SFDV 3006 Concurrent Programming
Lab 2 – Interference

MCQs

1. Which of the following is not true about interference and interleaving
a) interleaving may cause interference
b) when there is no interleaving there is no interference
c) there is no relation between interleaving and interference
d) interference happens only when there is interleaving

2. What are the minimum numbers of threads that are needed for interference?
a) 0
b) 1
c) 2
d) 3

3. Which of the following operations is atomic?
a) i = i--;
b) i = i + 2;
c) i++
d) i = 0;

4. Which of the following is true about interference?
a) when there is interference you always get correct results.
b) when there is interference you always get incorrect results.
c) it is easy to test for problems that are caused by interference
d) the solution to the problems of interference is Mutual Exclusion

5. What is the maximum numbers of threads that should be executing in
the critical section at any one time?
a) 0
b) 1
c) 2
d) depends on the number of CPUs in the computer

6. What is the solution to make a class thread safe in Java so that there is no interference for its shared object?
a) Make the class synchronized in the class declaration
b) Make the run() methods of the thread which access the object of this class syncrhonized
c) Make the methods in the objects synchronized
d) Make the methods in the class synchronized

7. What happens when a thread which calls a synchronized methods cannot acquire the object lock?
a) The thread sleeps
b) The thread goes into a busy loop
c) The thread will start to do I/O since there is nothing else to do
d) The thread is blocked and put into the entry set
e) The thread is blocked and put into the wait set

8. How can the performance of a multi threaded program be increased when its threads are accessing data of a shared object?
a) Do no make the methods of the class *synchronized*
b) Increase the scope of the *synchronized* from the method to the class
c) reduce the scope of *synchronized* from method to the block around the critical section
d) Increase the number of CPUs

Short Answer Questions

1. Assume that three threads *t1, t2* and *t3* access the shared object of the *Sequence* class (from the lecture) concurrently whose current value of *next* is *7*, what will be the value of the *next* in the shared *Sequence* object after the three threads finish executing? Can the value of *next* be predicted correctly? Why or why not?
2. Draw all the possible traces how the interleaving may happen between t1, t2 and t3. The traces must show the value of *next* in each case.
3. How many threads should be in the critical section at any one time? Explain why.
4. How does using Java *synchronized* prevent interference? Explain.
5. What are the problems with Java *synchronized*?
6. What is an entry set? Explain how and why the JVM puts a thread into the entry set.
7. Explain how decreasing the lock scope can increase performance.
8. What is mutual exclusion? Explain how mutual exclusion solves the problems of interference.

Lab Work

1. Run the *SequenceInterference* example. Understand what should be the correct value and why the correct value is not being produced.
2. Comment the following line in the *Sequence* class (from slide 5 in Lecture 2):
   ```
   Simulate.HWinterrupt();
   ```
   and then compile the run your code again. Is the output correct? Why or why not?
3. Change the *Sequence* class such that when there is no interference – so that always correct results are produced (make sure the `Simulate.HWinterrupt();` is not commented).
   [See page 3 for code the SequenceInterference or take from the web]
4. Run the *TwoThreads* class which has two inner classes *Thread1* and *Thread2*. What is the output when the *yield()* lines in the both the classes are commented. Compare the output when *yield()* line in both the classes is not commented. Explain the output for both the cases.

```
//Code for Lab Questions 1, 2 and 3

//Sequence.java
public class Sequence {
	private int next = 0;

	public int getNextInt() {
		//next = next + 1;
		int temp = next;
		Simulate.HWinterrupt();
		temp = temp + 1;
		next = temp;
		return next;
	}

	//return the current value of next
	public int getCurrentValue() {
		return next;
	}

}

//Simulate class from Magee and Kramer to simulate interference
class Simulate {
    public static void HWinterrupt() {
        if (Math.random()<0.5)
           try{Thread.sleep(200);} catch(InterruptedException e){};
            //used instead of Thread.yield() to ensure portability
    }
}

//SequenceThread.java
public class SequenceThread extends Thread {
	private Sequence sequence = null;

	public SequenceThread(Sequence _sequence) {
		sequence = _sequence;
	}

	public void run () {
		for (int i = 1; i <= 100 ; i++)
			System.out.println(this.getName() + ":"+sequence.getNextInt());
	}
}

//SequenceInterferenceTester.java
public class SequenceInterferenceTester {
	public static void main(String args[]) {
		//create an instance of the shared object
		Sequence sequence = new Sequence();

		//Create two thread which the getNextInt() on the shared object
		SequenceThread s1 = new SequenceThread(sequence);
		SequenceThread s2 = new SequenceThread(sequence);

		s1.start();
		s2.start();
	}
}
```

//Code for Lab Question 4

```
//From IBM DeveloperWorks Tutorial on Java Threads by Brian Goetz
//http://www.ibm.com/developerworks/java/tutorials/j-threads/j-threads-pdf.pdf
public class TwoThreads {
	public static class Thread1 extends Thread {
		public void run() {
			System.out.println("A");
			//yield();
			System.out.println("B");
		}
	}

	public static class Thread2 extends Thread {
		public void run() {
			System.out.println("1");
			//yield();
			System.out.println("2");
		}
	}

	public static void main(String[] args) {
		new Thread1().start();
		new Thread2().start();
	}
}
```