**CS 346 – Lab 4 – Threads**

Today, we will practice with threads in the Java environment.

Please log in to the Linux server, and create a new directory called lab04. Copy the file Driver.java from my /home/chealy/www/cs346/lab04 directory. Today we will be experimenting with three different programs. Since it may be convenient to refer to each one as "Driver.java", you may want to put each into its own directory. In this case, please create three subdirectories part1, part2 and part3 so that the class names from each program do not clash with the others.

1. (part1 directory) Driver.java consists of 4 classes. The purpose of this program is to repeatedly increment two variables, x and y. Each variable is incremented by a different thread. Because one of these threads also has additional work to do, the variables increment at different rates.

   Compile and run Driver.java. What does the output tell you about the execution of the two threads? In particular consider these issues:
   - Why would we print the same ordered pair twice (or several times) in a row?
   - Between consecutive printings, why might one value change, but not the other? Does this scenario occur often?

   ______________________________________________________________

   ______________________________________________________________

   ______________________________________________________________

   ______________________________________________________________

   Run the program a couple more times. Is the output the same every time? What do you believe is happening?

   ______________________________________________________________

   Edit the file Driver.java. Uncomment the code that contains the try/catch code. Recompile and run the program again. How is the output different? What is the effect of this new code we added to the program?

   ______________________________________________________________

   ______________________________________________________________

2. (part2 directory) Next, you will write a short Java program from scratch. It will contain two classes: one class (Driver) will contain main(), while the other class (Generate) will implement the Runnable interface.

The main() method will need to create an array of 9 Thread objects. Then, it will run all nine threads. Each thread will print a unique number 1 through 9. In other words, its unofficial “thread number”. The way to do this is to have the Generate class maintain this number as an attribute. It will have an initial-value constructor where the number 1-9 is passed in. The run() method just needs to print a message containing this number.
For example, the output of your program should look something like this:

```
Thread 1 is inside its run()
Thread 5 is inside its run()
Thread 2 is inside its run()
Thread 6 is inside its run()
Thread 8 is inside its run()
Thread 3 is inside its run()
Thread 7 is inside its run()
Thread 4 is inside its run()
Thread 9 is inside its run()
```

3. (part3 directory) Finally, let’s do some useful calculations with multiple threads. Since it is straightforward to create many threads, let’s give each one a piece of a problem. The problem we will tackle is to print all n-digit prime numbers, where n is some reasonable number like 5. Each thread will tackle one-ninth of the problem. For example, one thread will look at numbers starting with a 1, another thread will look at numbers starting with 2, … and the last thread will look at numbers starting with 9.

   Copy the Driver.java program you worked on in part2 into a new source file (in a new directory as appropriate).

   You now need to modify the code inside the run() method so that it will print all 5-digit prime numbers starting with the digit that matches the thread number. Remember that the Generate class already has an integer attribute in the range 1-9, and this will tell you what the first digit of the candidate prime numbers should be. For example, in thread #7, we should be looking at all the numbers from 70000 to 79999 to see if each is prime. For each prime you discover, print it out. Use an efficient procedure to determine primality. Here is a good way to see if a number n is prime: count the number of divisors from 2 to the square root of n. If you find no such divisors, the number is prime. In fact, if you do discover a divisor, you can break from the loop and go on to the next candidate prime.

   Compile and run your program. Because there is a lot of output, redirect the stdout to a text file. It’s very likely that your entire list of prime numbers is not in ascending order. Why would this be? ______________________________________________________

4. Let’s look at the behavior of your prime-generating program.
   a. Add code to the beginning and end of your run() method announcing that this thread number is starting/finishing. The messages should say, for example, “Thread 4 starting” and “Thread 4 finishing”. But of course you would not hard-code the number 4 in this string literal, because the thread number varies.

   b. Add a statement to the end of main() announcing that all the threads are finished. In order for this to really work, you need to make sure that all 9 threads are indeed done. You need to use the join() function on all the child threads. As a general guide, here is an example of how we can join many threads:

```
int numThreads = 10;

Thread [] worker = new Thread[numThreads];

for (int i = 0; i < numThreads; ++i)
{
   try
   {
      worker[i].join();
   }
   catch (InterruptedException e)  { }
}
```

c. Run your program, and redirect your output to a file called 5-digit.out. Run these Linux commands:

```
grep –n start 5-digit.out
grep –n finish 5-digit.out
```

The purpose of the "n" option is to give you the line number of the match in the file.

d. Based on the output of the previous commands, let's consider the behavior of your multi-threaded program. Do all the threads finish in the same order they started? Why or why not?

______________________________________________

______________________________________________

______________________________________________

Does any thread finish before some other thread has the chance to start? How can you tell?

______________________________________________

______________________________________________

______________________________________________

e. Modify your program so that it does 3-digit primes instead of 5-digit primes. A nice way to do this is to accept the digit length of primes (e.g. 3) as a command-line argument, and then pass this value to the Generate constructor along with the thread id number. The run method would use this variable number of digits instead of a hard-coded constant. Then you would run your program as "java Driver 3". How does the program behave now? (You may want to run the program a few times to be sure.)

______________________________________________

______________________________________________

______________________________________________

f. Finally, run your program so that it prints 4-digit primes (java Driver 4). Redirect your output to a new output file, e.g. 4-digit.out. Based on this output, how many times does each thread have a turn running in the CPU?

| Thread # | # turns at CPU |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

When you are finished, please send me your part3 Driver program, as well as your answers to all of the lab questions.