CSC 543 Multiprocessing & Concurrent Programming, Fall 2018

Dr. Dale E. Parson, Assignment 1, Thread-safe blocking queues in Producer -> Consumer Pipelines

This assignment is due by 11:59 PM on September 29 via make turnitin.

The goals of this assignment are to: 1) Write your first multi-threaded Java program for the course, using my single-threaded code as a starting point; 2) Use non-blocking Queue-derived classes and BlockingQueue-derived classes from java.util.concurrent; 3) Compare the performance of the above classes in a pipeline.

Perform the following steps to set up for this semester’s projects and to get my handout. Start out in your login directory on csit (a.k.a. acad).

```bash
cd $HOME
mkdir multip
```

Then log into mcgonagall via the ssh command from acad.

```bash
ssh mcgonagall
```

This is a Linux machine with 32 contexts (hardware threads) and the same architecture as acad. I am also getting accounts for you in place on harry for testing, a Sparc/Solaris machine with 64 contexts. Do your development and testing on mcgonagall, then do some additional testing on harry when available. Tests will take longer to run on harry, but timing will be less influenced by your classmates, due to the number of processors. Also, harry tends to magnify the benefits of successful approaches to multithreading more than the other machines. Harry serves as somewhat of a microscope into your programs’ architectures.

After logging into mcgonagall, do the following.

```bash
cd ./multip
unzip pipeline2018.problem.zip
cd ./pipeline2018
gmake test
```

The gmake program is just an alias for make on Linux (i.e., GNU make). You can just type make on Linux, but on Solaris it needs to be gmake. If your UNIX PATH is not set up to find gmake on harry, just type /opt/csw/bin/gmake instead of gmake. Also, you should edit two files, .bash_profile and .bashrc in your login directory (create them if they do not exist) and add this line at the top of each. You do not need to do this if gmake works OK on harry.

```bash
export PATH="$PATH:/opt/csw/bin"
```

Make sure to use plain old double-quote characters from your keyboard. Don’t paste them from this file.

Getting back to gmake test, it should work OK and create output that looks something like this:

```
$ gmake test
/bin/rm -f *.o *.class .jar core *.exe *.obj *.pyc
/bin/rm -f *.class *.out *.dif *.tmp sink.ref
```
The test driver in class pipeline2018.BigDecimalPipelineBuilder builds a pipeline of concrete, active objects derived from interface PipelineStage, alternating with connectors derived from interface Queue<E>, with <E> bound to <java.math.BigDecimal[]>.

A dataflow diagram for the pipeline constructed by BigDecimalPipelineBuilder, and a class diagram for the classes in the project, appear on the next page. The pipeline generates a series of bursts of pseudo-random java.math.BigDecimal values in the first stage, sums a given burst in the second, and writes the summations to a file in the third. The second stage also supports multiplication, but the contribution of multithreading is less helpful because the BigDecimal terms grow so big that memory consumption-based...
paging comes to dominate execution time. My solution shows similar performance enhancements on all test platforms for summation.

There are STUDENT comments in each of these files that outline your work.

```
$ grep -l STUDENT *
BigDecimalPipelineBuilder.java
PipeSinkFile.java
PipeSourceRandom.java
PipeStageMath.java
testscript
```

For each Java class (not the interface) you must precede the class declarations with one of the annotations from the textbook as found in jcip-annotations.jar (@NotThreadSafe, @ThreadSafe, and @Immutable), and for the latter two categories, you must ensure that the classes satisfy that tag.

1. Make any data field private that does not have to be more openly exposed.
2. Make any data field that does not need to change after construction final. These would be all fields for @Immutable.
3. Make any data field accessed by both the Pipe* constructor and configure() calls (which occur in the main thread) and the active object thread (which runs in the run() method) as volatile, since we need to guarantee that changes made in the constructor are visible to the object thread. We could get the same cache-consistency effect by locking the object, but we are not programming with locks yet.

4. At the top of the run() method copy any volatile field into a local variable, and use that local variable in the run() method. These volatile fields are not used outside the run() thread once it begins; copying them for use into locals eliminates unnecessary cache flushes and main memory fetches; local variables are thread-confined and not subject to inter-thread cache consistency problems.

I will be grading on all of these requirements.

For the above Pipe* files, you need to change a number of non-blocking calls to blocking calls for BlockingQueue objects. See STUDENT comments. Those blocking methods declare that they throw java.lang.InterruptedException. In places where my handout code does not catch this exception, you must catch and ignore it. There can be no InterruptedExceptions in this program because we do not use Java interrupts for communication between threads. In later assignments we will use them. For now an empty catch clause where mandated by the compiler is fine.

