Lecture 01:  
Processes and Threads  

Software Systems Components 2  
Behzad Bordbar  
School of Computer Science, University of Birmingham, UK
You are here for

- **Software Systems Components 2**
- Module page is in [http://www.cs.bham.ac.uk/resources/courses/java/ssc/](http://www.cs.bham.ac.uk/resources/courses/java/ssc/)
- Teaching material available from [www.cs.bham.ac.uk/~bxb](http://www.cs.bham.ac.uk/~bxb) follow the link to my “Teaching” page
Contents

- General discussion about concurrent programming
- Processes
- Threads and their relationship to processes
- Creating threads in Java
Threads in Java

How many thread in running Java program?
We all have seen

Exception in thread “main” ....

Helloworld has two thread:

- main()
- Garbage collection Thread.

GUI programs have even more threads!
Why using thread (in Java)?

Threads allow us to do more than one thing at the same time.

Main reasons for multi-threaded programming in a single processor machine:

- Better interaction with user (GUI)
- Easier programming (client/server)
- Parallel processing
Are we really doing multiple things in parallel at the same time?

No! In a single processor machine, a process shares one thread for a short time and switches to running other threads.

Multithreading is similar to *timesharing*.

Multi processor machines running multiple things at the same time. (Be careful, implementation of the JVM must also support)
A process is a program in execution.

- **address space;** list of memory locations from 0 to N, which process can write/read from:
  - execution program,
  - stack
  - program data.

- Each process there ia set of **registers:**
  - program counters, stack pointer, hardware registers
For example: Process in Unix

Processes contain information about program resources and program execution state, including:

- **Process ID, process group ID, user ID, and group ID**
- **Environment**
- **Working directory.**
- **Program instructions**
- **Registers**
- **Stack**
- **Heap**
- **File descriptors**
- **Signal actions**
- **Shared libraries**
- **Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory).**
OS handling of processes

- Periodically, OS decides to stop running of one process and start running of another
- To restart, the OS must start from the exact step that process was stopped

This change (context switching) is computationally costly.

Threads exist within these process resources and run as independent entities
Threads

Each Thread has its own

- Stack (called "runtime stack" in Java)
- Registers ("program counter")
- Scheduling properties (such as policy or priority)
- Set of pending and blocked signals
- Thread specific data

Hence, independent flow of control is accomplished.
Threads vs. processes

Because threads within the same process share resources, there is no expensive context switching.

- Changes made by one thread to shared system resources will be seen by all other threads.
- Two pointers having the same value point to the same data.
- Reading/writing to the same memory locations is possible, and therefore requires explicit synchronization by the programmer.

Your job becomes harder
Threads

- POSIX document 1003.4a (1995) that describes a threads API standard.
- **Portable Operating System Interface for uniX** (POSIX) is a family of standards specified by the IEEE to define the API (*what’s the story?*)
- Most implementations of the JVM run as a single process, but there are multiple Threads.
- Java threads are simpler, take care of their own memory management, and do not have the full generality (or overhead) of POSIX threads, but POSIX compatible
Defining and starting a thread in Java

Normally, two different objects required. One, the thread itself, knows how to execute code, and the other, the *run-object*, knows what code to execute.

In Java a thread is an object an instance of *Thread*. There are two ways to create a new thread:

1) **Extend the class** *Thread*

2) **Write a class** that implement *Runnable* interface and use it in the *Thread* constructor.
Extending Thread

Extend `java.lang.Thread` and override `run()`

```java
public class MyThread extends Thread{
  public void run(){
    // Write the code for to override run()
    System.out.println("This is my first thread");
  }
}
```
Extending Thread (continue)

A run-object to test:

```java
public class TestMyThread {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

*na-Exercise: Modify run() in MyThread.java to do something else.*
Implementing Runnable interface

1. Write a class, implementing Runnable, whose run method is the code you want to be executed by a thread.
2. Create an instance of that class. This is the run-object.
3. Create an instance of Thread, using the run-object as constructor parameter.
4. Call the start() method on the thread. This starts the thread executing run() on its run-object.
Extending Runnable interface

implement the interface and use the class in the Thread constructor.

```java
public class HelloRunnable implements Runnable{

    //implement run method here
    public void run(){
        System.out.println("Thread by implementing Runnable");
    }
}
```
Extending Runnable interface (continue)

run-object to test:

```java
public class TestHelloRunnable {
    public static void main(String[] args){
        HelloRunnable ht = new HelloRunnable();
        Thread t = new Thread(ht);
        t.start();
    }
}
```

*na-Exercise: Modify run() in HelloRunnable.java to do something else.*
Summary

Java is multithreaded

Thread vs. Processes

Advantages of using threads

Threads in Java

Extending Thread, or implementing Runnable
Exercises

These are unassessed exercises.

☐ Check out Threads API at http://java.sun.com/j2se/1.5.0/docs/api/

2) Run MyThread5Times.java. Do you see any change in the order of the printed sentences?