Laboratory Assignment #3

Using a Multi-Threaded Producer-Consumer Model

Andreas Landgrebe - DATE \@ "MMMM d, y" February 17, 2014

1. A paragraph that invites how multi-threaded java programs use synchronized.

In multi-threaded java programs, they use synchronized by being able to run thread at a time. In a multithreaded program, there are threads that are going to be being called at the same time. If nothing for this situation, then the program is being run, the output is throw an exception. To make sure that the threads are not run at the same time, then synchronized will have one thread run at a time. This allow fix the problem on having the exception if more than one thread is being run at the same time.

2) A detailed discussion of the execution behavior of a multi-threaded Java program.

The behavior of a multi-threaded java program given to use for this laboratory assignment will be calling several methods and functions to be able to run more than one threads synchronized. At first start off, in the main method called the ProducerConsumerModel.java, this is where you will be able to declare a hard coded number for the totalConsumeNumber and the totalDataItemsNumber variable to run different tests. The totalConsumeNumber will be calling the total number of Consumer for the model and the totalDataItemsNumber will be calling the total number of items being called. In order to have it so these two variable are not hard coded, the laboratory assignment has assigned us to use the jcommander tool to be able to parse the command line. Before this was done, after the program has been compiled, then one be able to run the program. Once the program was run, you can see the producer and consumer changed the number between 0 and 3, and then get and put is increasing by 1 for each line going down the output from the terminal. In the source code, in the Consumer.java file, it is the object for the CubbyHole is being instantiated where cubbyHole will be returning a number that is being called in. In the Producer.java file, the source code states that a number is going called to go to the producer where the CubbyHole object is being instantiated. After this is the case, after all the numbers being called from the TotalConsumeNumber and the TotalDataItemsNumber, the program will then print out that the producer is done and then the next line will print out Terminating Model. After this, I have personally written some extra code to be able to display the time spend to run this program. This is a replacement as to write as a command line /usr/bin/time. I feel this is a more efficient way to compare results from the different times this program is being run.

In order for commander to work properly, I needed to first download the source code the jcommander. After this was complete, I needed to created another java file which I called parameters. In the parameters file, this is where I imported the jcommander tool and I called in the parameters that I will be using for the assignment. In the main method of the ProducerConsumerModel, I was able to assign the variables of the totalConsumeNumber and the totalDataItemsNumbers to the parameters that I had declared before. Here is the source code:

Source Code for parameters.java

import com.beust.jcommander.Parameter;

public class parameters {

@Parameter (names = {"-totalDataItemsNumber","--tdin"}, required = true, description = "Toal number of data items going to be called")

Integer totalDataItemsNumber;

//goes through the total number of data items

@Parameter (names = {"-totanConsumeNumber","--tcn"}, required = true, description = "total number of consumers")

Integer totalConsumerNumber;

//goes through the total number of consumers

@Parameter(names = "-debug", description = "Debug mode")

boolean debug = false;

}

Source Code for ProducerConsumerModel.java:

import java.io.\*;

import java.io.File;

import com.beust.jcommander.JCommander;

public class ProducerConsumerModel

{

public static void main(String[] args)

{

parameters param = new parameters();

//call the parameters file

new JCommander(param, args);

//call the JCommander the parse the command line

// the debugging flag which will determine whether

// or not we produce output when running experiment

boolean debugOutput = param.debug;//true;

// the maximum number of consumers inside our model

int totalConsumerNumber = param.totalConsumerNumber; //set the totalConsumeNumber equal to the parameter being called to use jcommander

// the number of data items that we should produce

// and consume in this execution of the model

int totalDataItemsNumber = param.totalDataItemsNumber;//set the totalDataItemsNumber equal to the parameter being called to use jcommander

// create a CubbyHole that both the Producer and

// and the Consumer will use to store data items

CubbyHole cubbyHole = new CubbyHole(debugOutput);

// create the only producer as a single thread and

// then get it started on producing numbers

Producer producer =

new Producer(cubbyHole, 1, totalDataItemsNumber);

producer.start();

// this is the single Consumer variable that we will

// use to start each of the Consumer threads

Consumer consumerSource;

// create all of the different consumer threads

for(int i = 0; i < totalConsumerNumber; i++)

{

consumerSource = new Consumer(cubbyHole, i);

consumerSource.start();

}

// wait for the producer to finish and then terminate

// the process (note: this would not always work if the

// Consumer threads had "tasks" that took a long time

// to complete)

try

{

producer.join();

System.exit(1);

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

}

Source Code for Producer.java

public class Producer extends Thread

{

private CubbyHole cubbyhole;

private int number;

private int maxTimes;

long startTime = System.currentTimeMillis();//show the start time

public Producer(CubbyHole c, int number, int maxTimes)

{

cubbyhole = c;

this.number = number;

this.maxTimes = maxTimes;

}

public void run()

{

for (int i = 0; i < maxTimes; i++)

{

cubbyhole.put(number, i);

try

{

sleep((int)(Math.random() \* 100));

}

catch (InterruptedException e)

{ e.printStackTrace(); }

}

long timeSpend = System.currentTimeMillis() - startTime;

System.out.println("Producer done.");

System.out.println("Terminating model.");

System.out.println("Time Spend: " + (float)(timeSpend/1000.00) + " seconds"); //display the time spend in seconds

3) A comprehensive analysis of the output of each defective multi-threaded Java program.

