UNBC Laboratory Assignment
Computer Science 222—Winter 2017

Producers and Consumers via Semaphores

Due Date:

This assignment is due Tuesday, 2017-02-21.

Purpose

The purpose of this lab is to practise designing synchronization objects using JAVA semaphores.

Code all of this assignment in JAVA. Do *not* use the JAVA keyword **synchronized**. Instead use `java.util.concurrent.Semaphore`'s.

A Synchronous Channel

⇒ Build (as a class) a fair synchronized (`String`) channel with operations

- `public void send(String msg);`
- `public String receive();`

⇒ Implement a few simple tests to ensure that your channel works.

⇒ [Bonus.] If you are comfortable with JAVA generics, create a `SynchronizedChannel<E>` that sends and receives `E`s rather than `String`s.

An ASynchronous Channel

It is possible to use the `java.util.concurrent.ArrayBlockingQueue<String>` to implement an asynchronous channel almost directly. However, for this assignment, use only semaphores and an array.

⇒ Implement (as a class) an asynchronous `String` channel, and provide testing. Your channel should have a constructor that lets you set size of the underlying buffer. Your channel should be starvation-free, meaning (for sending) that any thread that attempts to send over the channel will eventually succeed, provided only that there are enough attempts to read from the channel. Similarly, any thread that attempts

to receive from the channel will eventually succeed, provided only that there are enough attempts to send over the channel. However, if there are multiple threads attempting to send over the channel there need not be any particular guarantee about the order in which they succeed.

Hints

The standard solution for one producer and one consumer involves two semaphores, the sum of whose counts equals the size of the buffer. One semaphore tracks the number of buffer slots available for writing to, the second semaphore tracks the number of buffer slots available for reading from. The channel `send` method starts by waiting on the write slot semaphore and ends by signalling the read slot semaphore.

When there is more than one producer and consumer internal channel state modification requires mutual exclusion, which can be provided by addtional semaphores. However it is important that when the `send` code is waiting on the write slot semaphore, that the `receive` code is not locked out, otherwise deadlock can occur.