Programming Assignment 5: Producer - Consumer Problem: Synchronization & Deadlocks v 4/2/2019 1PM

Deadline April 11th, 2019, 8:00 PM, Late Deadline April 12th, 2019, 8:00 PM

Purpose: This assignment requires synchronization when producers and consumers access a bounded buffer. You will be solving the problem using Java threads and Java’s synchronization capabilities.

Description of Assignment:

For this Java assignment you will be solving the producer-consumer problem with a bounded buffer.

1. The Bounded Buffer (Buffer.java): This buffer is a first-in first-out (FIFO) circular buffer. There should be exactly one instance of the buffer. The producers and consumers must reference the same buffer.

2. Producers (Producer.java): The producers are responsible for producing data items to be added to the buffer. If the buffer is full, the producers must wait for the consumer to consume at least one item before it can add a new item to the buffer. The producers are required to produce a given number of items. The item that the producers add to the buffer is a random character (with values between A to Z, in upper case). There can will be one or more producer threads running at the same time.

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

   **Producer 1 inserted 0 at index 0 at time 2019-03-24 14:34:57.592084**

   It is recommended that the keyword Producer and the Producer ID in the statement should be bold to allow grading to be done efficiently. (Selecting the font may not be possible for some terminals that are character based.)

3. Consumers (Consumer.java): The consumers are responsible for consuming elements, generated by the producers, from the buffer. If the buffer is empty, the consumers must wait for the producers to add an item to the buffer. There may be one or more 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 with nanosecond resolution, using this format:

   **Consumer 7 consumed 0 from index 0 at time 2019-03-24 14:34:57.650218**
It is recommended that the keyword Consumer and the Consumer ID in the statement should be underlined.

You will use wait() and notify() as the primitives to synchronize access to the buffer.

The producer and the consumer classes should take these arguments:

I. Copy of the instance of the buffer they will access in common,
II. Number of elements that thread of producer or consumer should generate or consume. This is the total number of elements to be produced/consumed divided by the total number of producers and consumers respectively,
III. The ID which is the number of the producer or the consumer thread, i.e. if it is the first producer thread to be created, 1 should be passed, 2 for second producer thread and so on.

4. Main / Calling program (ProducerConsumer.java): Your main program should accept the following command line arguments:

   I. Number of elements in buffer/buffer size,
   II. Number of items to be produced and consumed,
   III. Number of producers,
   IV. Number of consumers
   V. Seed (This will be referred to as seedProvided later on in the document)

The producer threads terminate when the specified number of items have been produced. They should print a message like this:

**Producers**: Finished generating 8 items (Note the bold Producers keyword)

The consumer threads terminate when the specified number of items have been consumed. They should print a message like this:

**Consumers**: Finished consuming 8 items (Note the underlined Consumers keyword)

**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. To avoid this use System.out.flush()

**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) The number of items to be generated/consumed should be equally distributed among the producer and the consumer threads. Note the number of threads of producers may not match with the number of consumer threads. But the number of elements can be perfectly divided among the producers and among the consumers with no fractions involved. Also, a seed is to be passed to each producer, with an offset. Producer 1 will take the seedProvided as the seed, whereas Producer 2 will take seedProvided+10 as the seed, Producer 3 takes seedProvided+20 as the seed and so on.

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

4) A consumer should wait if the buffer is empty.

5) That the printing requirements are met. It is recommended that the bold and the underlined keywords and IDs should be used as shown.

6) 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 items produced and consumed, along with the location and the timestamp with nanosecond resolution. The code to get the timestamp with nanosecond resolution is provided to you, in ProducerConsumer.java

7) There should be no deadlock. Your program will be executed multiple times, and it should run to completion every time without a deadlock.

8) Your program should work for any combination of the number of producers and consumers.

**Questions:**

Answer the following questions in the README file. They are worth 5 points

1. We could have done it just as easily in the C programming language. 1 point
   a. True b. False

2. What two functions defined in Java are used for synchronization between producers and consumers in your program? __________ and __________ 2 points

3. In which function do you override the body to define the new body of a thread in java? __________ 1 point
4. Which function is used to wait for a thread to finish and come back to calling program i.e. for a thread to die? __________ 1 point

Restriction and Deductions:

[R1] There is a 100-point deduction if you use an unbounded buffer for this assignment.

[R2] There is a 100-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] There is a 100-point deduction for using any classes other than the following:

1. java.util.Random
2. java.lang.Exception
3. java.time.Instant
4. java.time.Clock
5. java.time.Duration
6. java.util.Formatter

[R4] There is a 100-point deduction for using any external library.

[R5] There is an 80-point deduction for using 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. The solution must be based entirely on the use of wait() and notify().

[R5] The submitted Java code will be compiled using a makefile supplied by you and executed using command line on the department machines. There will be a substantial penalty if the TAs have to do something to make your program work, which can be up to 100%.

