Java Concurrency

Introduction

- concurrent programming allows two things
  - natural expression of many algorithms
    - e.g., GUI event handling vs fetching from network
  - more efficient program execution
    - particularly on multi-processor machine
    - e.g., 3D real-time graphics

Levels of Concurrency

- instruction level
  - two (or more) machine instructions may be executed simultaneously
  - usually handled by an optimizing compiler
  - or Superscalar architecture
- statement level
  - two (or more) statements may be executed simultaneously
- unit level
  - two (or more) subroutines may be run simultaneously
  - usually provides the most payoff
- program level
  - two (or more) programs may be run simultaneously
  - usually handled by the operating system
- statement and unit level may be in language

Concurrent units

- AKA Threads, Tasks, or light-weight processes
- multiple threads of control
- units can execute simultaneously if possible
  - EG, clothes washing machine and clothes dryer
  - EG, ice cream vendor and ice cream buyer
- synchronization must prevent unintended simultaneous update to data
  - Ways to create threads in Java
  - Synchronized methods and Blocks
  - Syncronization

Java Class ResourceManager

```
class ResourceManager {
    boolean isFree[];
    ResourceManager(int numberOfItems) {
        isFree = new boolean[numberOfItems];
        for (int i=0; i<isFree.length; ++i)
            isFree[i] = true;
    }

    ....
```

Class ResourceManager.request()

```
synchronized int request() {
        while (true) {
                for (int i = 0; i < isFree.length; ++i)
                        if ( isFree[i] ) {
                                isFree[i] = false;
                                return i;
                        }
                this.wait(); // block until someone releases a Resource
        }
}
```

Class ResourceManager.release()

```
synchronized void release( int index ) {
        isFree[index] = true;
        this.notify(); // let a waiting thread run
}
```

Implementation Issues

- there is overhead associated with context switching
- run-time support for a concurrent language is called a `kernel`
- a kernel has the following pieces
  - process descriptor
  - queues of processes
  - `running` process
  - scheduler algorithm

process descriptor

- contains information about a process
  - process status (e.g., ready, running, waiting)
  - priority
  - id
  - pc, registers, etc

queues of processes

- Examples include
  - `ready` queue
  - condition queue
  - semaphore queue (one for each semaphore)
- `running` process
  - descriptor of running unit
- Scheduler algorithm
  - may use `time slicing`
  - clock interrupt causes switch to new ready

Switch process in scheduler

- save status into running
- enter running into ready queue
- move a descriptor from ready queue to running
- restore status from running