HW5: Programming Assignment  v3.13.2021.5:00PM

PRODUCER - CONSUMER PROBLEM: SYNCHRONIZATION

This assignment requires synchronization when multiple producers and consumers access a buffer. The problem you will be solving is the bounded-buffer producer-consumer problem using threads in Java.

Due Date: Thursday, March 25, 2021, 11:00PM
Extended Due Date with 20% penalty: Friday, March 26, 2021, 11:00PM

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. It assumes that n items are to be produced and consumed.

1. The Bounded Buffer (Buffer.java): 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 producer and the 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, a producer must wait for a consumer to consume at least one item before it can add a new item to the buffer. If there are p producers, each producer is required to produce \(\frac{n}{p}\) number of items. The item that the producer adds to the buffer is a random number (with values between 0 to 99, both inclusive).

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

```
Producer   1 inserted  62 at index  0 at time 2021-03-16 15:26:50.924314
```

Checksum is used to keep the sum of all the values inserted by the producer i.e., if three elements (3, 5, and 9) are inserted by the producer, then the checksum will be 3+5+9=17. When a producer successfully inserts an item in the buffer, only then it should also add it to the checksum. This class should have a function named `getCheckSum()`, which returns the checksum of the items it has produced.

3. Consumers (Consumer.java): The consumers are responsible for consuming elements, generated by the producer, from the buffer. If the buffer is empty, the consumers must wait for the producer 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 microsecond resolution, using this format:

```
Consumer   3 consumed  62 from index  0 at time 2021-03-16 15:26:50.946821
```
When a consumer successfully consumes an item from the buffer it should also add it to the checksum. This class should have a function named `getCheckSum()`, which returns the checksum of the items it has consumed.

Mind that the statement of Consumer consuming an item should be underlined, so that it is easy to distinguish a statement from Producer and Consumer.

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

The producer class object should take these arguments:

i. Copy of the instance of the buffer it will access in common with other Producers and other Consumers,
ii. Number of elements that producer should generate,
iii. The ID, which is the number of the producer thread (i.e.: the first producer thread should be 1, the second should be 2, etc.)
iv. The seed which is used by the random number generator to generate the random numbers to be inserted.

The consumer class objects should take these arguments:

i. Copy of the instance of the buffer it will access in common with the Producers and other Consumers,
ii. Number of elements that this thread of consumer should consume. This is the total number of elements to be consumed divided by the total number of consumers (if it is evenly matched),
iii. The ID, which is the number of the consumer thread (i.e.: the first consumer thread should be 1, the second should be 2, etc.)

4. Main / Calling program (`Coordinator.java`): Your main program should accept the following command line arguments and must be followed in this order:

i. Number of elements in buffer/buffer size,
ii. Number of items to be produced and consumed,
iii. Number of consumers (between 1 and 3)
iv. Number of producers (between 1 and 3)
v. Seed (This will be referred to as seed, mentioned later on in the document)

Each producer thread terminates when the specified number of items has been produced. The main/calling program should print a message like this:

`Producer(s): Finished producing 18 items with checksum being 864`

Each consumer thread terminates when the specified number of items has been consumed. The main/calling program should print a message like this:

`Consumer(s): Finished consuming 18 items with checksum being 864`

(Note: Consumers’ statement is expected to be underlined in the output)

**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()`

### 2. Task Requirements

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 consumed should be equally distributed among the consumer threads (whenever possible). Verify that the number of elements can be perfectly divided among the consumers with no fractions involved, if not arrange for one of the consumer threads to handle the difference. Also, a seed is to be passed to the producer.

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

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

5. Make sure that the printing requirements are met, specifically the underlined statements.

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 microsecond resolution). The code to get the timestamp with microsecond resolution is provided to you, in Coordinator.java. The checksum of all Producer threads should be obtained after the join of each Producer object, by calling `getChecksum()` in the Producer class. The checksum of each Consumer thread is obtained after each Consumer thread join, by calling `getChecksum()` in the Consumer class. The sum of all the Consumer checksums should be equal to the sum of all the Producer checksums.

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 consumers, producers, number of elements, and buffer size.

### 3. Files Provided

Files provided for this assignment include: the description file (this file), a README, and skeleton Coordinator.java and Producer.java files. This can be downloaded as a package from the course Canvas assignment page.

Please refer to the README.txt file inside the package for the questions. You need to answer the questions in the README file.

