March 6, 2019

CPSC-35000 Operating Systems

Spring 2019

*Lab-3 Java threading basics*

In this lab, you work on two simple Java threading programs that demonstrate race conditions and simple thread synchronization.

**Race condition**

To force race condition, two threads that update a shared variable are created and started concurrently and continuously until a corrupted value ends up in the shared variable. In Java, all primitive data types are passed by value, therefore, an array of a single integer is used as a mechanism to allow both threads update the exact same memory cell. Primitive arrays are passed by reference in Java.

Code below consists of one Thread class that simply increments a shared variable in its run function. A constructor is used to pass in the shared variable (integer array: a), which will be stored in a member variable and incremented in the run function. Two thread instances are created and run concurrently in the main function. The main function keeps on creating and running the two threads until a corrupted value indicating a race condition is obtained.

RaceCondition.java

import java.util.concurrent.\*;

import java.util.Scanner;

public class RaceCondition implements Runnable {

private int[] shared\_variable;

public RaceCondition(int[] a) {

this.shared\_variable = a;

}

public void run() {

this.shared\_variable[0] = this.shared\_variable[0] + 1;

}

public static void main(String[] args) {

while (true) {

int[] x = new int[1];

x[0] = 5;

Thread t1 = new Thread(new RaceCondition(x));

Thread t2 = new Thread(new RaceCondition(x));

t1.start();

t2.start();

try {

t1.join();

t2.join();

} catch (InterruptedException ie) {}

if (x[0] != 7) {

System.out.println("boom! race condition occurred!\n value of x: " + x[0]);

break;

} else {

System.out.println("x = " + x[0]);

}

}

}

}

To run above code, copy and paste into a Java source file and name it (RaceCondition.java). Compile and run the program, wait until you get race condition, the amount of time you have to wait for race condition to occur is indeterministic, based on my experience running the code on my Windows machine, it never took more than 2 minutes for the race condition to occur. Therefore, be patient and if it takes more than 4 minutes, try to restart the program, also, it will help if you keep the CPU busy running other programs.

**Thread coordination**

The following simple program uses Java coordination primitives (wait/notify) to execution order of two statements that belong to two different threads. This is similar to the example that we discussed in class and in our textbook.

The program consists of two thread classes, ExecuteBefore and ExecuteAfterThread, the run functions have two print statements that should go in certain order, remaining code in run functions is needed to coordinate execution of both threads and force desired ordering of the print statements.

To run code, copy paste both classes into two separate files and name them according to their respective class names. Compile, run your code and observe print statement ordering. Remove synchronization code, rerun your code and observe how order of print statements is indeterministic.

ExecuteBefore.java

import java.util.concurrent.\*;

import java.util.Scanner;

public class ExecuteBefore implements Runnable {

private Object sync;

public ExecuteBefore(Object ob) {

this.sync = ob;

}

public void run() {

System.out.println("Should appear first");

synchronized(this.sync) {

this.sync.notify();

}

}

}

ExecuteAfterThread.java

import java.util.concurrent.\*;

import java.util.Scanner;

public class ExecuteAfterThread implements Runnable {

private Object sync;

public ExecuteAfterThread(Object ob) {

this.sync = ob;

}

public void run() {

try {

synchronized(this.sync) {

this.sync.wait();

System.out.println("Should appear second");

}

} catch (InterruptedException ie){}

}

public static void main(String[] args) {

Integer shared\_object = new Integer(0);

Thread threadAfter = new Thread(new ExecuteAfterThread(shared\_object));

Thread threadBefore = new Thread(new ExecuteBefore(shared\_object));

threadAfter.start();

threadBefore.start();

try {

threadAfter.join();

threadBefore.join();

} catch (InterruptedException ie) {}

}

}