Homework 4

PRODUCE CONSUMER PROBLEM: MANAGING SYNCHRONIZATION

This assignment combines aspects of synchronization and deadlocks. The problem you will be solving is the bounded-buffer producer-consumer problem using threads. You are required to use Java for this assignment.

DUE DATE: October 10th, 2018 @ 5:00 pm

1 Description of Task

For this assignment we will be solving the producer-consumer problem with a bounded buffer. You are required to implement this assignment in Java. There are **three components** in this assignment.

(1) The Bounded Buffer: This buffer can hold a fixed number of items. The number of elements in this buffer is limited to 1000. This buffer needs to be a first-in first-out (FIFO) buffer. You should implement this as a Circular Buffer that satisfies the FIFO property and is restricted to hold at most 1000 items at a time.

(2) Producer: The producer is responsible for producing data items to be added to the buffer. If the buffer is full, the producer must wait for the consumer to consume at least one item before it can add a new item to the buffer. The producer is required to produce a total of 1,000,000 items. The item that the producer adds to the buffer is a random Double that is generated using java.util.Random.

```
Random random = new Random();
Double bufferElement = random.nextDouble() * 100.0;
```

(3) Consumer:
The consumer is responsible for consuming elements from the buffer. If the buffer is empty the consumer must wait for the producer to add an item to the buffer. The consumer is required to consume all elements (1,000,000) generated by the producer.

There should be exactly one instance of the buffer, producer, and consumer. The producer and consumer must reference the same buffer. You can only use **`wait()`** and **`notify()`** as the primitives to synchronize access to the buffer.

Correctness Verification:

To verify the correctness of your program both the producer and consumer are required to maintain running totals of the values of the items added-to/removed-from the buffer. This would be maintained in a variable bufferValueCounter.

Since the buffer is a FIFO buffer, the *value reported for the two items below should be the same*:
(a) The first N elements generated by the producer
(b) The first N elements consumed by the consumer

2 Requirements of Task

1) Implement the FIFO Circular Buffer and ensure that the buffer can hold a maximum of 1000 items at a time.
2) Consumer should wait if there are no items in the buffer
3) Producer should wait if the buffer is full
4) Ensure that the producer can produce 1,000,000 items and the consumer can consume 1,000,000, items with a bounder buffer that can hold at most 1000 items at a time.
   a. *You should not run out of memory*. Remember, your producer should not produce if the buffer is full. You should not set the size of the Java VM in your makefile or expect that the JVM will be set to a certain value.
   b. Your solution must satisfy the correctness constraint i.e. you *consume each item exactly once, in the order that it was produced*, and *demonstrate this* by printing out the value of your counter (at both the producer and consumer) every time you have produced or consumed 100,000 elements. See the example output below; note how the counter values match at the producer and consumer for the same N.
   c. *There should be no deadlock*. Your program will be executed 5 times, and it should run to completion every time without a deadlock.

Restrictions and Deductions:

[R1]. There is a 10-point deduction if you use an unbounded buffer for this assignment or if your buffer holds more than 1000 items at a time.

[R2]. There is a 9-point deduction if you use Thread.sleep() to synchronize access to the buffer. You can only use **`wait()`** and **`notify()`** as the primitives to synchronize access to the buffer.

[R3]. You should not use any classes from the java.util.concurrent package to solve this problem. There is a 10-point deduction if you do so.

[R4]. You should not use ANY external library to solve this problem. There is a 10-point deduction if you do so.

[R5]. You should not use a Boolean flag or any variable that toggles in values so that your producer and consumer take turns adding-to or consuming-from the buffer. There is a 8-point deduction if you do this. The solution must be based entirely on the use of **`wait()`** and **`notify()`**.

3 Example Outputs:

Print out of what the outputs of the executing programs look like

```
> java ProducerConsumer
Consumer: Consumed 100,000 items, Cumulative value of consumed items=4996680.774
Producer: Generated 100,000 items, Cumulative value of generated items=4996680.774
Producer: Generated 200,000 items, Cumulative value of generated items=9991051.138
Consumer: Consumed 200,000 items, Cumulative value of consumed items=9991051.138
Producer: Generated 300,000 items, Cumulative value of generated items=14999465.523
Consumer: Consumed 300,000 items, Cumulative value of consumed items=14999465.523
Producer: Generated 400,000 items, Cumulative value of generated items=19989453.197
Consumer: Consumed 400,000 items, Cumulative value of consumed items=19989453.197
Producer: Generated 500,000 items, Cumulative value of generated items=24985125.256
Consumer: Consumed 500,000 items, Cumulative value of consumed items=24985125.256
Consumer: Consumed 600,000 items, Cumulative value of consumed items=29993909.745
Producer: Generated 600,000 items, Cumulative value of generated items=29993909.745
Producer: Generated 700,000 items, Cumulative value of generated items=34999014.908
Consumer: Consumed 700,000 items, Cumulative value of consumed items=34999014.908
Consumer: Consumed 800,000 items, Cumulative value of consumed items=40007320.547
Producer: Generated 800,000 items, Cumulative value of generated items=40007320.547
Producer: Generated 900,000 items, Cumulative value of generated items=44987097.526
Consumer: Consumed 900,000 items, Cumulative value of consumed items=44987097.526
Producer: Generated 1,000,000 items, Cumulative value of generated items=49988555.385
Consumer: Consumed 1,000,000 items, Cumulative value of consumed items=49988555.385
Producer: Finished generating 1,000,000 items
Consumer: Finished consuming 1,000,000 items
Exiting!
```

Nota Bene:

In the outputs above, you may notice that occasionally the Consumer prints out that it has consumed a certain number of items (say **X**) before the producer prints out that it has produced an **X** number of items. This occurs because of thread context switching; where the system context-switches to the consumer thread, before the producer thread has had a chance to print to screen.

4 What to Submit

Assignments should be submitted through Canvas. E-mailing the codes to the Professor, GTA, or the class accounts will result in an automatic 1 point deduction.

Use the CS370 *Canvas* to submit a single .zip file that contains:

- All Java files related to the assignment (please document your code),
- a Makefile that performs both a *make clean* as well as a *make all,*
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program.

Filename Convention: You should call the Class that drives your program ProducerConsumer. You can name your other supporting java files anything you want. The archive file should be named as <LastName>-<FirstName>-HW4.zip . E.g. if you are Cameron Doe and submitting for HW4, then the zip file should be named Doe-Cameron-HW4.zip.

5 Grading

This assignment would contribute a maximum of 10 points towards your final grade. The grading will also be done on a 10-point scale. The points are broken up as follows:

1 point Task 1

A total of **1 point** for Tasks 2 and 3

8 points for Task 4
1 point for 4.(a)
3 points for 4.(b) and
4 points for 4.(c))

You are required to work alone on this assignment.

6 Late Policy

Click here for the class policy on submitting late assignments.