### 4. Example Outputs:

1. **Example 1**: buffer of 10, 18 items, 3 consumers, 1 producer, and seed of 4

   ```
   ~$ java Coordinator 10 18 3 1 4
   Producer   1 inserted  62  at  index  0 at time 2021-03-16 15:26:50.924314
   Producer   1 inserted  52  at  index  1 at time 2021-03-16 15:26:50.943968
   Producer   1 inserted   3  at  index  2 at time 2021-03-16 15:26:50.944288
   ```
Producer 1 inserted 58 at index 3 at time 2021-03-16 15:26:50.944606
Producer 1 inserted 67 at index 4 at time 2021-03-16 15:26:50.944918
Producer 1 inserted 5 at index 5 at time 2021-03-16 15:26:50.945246
Producer 1 inserted 11 at index 6 at time 2021-03-16 15:26:50.945625
Producer 1 inserted 46 at index 7 at time 2021-03-16 15:26:50.945925
Producer 1 inserted 62 at index 8 at time 2021-03-16 15:26:50.946230
Producer 1 inserted 27 at index 9 at time 2021-03-16 15:26:50.946522
Consumer 3 consumed 62 from index 0 at time 2021-03-16 15:26:50.946821
Consumer 3 consumed 52 from index 1 at time 2021-03-16 15:26:50.947194
Consumer 3 consumed 3 from index 2 at time 2021-03-16 15:26:50.947475
Consumer 3 consumed 58 from index 3 at time 2021-03-16 15:26:50.947749
Consumer 3 consumed 67 from index 4 at time 2021-03-16 15:26:50.948013
Consumer 3 consumed 5 from index 5 at time 2021-03-16 15:26:50.948309
Consumer 1 consumed 11 from index 6 at time 2021-03-16 15:26:50.948888
Consumer 1 consumed 46 from index 7 at time 2021-03-16 15:26:50.949221
Consumer 1 consumed 62 from index 8 at time 2021-03-16 15:26:50.949461
Consumer 1 consumed 27 from index 9 at time 2021-03-16 15:26:50.949716
Producer 1 inserted 92 at index 0 at time 2021-03-16 15:26:50.949997
Producer 1 inserted 8 at index 1 at time 2021-03-16 15:26:50.950263
Producer 1 inserted 52 at index 2 at time 2021-03-16 15:26:50.950518
Consumer 1 consumed 92 from index 0 at time 2021-03-16 15:26:50.950771
Consumer 1 consumed 8 from index 1 at time 2021-03-16 15:26:50.951083
Producer 1 inserted 60 at index 3 at time 2021-03-16 15:26:50.951357
Producer 1 inserted 62 at index 4 at time 2021-03-16 15:26:50.951727
Producer 1 inserted 52 at index 5 at time 2021-03-16 15:26:50.951972
Producer 1 inserted 69 at index 6 at time 2021-03-16 15:26:50.952201
Consumer 2 consumed 52 from index 2 at time 2021-03-16 15:26:50.952427
Consumer 2 consumed 60 from index 3 at time 2021-03-16 15:26:50.952676
Consumer 2 consumed 62 from index 4 at time 2021-03-16 15:26:50.952900
Consumer 2 consumed 52 from index 5 at time 2021-03-16 15:26:50.953186
Consumer 2 consumed 69 from index 6 at time 2021-03-16 15:26:50.953411
Producer 1 inserted 76 at index 7 at time 2021-03-16 15:26:50.953652
Consumer 2 consumed 76 from index 7 at time 2021-03-16 15:26:50.953872

Producer(s): Finished producing 18 items with checksum being 864
Consumer(s): Finished consuming 18 items with checksum being 864

2. **Example 2:** buffer of 7, 21 items, 3 consumers, 3 producers, and seed of 7

```
~$ java Coordinator 7 21 3 3 7
Producer 1 inserted 36 at index 0 at time 2021-03-16 15:28:54.517321
Producer 1 inserted 64 at index 1 at time 2021-03-16 15:28:54.537506
Producer 1 inserted 85 at index 2 at time 2021-03-16 15:28:54.537830
Consumer 3 consumed 36 from index 0 at time 2021-03-16 15:28:54.538197
Consumer 3 consumed 64 from index 1 at time 2021-03-16 15:28:54.538595
Consumer 3 consumed 85 from index 2 at time 2021-03-16 15:28:54.538967
Producer 3 inserted 36 at index 3 at time 2021-03-16 15:28:54.539566
Producer 3 inserted 64 at index 4 at time 2021-03-16 15:28:54.539922
Producer 3 inserted 85 at index 5 at time 2021-03-16 15:28:54.540271
Producer 3 inserted 44 at index 6 at time 2021-03-16 15:28:54.540572
Producer 3 inserted 80 at index 0 at time 2021-03-16 15:28:54.540844
```
5. What to Submit

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

- All .java files listed below and descriptive comments within,
  - Coordinator.java
  - Producer.java
  - Consumer.java
  - Buffer.java
- a Makefile that performs both a make build as well as a make clean,
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program, and answers for the 4 questions

For this and all other assignments, ensure that you have submitted a valid .zip/.tar file. After submitting your file, you can download it and examine it to make sure it is indeed a valid zip/tar file, by trying to extract it.
**Filename Convention:** The archive file must be named as: `<FirstName>_<LastName>_HW5.<tar/zip>`. E.g. if you are John Doe and submitting for assignment 1, then the tar file should be named John-Doe-HW5.tar

6. **Grading**

The assignments much compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux/Mac OS X, but not on the Lab machines are considered unacceptable.

The grading will also be done on a 100 point scale. The points are broken up as follows:

<table>
<thead>
<tr>
<th>Objective</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly performing Tasks 1-8 (10 points each)</td>
<td>80</td>
</tr>
<tr>
<td>Providing a working Makefile.</td>
<td>5</td>
</tr>
<tr>
<td>Questions in the README file</td>
<td>5</td>
</tr>
<tr>
<td>Text alignment and underlining</td>
<td>5</td>
</tr>
<tr>
<td>Descriptive comments</td>
<td>5</td>
</tr>
</tbody>
</table>

**Questions:** (see README.txt file in the skeleton provided)

**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]. Java has advanced classes for synchronization. These cannot be used for this assignment. Hence, 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`

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

[R4]. 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()`.

You are required to **work alone** on this assignment.

**Notes:**

1. The output should have Consumer statements underlined. You can use "\033[0;4m" to start underline formatting and "\033[0;0m" to stop the underline 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 the formatter to display the ID aligned next to the Consumer and next to the Producer. In other words, outputs from both, Consumer and Producer, should be aligned.

4. The number of elements to be consumed might be a multiple of the number of consumers (when divided evenly), otherwise one of the consumers might have to take on a few more elements.

5. The number of elements to be produced might be a multiple of the number of producers (when divided evenly), otherwise one of the producers might have to take on a few more elements.

6. 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 using command line.

---

7. **Late Policy**

Click here for the class policy on submitting late assignments.

**Revisions:** Any revisions in the assignment will be noted below.

03/16/2021 – Revised the outputs and the number of producers/consumers, (each 1 to 3) to be tested on.