Colorado State University

CS 370: OPERATING SYSTEMS Fall 2024

HW5: Programming Assignment v11.5.2024.05:00PM

Evaluating the Producer and Consumer Problem via a Virtual Network Simulation using Bounded Buffers and Java Concurrency Mechanisms

In this programming assignment, you will create a multi-threaded virtual network simulation that creates a network of N Nodes each with K neighbors. Nodes will use Java concurrency mechanisms to produce and consume Messages that they exchange with their neighbors, keeping track of the sum and count of Messages sent and received. The simulation will send M total Messages distributed deterministically across all the nodes in the system.

Due Date: Thursday, November 7, 2024, 11:59PM

Extended Due Date (with 10% penalty per day): Saturday November 9, 2024, 11:59PM

1. Description of Task

You are required to implement this assignment in Java. This simulation assumes that there are N Nodes each with K neighboring Nodes, and that there will be a total of M Messages sent.

You will pass a seed, the numbers of Nodes, neighbors, the MessageBuffer size, and the total number of Messages to be sent as arguments to the Virtual Network Simulation program. The Simulation will generate an overlay that represents a ***connected*** and ***directed*** graph of Nodes, where each Node has a set of exactly K neighboring Nodes that it may send Messages to. Nodes can receive Messages from any Node in the simulation. After the network overlay is generated, the Simulation will spawn Node threads and place them in an array backed Thread Pool. Once all Nodes are initialized and signal that they are ready, the Simulation will signal all Nodes to begin the exchange of Messages by calling **.start()** on each of the Nodes.

Each Node will produce and send M/N total Messages distributed randomly among its K neighbors. Nodes are responsible for keeping track of the counts and sums of messages sent and messages received in total and for each neighboring Node**.** These counts and sums will be collected by the **Simulation** at the end and will be used to verify program correctness.

Every Node thread must have a unique ID and will also have an array of references to its neighboring Nodes, a Random object seeded with the sum of the provided seed and the Node's ID, and a MessageBuffer that holds received Messages. The MessageBuffer size will be specified by argument.

Furthermore, each Node must have K Producer threads and K Consumer threads each stored in an array of their respective types. The Producer threads will use their Node's public methods to generate random messages that they can send to any of the neighboring Nodes' MessageBuffer.

The Consumer threads will poll the Node's MessageBuffer (see below for more details) for Messages, processing the incoming messages and updating the counts and sums and logging output as shown in the example below. If there are no messages in the MessageBuffer**,** then the Consumer must wait.

Once the Nodes have sent all their messages, they will signal to the simulation that they have finished sending. The simulation should wait a few seconds to ensure all messages have been received and processed before collecting the results from each `Node` and printing them to Standard Output in the format shown in the example below.

2. Task Overview

You must implement the following Classes, and the behavior specified above, through specific methods and fields as defined in the included UML diagram (Sec 3). You are encouraged to define various additional helper methods to assist you with your implementation. However, you **MUST** implement all the Classes, Fields, and Methods exactly as defined in the provided UML diagram. This will ensure that your submission is compatible with the grading test suite.

`VirtualNetworkSimulation.java`

- `VirtualNetworkSimulation` is the underlying `Simulation` class and program entry point. It handles the creation of the Network Overlay, instantiation of `Nodes`, signaling the initiation of the `Node's` message exchange, and final accounting of the simulation results.
- Takes the following arguments: Seed, N nodes, K neighbors, B buffer size, and M the quantity of Messages to be sent in total.
- The Network Overlay must be a 2D int array (Matrix) of dimensions NxK representing connected graph where each Node has K neighbors, and each Node has references to all adjacent `Nodes`.
- E.G. For **overlay[i][j]**:
  - The i^{th} index is **nodeID**
  - The j^{th} index is **neighborNodeID**
- The `simulation` will spawn all `Nodes` and store a reference to each `Node` in an array backed thread pool of length N.
- The `simulation` shall call **node.start()** on each of the `Nodes` in the thread pool to start its thread and initiate the message sending process.

`Node.java`

- `Node` represents a `Node` in the simulated network. It will have a unique and incrementally assigned Node ID in the range [0, N-1], an array with references to its K neighboring `Nodes`, a Random object seeded with the sum of the provided seed and its Node ID, K `Producer` threads, and K `Consumer` threads stored in arrays of their corresponding type.
- Node constructor takes the following arguments: String ID, Long seed, Long messagesToSend, int K, int bufferSize
- `Node` must extend the Thread class.
- Each `Node` will expose its `MessageBuffer`, this is where all `Messages` directed towards that `Node` must be sent.
- `Node` is responsible for generating the Longs for the `Messages`, creating and handling its `Producer` and `Consumer` threads, and keeping track of the sum and count of all the `Messages` sent and received. Each message should be sent and received exactly once.
- `Message` Values should be between [1, 1024)

Message.java

- Message is a data object class that Nodes pass between themselves.
- Each Message must contain the following public fields:
  String: src
  String: dst
  Long: message
- Used by the Node, MessageBuffer, Producer, and Consumer classes.

