CS563 Assignment 7: Barriers and Semaphores

Instructor: Xinghui Zhao

Due: 11:59pm, March 15, 2020

1 Overview

The purpose of this assignment is to give you some practice on implementing and using barriers and semaphores.

2 Barriers

2.1 Barrier solution using `await`

Assume there are `n` worker processes, numbered from 1 to `n`. Also assume that your machine has an atomic increment instruction. Consider the following code for an n-process barrier that is supposed to be reusable:

```
int count = 0; go = 0;
```

code for `Worker[1]`:

```
<await (count == n-1)>; #shared variables
count = 0;
go = 1;
```

code for `Worker[2:n]`:

```
<count++;>
<await (go == 1);>
```

a) Explain what is wrong with the above code.

b) Fix the code so that it works. Do not use any more shared variables, but you may introduce local variables.

2.2 Tournament barrier

A *tournament barrier* has the same kind of tree structure as the combining tree barrier we discussed in the lecture, as shown in Figure 1, but the worker processes interact differently than in the tree barrier. In particular, each worker is a leaf node. Pairs of adjacent workers wait for each other to arrive. One "wins" and proceeds to the next level up the tree; the other waits. The winner of the "tournament" at the top of the tree announces that all workers have reached the barrier - i.e., it tells all of them that they can continue.

```
leaf node L:   arrive[L] = 1;
               ⟨await (continue[L] == 1);⟩
               continue[L] = 0;

interior node I:  ⟨await (arrive[left] == 1);⟩
                  arrive[left] = 0;
                  ⟨await (arrive[right] == 1);⟩
                  arrive[right] = 0;
                  arrive[I] = 1;
                  ⟨await (continue[I] == 1);⟩
                  continue[I] = 0;
                  continue[left] = 1; continue[right] = 1;

root node R:  ⟨await (arrive[left] == 1);⟩
              arrive[left] = 0;
              ⟨await (arrive[right] == 1);⟩
              arrive[right] = 0;
              continue[left] = 1; continue[right] = 1;
```

Figure 1: Combining Tree Barrier

Write programs for the workers, showing all details of how they synchronize. Assume that the number of workers, `n` is a power of 2. Either use two sets of variables or reset their values so that the worker processes can use the tournament barrier again on their next loop iteration.

3 Semaphores

3.1 Semaphore Implementation

A simple Java implementation of semaphore is provided. It is a class of a general semaphore with the following operations:

`Semaphore (int s)` // Constructor (sets the initial value, s>=0)

`void P()`

`void V()`

Note that the operation `P()` may throw an InterruptedException. This should just be caught and ignored in your code. Your task is to use the provided semaphore to solve the *atomic broadcast* problem.

If you would prefer to solve this problem using another programming language, it is okay. However, you should make sure that you only use semaphore to implement synchronization. If the programming language provides semaphore, you can use it directly. Otherwise you should first implement a semaphore, and then use it to solve the following problem.

3.2 Atomic Broadcast

Assume one producer process and n consumer processes share a buffer. The producer deposits data into the buffer, consumers fetch them. Every data item deposited by the producer has to be fetched by all n consumers before the producer can deposit new data into the buffer.

1. Develop a solution for this problem using provided semaphore code for synchronization. Note that semaphore is the only synchronization mechanism you can use in this problem.
2. Now assume the buffer has b slots. The producer can deposit data only into empty slots and every data item has to be received by all n consumers before the slot can be reused. Furthermore, each consumer is to receive the data items in the order they were deposited. However, different consumer can receive data at different times. For example, one consumer could receive up to b more data items than another if the second consumer is slow. Extend your previous solution to solve this more general problem.

4 Submission

Submit the following on Blackboard:

1. A PDF file, with pseudo code and answers to the barriers questions;
2. Source code for the atomic broadcast problems;
3. Sample output from your atomic broadcast implementations.

5 Grading Scheme

This assignment will be graded out of 100. For your information, the grading scheme is shown in the following table.

Item	**Percentage**
Await barrier	20%
Tournament barrier	20%
Atomic broadcast	30%
Generalized atomic broadcast	30%