Overview

- Review – what constitutes a thread
- Creating threads – general
- Creating threads – Java
- What happens if synchronization is not used?
- Assignment

What constitutes a thread?

- Instruction pointer
- Stack space (how much?)
- Registers/Local variables
- Auxiliary data
  - Exception handling stack
  - Priority
  - Interruption status
  - Thread-private data
  - …
- Thread control block (TCB)

Creating Threads - general

- What has to be supplied to create a thread?
  - Always and most important: some designation for the code it is to run
  - Stack space
  - Initial state
  - Daemon status
  - Human-readable identifier?
  - Thread group?
  - Initial priority?

Creating Threads - Java

- Built-in Thread class
- Supplying the code (1st way)
  - Subclass `Thread` and override `run()` method
    - ```
      class MyThread (Thread) {
          void run () {
              …this code can be run
        as the body of a new
        thread
        }
      }
      MyThread mt = new
        MyThread()
      ```
- At this point `mt` is a `Thread` object but no new thread of execution exists
      ```

Constructing Threads - Java

- Supplying the code (2nd way)
  - Provide an object of a class that implements `Runnable` when creating a new Thread object.
    - ```
      class MyRunnable
      implements Runnable {
          void run () {
              …this code can be run
        as the body of a new
        thread
        }
      }
      MyThread mt = new
        Thread(new MyRunnable())
      ```
- Again, `mt` is just a `Thread` object. No new executable thread.
- What has to happen?
      ```

Creating an executing thread

- `mt.start()`
  - `Thread.start()` method constructs new stack and tells JVM to call runnable's `run()` on the new stack
- What happens if you just call `mt.run()`?

Providing Initial State

- Why?
  - Want many threads that are “almost the same”
- How
  - Some approaches allow parameters to the run method, but not Java
  - In Java, pass arguments to Runnable (or Thread subclass) constructor then assign to object data fields

```
MyRunnable(int instanceNum) {
  this.instanceNum = instanceNum;
}
```

Stacksize testing Runnable

```
class StacksizeTest implements Runnable {
    int depth = 0;
    public void run() {
        try {
            doOverflow();
        } catch (StackOverflowError e) {
            System.out.println("Overflow occurred at
depth " + depth + ".");
        }
    }
    void doOverflow() { depth += 1; doOverflow(); }
}
```

Example multi-threaded code (bad)

- This example is also used in the assignment
- Let's tour a program pointing out areas that may be troublesome.

Racer

Assignment

1. Install java on the machine you'll be using for the class (Eclipse or NetBeans or BlueJ IDEs, jdk or openjdk for command line). Which one did you install?
2. Describe the machine that you are using: number of processor cores, physical memory, 64 or 32 bit architecture.
3. What is the default stack size for threads in the Java environment that you installed? (Dive into the documentation) How can it be changed? Can it be changed per-thread?
4. What does this suggest as the maximum number of threads that will run well on your machine using the default stack size? (Consider the available physical memory and how much is used by each thread)
5. Write a test driver program to run the code of the following Runnable in a new thread. How many stack frames does it succeed in creating before throwing StackOverflow when using the default stack size?
6. For Thursday, 1/17: Bring your answers and a code listing for your stack overflow test to discuss in class next time.
7. For Tuesday, 1/22: Compile and run the Race example code from www.eecs.wsu.edu/~hauser/cs580/handouts/Race.java. You will get different results on machines with different numbers of processors and different compilers. See if you can start the JVM using only one processor if you have a MP machine. Turn in a written report for both parts on the class turnin page.
8. Vary the number of iterations (class variable `limit`) until you regularly get an error (or you give up trying – don't give up too soon).

Note: Java “main” program

```
class foo {
   public static void main(String argv[])
   {
       … code for main program...
   }
}
% javac foo.java
% java foo
```

Note: forcing uniprocessor execution

```
% taskset -c 0 <command>
% taskset -c 1 java Race
```

Note that on a hyper-threaded processor actual parallelism will still occur

Next time

- Thread safety – what are the rules about accessing shared variables in Java? (Chapter 2 in the Java book).