Here is the output when both the synchronized methods for get and put in cubbyhole is commented out:

The output shown below is what happens when multiple threads are trying to be run at the same time. As long as the notifiyAll() methods and the while loops for both the get and the put methods are not commented out, the output will be shown as an exception where multiple threads are trying to be run at the same time.

Exception in thread "Thread-0" Exception in thread "Thread-1" Exception in thread "Thread-2" java.lang.IllegalMonitorStateException

at java.lang.Object.wait(Native Method)

at java.lang.Object.wait(Object.java:503)

at CubbyHole.get(CubbyHole.java:22)

at Consumer.run(Consumer.java:22)

Exception in thread "Thread-4" java.lang.IllegalMonitorStateException

at java.lang.Object.notifyAll(Native Method)

at CubbyHole.put(CubbyHole.java:70)

at Producer.run(Producer.java:25)

java.lang.IllegalMonitorStateException

at java.lang.Object.notifyAll(Native Method)

at CubbyHole.get(CubbyHole.java:39)

at Consumer.run(Consumer.java:22)

Exception in thread "Thread-3" %

Here is the output when the synchronized method for get is commented out and the put synchronized method is not commented out:

The output shown below is what happens when multiple threads are trying to be run at the same time. As long as the notifiyAll() methods and the while loops for both the get and the put methods are not commented out, the output will be shown as an exception where multiple threads are trying to be run at the same time.

Exception in thread "Thread-1" Exception in thread "Thread-4" Exception in thread "Thread-3" Exception in thread "Thread-2" java.lang.IllegalMonitorStateException

at java.lang.Object.notifyAll(Native Method)

at CubbyHole.get(CubbyHole.java:39)

at Consumer.run(Consumer.java:22)

java.lang.IllegalMonitorStateException

at java.lang.Object.wait(Native Method)

at java.lang.Object.wait(Object.java:503)

at CubbyHole.get(CubbyHole.java:22)

at Consumer.run(Consumer.java:22)

java.lang.IllegalMonitorStateException

at java.lang.Object.wait(Native Method)

at java.lang.Object.wait(Object.java:503)

at CubbyHole.get(CubbyHole.java:22)

at Consumer.run(Consumer.java:22)

java.lang.IllegalMonitorStateException

at java.lang.Object.wait(Native Method)

at java.lang.Object.wait(Object.java:503)

at CubbyHole.get(CubbyHole.java:22)

at Consumer.run(Consumer.java:22)

Here is the output when the synchronized method for put is commented out and the get synchronized method is not commented out:

The output shown below is what happens when multiple threads are trying to be run at the same time. As long as the notifiyAll() methods and the while loops for both the get and the put methods are not commented out, the output will be shown as an exception where multiple threads are trying to be run at the same time.

Exception in thread "Thread-0" java.lang.IllegalMonitorStateException

at java.lang.Object.notifyAll(Native Method)

at CubbyHole.put(CubbyHole.java:70)

at Producer.run(Producer.java:25)

Defect when the both the while loops and the notfiyAll() method for both the get and put method is commented out when not using synchronized:

In this defect, when all of the information was commented out, the defect was that it was not able to print out the producer or show the get methods. This is the case because when the while loop, notifyAll() and synchronized for both get and put are commented out, then all the threads are trying to use at the same time and since the totalConsumeNumber is the number that is being parsed as a command line tool, the consumer will have first priority which is the reason why one will see the output for the consumer and not the producer.

4) The report from an experimental study that characterizes the performance of the model.

In this laboratory assignment, we were given the task to study the source for the java files given to us through the version control repository. In this tasks, we needed to run different experiments for different possible amount of totalDataItemsNumber. One of the experiments that I had run was when I decided to use the totalConsumeNumber as 8 and and the totalDataItemsNumber as 10. Here is the results.

## This is when I experimented with the totalConsumeNumber to be 8 and the TotalDataItemsNumber was 10 on 5 different times. As you can see in the graph, the second time I ran this experiment, it took the shortest amount of time and when I ran the third experiment, it took the longest amount of time compared all of the other times I ran this experiment.

I had decided to run any experimented where the totalConsumeNumber was 4 and the TotalDataItemsNumber was 100. Here are the results from that experiment:

This was when I experimented where the totalConsumeNumber was 4 and the totalDataItemsNumber is 100. As you can in the graph, the second experiment took the longest amount of time and the fourth experiment took the shortest amount of time.

I had decided to run another experimented where I tested totalConsumeNumber to be 4 and the totalDataItemsNumber to be 1,000. Here are the results:

These were the results from when I was running the totalConsumeNumber to be 4 and the totalDataItemsNumber to be 1000. As you can see in the graph, the first experiment took the longest amount of time and the second experiments took the shortest amount of time.

The last experimented I ran was when the totalConsumeNumber was 4 and the totalDataItemsNumber was 10,000.

These the results from when the TotalConsumeNumber was 4 and the totalDataItemsNumber was 10,000. As you can see in the chart, the second experiment took the longest amount of time and the third experiments took the shortest amount of time.