Programming Assignment HW6

PRODUCER CONSUMER PROBLEM: SYNCHRONIZATION & DEADLOCKS v11/10/2017 6PM

Objective: This assignment requires synchronization among multiple Java thread. The problem you will be solving is implementing a bounded-buffer with one producer and multiple consumers, and verifying the implementation.

DUE DATE: Fri, Nov 17, 2017 @ 5:00 pm (late Nov 19, 5:00 PM)

Description of Assignment

Write a Java program implementing the producer-consumer problem with a bounded circular buffer. You must implement this with a multi-threaded Java program. In addition to a class named HW6.java, you will have

(1) CirBuffer.java: This buffer can hold a fixed number of items. This buffer is a first-in first-out (FIFO) Circular buffer. There should be exactly one instance of the buffer. The producer and consumers must reference the same buffer.

(2) Prod.java: The producer produces data items to be added to the buffer. If the buffer is full, the producer must wait for a consumer to consume at least one item. 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: Inserted integer 27 at Location 2 at instant: 2017-11-02 15:46:50.20912

If the random number generated is 28.

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

Producer: Unable to insert, buffer full, at instant: 2017-11-02 15:47:50.21012

The producer keeps a sum of all integers generated. At the end it prints:

Producer: Sum of numbers generated: 735

(3) Cons.java: A 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 27 from location: 2 at Time: 2017-11-02 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.

Each consumer keeps a sum of all the numbers consumed. They are printed out at the end in this format:

Consumer 1: Sum of numbers consumed: 248

(4) 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 producer).

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

There is one producer thread, and one or two consumer threads running at a time.

The producer threads terminate when the specified number of items have been produced. The consumer threads terminate when the buffer has been empty for 15 times the Maximum wait time. The program terminates when all 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), and should work as specified.

3) The producer should wait if the buffer is full, and should work as specified.

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 consumers and maximum wait time.
Restriction and Deductions:

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

[R2]. There is a 15-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. Java wait(long) 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 10- 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 HW6.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, along with the command line invocation.

Ensure that it runs as expected on the CS department machines. You must ensure that all the .java files can be placed in a single directory where they can be compiled and run using command line.

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:
Example output: See here.

Notes:

- Your main program may need java.util.Scanner, producer/consumer may need java.sql.Date,, java.text.SimpleDateFormat, java.text.SimpleDateFormat, java.util.Calendar, java.util.Random.
- You may need some time/calendar related code like
  ```java
  this.start = System.nanoTime();
  DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss.SSSSSS");
  Calendar cal = Calendar.getInstance();
  ```
- You can use tar –cvf <filename>.tar <list of files> to tar your submission
- Include only the .java files not the src folder or any other folder structures
- You can implement the program using as many classes you need but name the driver class as HW6.java
- Make sure you remove “package org.java…….” from the java files which is the very first line if you are coding using IDE’s like eclipse.
- Program should run on CS Department machines. In order to verify it runs properly, you can run the following command
  ```bash
  javac *.java  -> compiles all the java files
  java HW6 <argument list>  -> runs the program with argument provided.
  ```