MessageBuffer.java

- MessageBuffer is a FIFO Bounded Buffer that can hold a fixed number of items. You should implement this as an array backed Circular Buffer.
- Your MessageBuffer implementation must be **Thread Safe.**
- There should be exactly ONE instance of MessageBuffer per Node.
- All Messages sent to a Node by other Node's Producers must be sent to this MessageBuffer.
- The MessageBuffer size is the same for all Nodes and this value is specified as a command line argument on program execution.

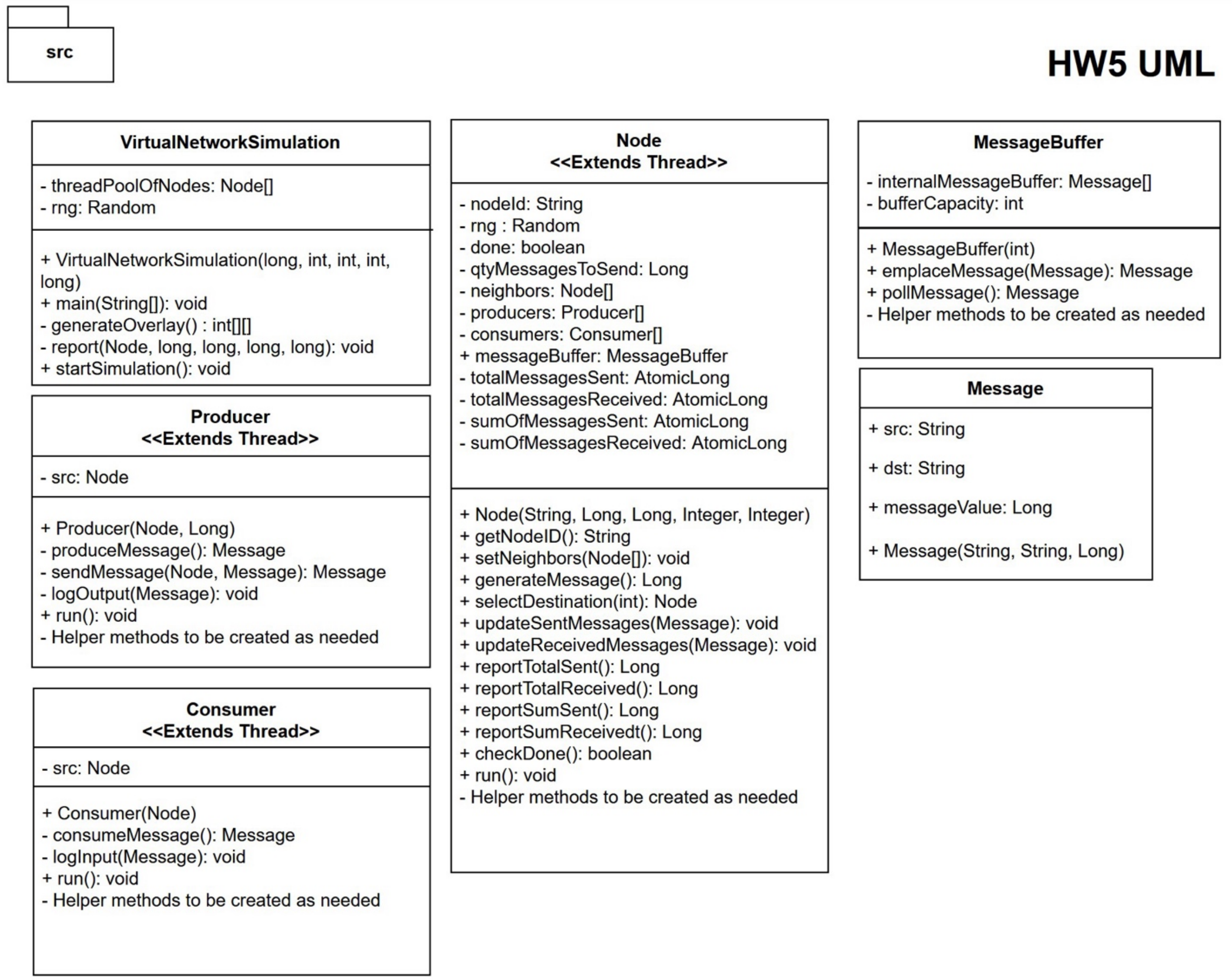
Consumer.java

- Consumers extend the Thread class and are used inside Nodes to handle Message consumption via the MessageBuffer and updating the total and sum messages received counters using the src Node's **updateSentMessages(Long)** method.
- Consumers poll Messages directly from their src Node's MessageBuffer, log the incoming Messages, and update the counts and sums as specified above.
- Consumers must wait if the MessageBuffer is empty.