In BigDecimalPipelineBuilder you will find this comment:

// STUDENT write this block of code.
// You will write code to run each pipeline stage in transformers
// in its own Thread (start running from transformers.get(0) in
// order to pump up the queues),

Your solution in this block of code must start a Thread of execution in left-to-right sequence for each pipeline stage, then join() each Thread after they have all started via Thread.join() (a separate join loop), and finally sum and report all statistics reported by my single-threaded code block that precedes your code block. The Pipe* threads sum their aggregate CPU and USER times in AtomicLong counters. You must read these counters and combine their values with the values for the main thread that runs your code.

Finally, you must add tests to script testscript, whose listing appears below. I have highlighted requirements below.

```
testscript
1  #!/bin/bash
2  # testscript runs a series of tests for csc543 assignment 1 fall 2018 D. Parson
3  # HERE IS D. PARSON'S HANDOUT TEST FUNCTION. KEEP IT UNCHANGED & CALL IT.
4  function runatest() {
5      # Parameters:
6      # $1 is USETHREADS
7      # $2 is USEBLOCK
8      # $3 is the name of the Queue class; use its full package path in the call.
9      # $4 is 1 to exit on run-time error, 0 to not exit
10     CLASSPATH=..:jcip-annotations.jar time java pipeline2018.BigDecimalPipelineBuilder $1 $2
12     pipeline2018.PipeSinkFile ~parson/tmp/${STUDENT}sink.out
```
exitStatus=$?
chmod 666 ~parson/tmp/${STUDENT}sink.out
ln -s ~parson/tmp/${STUDENT}sink.out ./sink.out
if [ $exitStatus -ne 0 ]
then
echo "TEST ERROR" 1>&2
exit $exitStatus
fi
diff ./sink.out ~parson/tmp/sink.ref > sink.dif
exitStatus=$?
if [ $exitStatus -ne 0 ]
then
echo "DIFF ERROR" 1>&2
exit $exitStatus
fi
/bin/rm -rf ~parson/tmp/${STUDENT}sink.out ./sink.out
}

# 1. HERE IS MY TEST. STUDENT TESTS GO AT THE BOTTOM.
runatest false false java.util.LinkedList   # This is the first test.

# The first "false" above is USETHREADS, and the second is USEBLOCK.
# IF POSSIBLE:
# You should run all of your tests with USETHREADS=false, and then, for the
# Queue objects that are thread-safe, run again with USETHREADS=true.
# For the Queue objects that are BlockingQueue objects, make a third test run
# with USETHREADS=true and USEBLOCK=true. We are testing the multithreaded
# BlockingQueues using both their non-blocking and their blocking interfaces.
# You may not be able to run some of the Queue types in all possible
# configurations. For example, trying to write to a size-bounded
# queue may throw an exception and kill the program. Also, some of the
# queue type(s) in java.util.concurrent require constructor parameter(s), which
# are not supported by this test setup. For the ones that you cannot run,
# add a comment line below explaining why you cannot run that set of
# USETHREADS and USEBLOCK parameters.

# STUDENT: Write your tests, substituting for java.util.LinkedList a
# thread-safe Queue or BlockingQueue class path,
# and varying USETHREADS and USEBLOCK command line arguments as indicated
# in the above paragraph. Use blank line(s) to separate tests so I can see them.
runatest true false java.util.LinkedList

is7=`java -version 2>&1 | grep version | grep '1\.[789]' | wc -l`
if [ $is7 -gt 0 ]
then
    echo "Replace this line with tests using Queue classes from Java 1.7 or higher"
    fi

# NEXT ONE NOT THREAD SAFE, IT BLOWS UP IN PREP SOMETIMES.
runatest false false java.util.LinkedList
Get all of your Queue and BlockingQueue classes for testscript from java.util.concurrent. As noted above, some combinations of USETHREADS and USEBLOCK may not work for some of those classes, and in fact some of the classes may not work with our test setup. Use ALL of the ones in java.util.concurrent that complete without inherent errors (fix your bugs), and WRITE A COMMENT FOR EACH Queue/BlockingQueue test line that does not work, and explain why.

NOTE THAT YOU WILL ONLY BE ADDING TEST LINES THAT LOOK LIKE THIS

runatest false false java.util.LinkedList  # Replacing false with true and other Queue classes, but do not change # function runatest. It is already done.

Run make testcsv to create a time-sorted, comma-separated execution profile of tests on each machine after you have the program working. You must make testcsv on mcgonagall and on harry if our accounts are set up in time.

Run make turnitin on one of our Linux machines by the due date. The late penalty is 10% per day, and I will not accept solutions after I go over an assignment. Plan to attend class if possible and ask questions.