves like a mutually exclusive lock

```java
java.util.concurrent.Semaphore

+Semaphore(numberOfPermits: int)
+Semaphore(numberOfPermits: int, fair: boolean)
+acquire(): void
+release(): void

Creates a semaphore with the specified number of permits. The fairness policy is false. Creates a semaphore with the specified number of permits and the fairness policy. Acquires a permit from this semaphore. If no permit is available, the thread is blocked until one is available. Releases a permit back to the 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();
            }
        }
    }
}
```
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();
        }
    }
}
```
Outline

- Semaphores
- Program
Consider a program on your computer that plays audio files (such as iTunes). To be able to play audio files, the program needs access to the speakers of the computer. Since there may be other sounds that need to be played at the same time iTunes is playing, the speakers cannot be implemented with a Lock. Instead, write a program that simulates the speakers being shared by up to five different programs.

In addition, provide some means for the operating system to always be able to access the speakers above the other five programs.