Producer.java

- Producers extend the Thread class and are used inside Nodes to handle Message production and dispatch concurrently.
- Each Producer will produce M/NK messages
- Producers produce Messages and determine the Message recipient using the Node's public MessageBuffer object
- Producers log message output and update their src Node counters using the src Node's **updateReceivedMessages(String, Long)** method.
- Producers determine which Node to send the Messages to other using the following: **messageValue % K**
  - This creates an integer that will be the index of the neighboring Node that the Producer should send the Message to.
- If the destination MessageBuffer is full, then the Producer should wait for space to become available in the destination MessageBuffer.

3. UML Diagram



+ denotes public, - denotes private. Format: (public/private) name(arg_types): return_type

4. Task Requirements

1. Implement all Classes, Fields, and Methods defined in the UML diagram and produce an executable JAR file that runs the `simulation` program. This jar should be run the following way:

   java -jar LastName-FirstName-VirtualNetworkSimulation.jar Seed N K B M

2. Each `Node` N_i in the simulation will send M/N of the total Simulation `messages` M distributed among each of its K neighboring `nodes`, keeping track of the total count and sum of all messages sent received. The count and sum will be used to verify the program's correctness.
3. Every `Node` must be implemented as its own thread, furthermore each node must also have K `Producers` and K `Consumers`.
4. Implement the FIFO Circular Buffer in a thread safe manner and ensure that the buffer holds no more than the max number of `Messages` at any given time.

5. `Consumers` must wait if the `MessageBuffer` is empty.
6. `Producers` must wait if the destination `MessageBuffer` is full.
7. Your solution must satisfy the correctness constraints (ie. You consume each item exactly one, in the order that it was produced, demonstrating this through the value of the count and sum of the messages in addition to logging each message received.)
8. There must be no deadlocks in your program.
9. Your program will be executed several times with various seeds and values for N, K, B, and M so ensure that it works for any combination of `Nodes`, `Producers`, `Consumers`, and `Messages`.

5. Files Provided

Files provided for this assignment shall include the assignment description (this file), and skeleton code of every required class: VirtualNetworkSimulation.java, Node.java, MessageBuffer.java, Producer.java, Consumer.java.

6. Example Output

javac -d target src/*.java

jar -cfe Simulation.jar src.VirtualNetworkSimulation -C target .

java -jar Simulation.jar 100 2 1 10 100

Generating overlay...

Node 0 Peer: 1

Node 1 Peer: 0

Node 1: Sent 172 to Node 0.

Node 1: Sent 61 to Node 0.

Node 1: Sent 43 to Node 0.

Node 0: Received 172 from Node 1.

Node 0: Received 61 from Node 1.

Node 0: Received 43 from Node 1.

Node 0: Received 213 from Node 1.

Node 0: Sent 76 to Node 1.

Node 1: Received 76 from Node 0.

... output omitted ...

Node 0: Received 187 from Node 1.

Node 1: Sent 187 to Node 0.

Node 1: Received 879 from Node 0.

Node 0: Sent 879 to Node 1.

Node 0: Sent 819 to Node 1.

Node 0: Sent 809 to Node 1.

Node 0: Sent 388 to Node 1.

Node 0: Sent 152 to Node 1.

Node 1: Received 819 from Node 0.

Node 1: Received 809 from Node 0.

Node 1: Received 388 from Node 0.

Node 1: Received 152 from Node 0.

Node 0 statistics:

Sum Sent: 27330

Sum Received: 25174

Total Sent: 50

Total Received: 50

Node 1 statistics:

Sum Sent: 25174

Sum Received: 27330

Total Sent: 50

Total Received: 50

Final Results:

Total Messages Sent -> 100

Total Messages Received -> 100

Global Sum Sent -> 52504

Global Sum Received -> 52504

7. What to Submit

- One zip file: **Lastname-Firstname-HW5.zip** containing the fully implemented .java files, the .jar file that will be used to run the program, and a README.txt file explaining your code.

- Java files must be located in a package called **src**.

8. Revisions

11/1/2024:

- Fixed typos and updated UML diagram.
- Quantity of Messages in the simulation is now passed as a command line parameter.
- Each Node's MessageBuffer is now a public object, simplifying the assignment implementation, eliminating the need for helper methods. Updated UML diagram to reflect this change.

11/4/2024:

- Updated required methods to include Node.reportTotalSent(), Node.reportTotalReceived(), Node.reportSumSent(), Node.reportSumReceived()
- Minor changes to wording to improve assignment clarity.

11/4/2024:

- UML diagram updated.