Sample Output:

Sample 1:

System-Name:Folder/Location$ java ProducerConsumer 3 8 2 2 4

Producer   1 inserted N at index 0 at time 2019-03-24 14:30:40.535512
Producer   1 inserted D at index 1 at time 2019-03-24 14:30:40.604450
Producer   1 inserted E at index 2 at time 2019-03-24 14:30:40.604664
Consumer   2 consumed N from index 0 at time 2019-03-24 14:30:40.605040
Consumer   2 consumed D from index 1 at time 2019-03-24 14:30:40.633689
Consumer   2 consumed E from index 2 at time 2019-03-24 14:30:40.633924
Producer   2 inserted Q at index 0 at time 2019-03-24 14:30:40.634168
Producer   2 inserted K at index 1 at time 2019-03-24 14:30:40.634382
Producer   2 inserted S at index 2 at time 2019-03-24 14:30:40.634579
Consumer   2 consumed Q from index 0 at time 2019-03-24 14:30:40.634785
Consumer   1 consumed K from index 1 at time 2019-03-24 14:30:40.635426
Consumer   1 consumed S from index 2 at time 2019-03-24 14:30:40.635662
Consumer 1 consumed C from index 0 at time 2019-03-24 14:30:40.635967
Producer 1 inserted J at index 1 at time 2019-03-24 14:30:40.636236
Consumer 1 consumed J from index 1 at time 2019-03-24 14:30:40.636465

Producers: Finished generating 8 items
Consumers: Finished consuming 8 items

Sample 2:

System-Name: Folder/Location$ java ProducerConsumer 5 14 2 7 10

Producer 1 inserted O at index 0 at time 2019-03-24 14:34:57.592084
Producer 1 inserted G at index 1 at time 2019-03-24 14:34:57.649273
Producer 1 inserted T at index 2 at time 2019-03-24 14:34:57.649476
Producer 1 inserted Q at index 3 at time 2019-03-24 14:34:57.649656
Producer 1 inserted W at index 4 at time 2019-03-24 14:34:57.649858
Consumer 7 consumed O from index 0 at time 2019-03-24 14:34:57.650218
Consumer 7 consumed G from index 1 at time 2019-03-24 14:34:57.684396
Consumer 6 consumed T from index 2 at time 2019-03-24 14:34:57.684830
Consumer 6 consumed Q from index 3 at time 2019-03-24 14:34:57.685043
Consumer 5 consumed W from index 4 at time 2019-03-24 14:34:57.685343
Producer 2 inserted E at index 0 at time 2019-03-24 14:34:57.685614
Consumer 5 consumed E from index 0 at time 2019-03-24 14:34:57.685866
Producer 2 inserted M at index 1 at time 2019-03-24 14:34:57.686220
Producer 2 inserted C at index 2 at time 2019-03-24 14:34:57.686443
Producer 1 inserted H at index 3 at time 2019-03-24 14:34:57.686646
Producer 1 inserted X at index 4 at time 2019-03-24 14:34:57.686866
Consumer 1 consumed M from index 1 at time 2019-03-24 14:34:57.687106
Consumer 1 consumed C from index 2 at time 2019-03-24 14:34:57.687374
Consumer 4 consumed H from index 3 at time 2019-03-24 14:34:57.687700
Consumer 4 consumed X from index 4 at time 2019-03-24 14:34:57.687927
Producer 2 inserted M at index 0 at time 2019-03-24 14:34:57.688248
Producer 2 inserted G at index 1 at time 2019-03-24 14:34:57.688443
Producer 2 inserted V at index 2 at time 2019-03-24 14:34:57.688613
Producer 2 inserted J at index 3 at time 2019-03-24 14:34:57.688800
Consumer 2 consumed M from index 0 at time 2019-03-24 14:34:57.689056
Consumer 2 consumed G from index 1 at time 2019-03-24 14:34:57.689258
Consumer 3 consumed V from index 2 at time 2019-03-24 14:34:57.689510
Consumer 3 consumed J from index 3 at time 2019-03-24 14:34:57.689689

Producers: Finished generating 14 items
Consumers: Finished consuming 14 items
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 makefile to compile the programs and also to perform a make clean.

Ensure that your code and makefile work as expected on the CS department machines.

Filename Convention: File to be submitted should be named using the following convention: 
<FirstName>-<LastName>-HW5.tar

Notes:

1. The output should have Producer keyword and Producer IDs in bold and the Consumer keyword and Consumer IDs underlined. You can use \033[0;1m to start bold formatting, \033[0;4m to start underline formatting and \033[0;0m to stop any formatting in your print statement.
2. Look at the spacing for the word ‘at’ in Producer and the word ‘from’ in Consumer. Maintain the same spacing as only then the time stamps line up correctly and can be viewed easily.
3. Use %3d for formatter to display the ID aligned next to the Producer and Consumer. This is because we will have test case going up to 200 producers or consumers, which means the ID can be 3 digits.
4. The number of producers and the number of consumers may not be the same. However, the number of elements to be produced or consumed will always be a multiple of the number of producers and number consumers.
5. Do not define a package inside of your programs which includes all your programs, as this will raise an issue when the programs are run on terminals.
6. In order to get timestamp to the level of nanoseconds, you need to use Java 9 or above. The lab systems have Java 11, but it is set to set Java 8 as default. To make Java 11 your default, please use the following two commands.

   $ export PATH=/usr/lib/jvm/jre-11-openjdk/bin/:$PATH

   $ export LD_LIBRARY_PATH=/usr/lib/jvm/jre-11-openjdk/lib/:$LD_LIBRARY_PATH
Grading:

This assignment would contribute a maximum of 100 points towards your final grade. The grading will also be done on a 100-point scale. The points are broken up as follows:

<table>
<thead>
<tr>
<th>Points</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 points</td>
<td>10 points for each of the 8 requirements</td>
</tr>
<tr>
<td>10 points</td>
<td>For a providing a working makefile</td>
</tr>
<tr>
<td>5 points</td>
<td>Answering the questions correctly</td>
</tr>
<tr>
<td>5 points</td>
<td>For writing appropriate comments that gives a high-level overview of your code</td>
</tr>
</tbody>
</table>

Revision notes: Any revision notes will be found here.
3/30/19: Some skeleton files provided.
4/2/19: Requirement of Tasks, requirement 5 now makes bold and underlined optional.