# CS601: Principles of Software Development

## Introduction to Threads.

Olga A. Karpenko

Parts of this presentations are based on the materials of Prof. Engle.

## Reading

- Deitel&Deitel book
- http://www.ibm.com/developerworks/java/tutorials/j-threads/j-threads.html
- http://tutorials.jenkov.com/java-concurrency/volatile.html

## Problem

- Multiple tasks for the computer
  - Draw & display images on screen
  - Check keyboard & mouse input
  - Send & receive data via the network
  - Read & write files to disk
  - Perform computations
  - …

## Problem

- How does computer do it all at the same time?
  - Multitasking
  - Multiprocessing

# Multitasking

- Approach:
  - Computer quickly switches between tasks
  - Tasks managed by operating system (scheduler)
- Computer seems to work on tasks concurrently
- Can improve performance by reducing the wait

## Multithreading

- Approach
  - Multiple cores
  - Computer works on several tasks in parallel
  - Performance can be improved

## Process vs Thread

- Process
  - – A program that is currently running
    (Ex: web browser)
  - – Contains at least one “thread of execution”
- Threads
  - Lightweight processes, exists within a process
  - Built-in support in Java

## Process

- Executable program loaded in memory
- Has its  own memory space
- Each process may execute a different program
  - Communicate via operating system, files, network

## Process

- Has at least one main thread
- May have additional “worker threads”
  - i.e. threads that take on a specific task, or part of a task
  - often managed by the main thread

## Thread (“lightweight process”)

- Sequentially executed stream of instructions
- Shares address space with other threads
- Has its own execution context
  - Program counter, call stack (local variables)
- Communicate via shared access to data

# Example

- Web Server
  - Must handle multiple simultaneous requests
  - Must be responsive AND efficient

  (e.g. respond quickly, finish quickly)

- Implementation: Multithreading
  - Use one thread per request?

# How to Use Multithreading

- Need a problem that can be parallelized
- Create threads to handle individual tasks
- Synchronize threads to get final results

## Issues with Multithreading

- Overhead to creating threads
- For large amounts of work, can achieve significant speedup
- Must protect access to shared data with synchronization
- Difficult to debug

## Creating Threads in Java

- Write a class that implements `Runnable`
  - Override the `run()` method – that's where you put the work the thread will do
- Create an object of that class
- Create a thread and pass the object to the constructor
- Call the start() method on the thread

## Creating Threads in Java

```
public class Task implements Runnable {
     public void run() {
          // work for the thread
     }
}

// In another class:
Thread t = new Thread(new Task());
t.start();
```

## Example

```
public class PrintTask implements Runnable {
     public void run() {
          for (int i = 0; i < 3; i++)
               System.out.println(i);
     }
     public static void main(String[] args) {
          new Thread(new PrintTask()).start();
          new Thread(new PrintTask()).start();
          System.out.println("Done");
      }
}
```

# Example

Possible outputs

- 0, 1, 2, 0, 1, 2, Done  // thread 1, thread 2, main()
- 0, 1, 2, Done, 0, 1, 2  // thread 1, main(), thread 2
- Done, 0, 1, 2, 0, 1, 2  // main(), thread 1, thread 2
- 0, 0, 1, 1, 2, Done, 2  // main() & threads interleaved

## Example

- Use join()
- See `PrintTaskExample.java`

## join()

```
public static void main(String args[]) {

    Thread t = new Thread(new RunnableTask());
    t.start();
    t.join();
    // main thread will become runnable
    // when thread t completes
}
```