Programming Assignment 5

PRODUCER CONSUMER PROBLEM: SYNCHRONIZATION & DEADLOCKS v10/27/2016 5PM

This assignment requires synchronization when producers and consumers access a buffer. The problem you will be solving is the bounded-buffer producer-consumer problem using threads. You must use Java for this assignment.

DUE DATE: Thursday, Nov 10, 2016 @ 5:00 pm (late Nov 14, 5:00 PM)

Description of Assignment

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

(1) The Bounded Buffer: This buffer can hold a fixed number of items. This buffer needs to be a first-in first-out (FIFO) buffer. You should implement this as a Circular Buffer that satisfies the FIFO. There should be exactly one instance of the buffer. The producers and consumers must reference the same buffer.

(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 given number of items. The item that the producer adds to the buffer is a random integer (with value between 0 to 99).

When a producer successfully inserts an item in the buffer it should print the location of insertion and time when insertion occurs (with one hundredth millisecond resolution), using this format:

```
Producer 1: Placed 58 at Location 2 at Time: 2016-10-29 15:46:50.20912
```

If the random number generated is 58.

If the producer is unsuccessful it should report the failure and time like this

```
Producer 1: Unable to insert, buffer full, at Time: 2016-10-29 15:47:50.21012
```

(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 the elements generated by the producer. There may be one or two consumer threads running at the same time.

When a consumer successfully removes an item from the buffer it should print the location of removal and time when removal occurs using this format:

```
Consumer 1: Removed 58 from location: 2 at Time: 2016-10-29 15:49:50.20924
```

If the consumer is unsuccessful it should report the failure and time like this

```
Consumer 2: Unable to consume, buffer empty, at Time: 2016-10-29 15:57:50.31012
```

Both producer and consumer should wait for a random period (as specified below using the Maximum wait time) before generating or consuming the next item, to simulate the uncertainty in timing. You can only use wait() and notify() as the primitives to synchronize access to the buffer.

The producer and the consumer classes should take these arguments: 1. number of items in the buffer (the size of the buffer) , 2. Maximum wait time (used for generating random wait time would be smaller) in nanoseconds, 3. The number of items to be produced (only for the producers).

4. Your main program should accept these command line arguments 1. number of items in the buffer (buffer size) BI, 2. number of items to be produced/consumed 3. Maximum wait time.

There may be one or two producer threads, and one or two consumer threads running at a time. Default is 2 producer and 2 consumer threads. The number of threads should be specified like this:

```
public static final int   numProducers = 2    //Choose 1 or 2
public static final int   numConsumers = 2    //Choose 1 or 2
```

The producer threads terminate when the specified number of items have been produced. The consumer threads terminate when the buffer has been empty for 10 times the Maximum wait time. The program terminates when all producer/consumer threads have terminated.

Correctness Verification:

The items produced should match the items consumed.

The circular buffer should work as intended. Only one thread should be able to access the buffer at a time.

An item can be consumed only after it has been produced. However if the consumption is very quick, within the smallest time resolution, production/consumption may appear to happen at the same time, and the reports may get printed in wrong order, if the consumer printing occurs first.

Requirements of Task

1) Implement the FIFO Circular Buffer and ensure that the buffer can hold the right number items at a time, and the access to it is synchronized.

2) A consumer should wait if there are no items in the buffer (up to a time limit as specified above)

3) A producer should wait if the buffer is full.

4) a. That the printing requirements are met. 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 status. c. There should be no deadlock. Your program will be executed multiple times, and it should run to completion every time without a deadlock. d. Your program should work for any combination of the number of producers or consumers in the third part.

Restriction and Deductions:

[R1]. There is a 10-point deduction if you use an unbounded buffer for this assignment.

[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. Thread.sleep() may be used for inserting random delays.

[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 an 8- point deduction if you do this. The solution must be based entirely on the use of wait() and notify().

What to Submit

Use Canvas for CS370 to submit a single .tar file that contains:

- All Java files related to the assignment (please document your code),
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program.
- A text file that captures your output for a specific run.

Ensure that it runs as expected on the **CS department machines**.

Filename Convention: Use the filenames specified. The archive file should be named as PA3.tar

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:

20 points for Task 1

A total of 20 points for Tasks 2 and 3

60 points for Task 4: 10 point for 4.(a), 20 points for 4.(b) , 20 points for 4.(c) and 10 point for 4.(d).

You are required to work alone on this assignment.

Late Policy: 10% off for the first 24 hours, 20% for the next 24 hours. No credit after that.

Resources

You may find these documents useful: The SimpleThreads Example, Synchronized Methods:

Notes:

- Your main program may need java.util.Scanner, producer/consumer may need java.sql.Date,, java.text.DateFormat, java.text.SimpleDateFormat, java.util.Calendar, java.util.Random.
- You may need some time/calendar related code like

```
        this.start = System.nanoTime();
        DateFormat dateFormat = new
                                  SimpleDateFormat("yyyy/MM/dd HH:mm:ss.SSSSS");
        Calendar cal = Calendar.getInstance();
```