SOFE 4790U Distributed Systems (Fall 2018)

Lab#5

Coordination and Agreement

Objective

In this lab you will be experimenting with, and implementing algorithms for coordination and agreement in distributed systems. **Hint**: starter code is provided for Task#2, so you may want to complete it first.

Note: The Lab tasks must be completed in the lab. If you don't show up for a lab, you will receive a zero and there is no need for you to submit a lab report.

Disclaimer: Parts of this lab have been adapted from (http://www.cs.wustl.edu/~kjg/CS333_SP97/leader.html) and source code from the book (Concurrent and Distributed Computing in Java, by Vijay K. Garg)

Task 1: Use Logical Clocks to Implement Mutual Exclusion (20 marks)

In Task#2 of Lab#4, you have experimented with logical timestamps. In this task, you need to use one of the previously provided classes (LamportClock.java – available on Blackboard, which implements the Lamport logical clock algorithm) to implement the Ricart Agrawala algorithm for distributed mutual exclusion to ensure that at most one process has access to a shared resource at a given time.

Each process keeps a request queue containing the ID and logical time of the request. To request access, a process multicasts the request with its logical time to everyone (and sets state to WANTED). When a request is received, a process that is not in WANTED state or HELD state will reply immediately. If in HELD state, then wait until it exits the critical region, then reply to every request on queue (and sets its state to RELEASED). If in WANTED state, then compare logical time of request with its own re-quest time and only reply if less. Study the details of the algorithm (see below), implement it and test it. For testing, you may consider a process has a loop in which it sleeps for a random length of time and then wants to access the critical region (in which you may also put a sleep statement). A higher resolution figure is included in lab files.

```
On initialization
    state := RELEASED;
To enter the section
    state := WANTED;                          }
    Multicast request to all processes;       } request processing deferred here
    T := request's timestamp;                 }
    Wait until (number of replies received = (N – 1));
    state := HELD;
On receipt of a request <T_i, p_i> at p_j (i ≠ j)
    if (state = HELD or (state = WANTED and (T, p_j) < (T_i, p_i)))
    then
        queue request from p_i without replying;
    else
        reply immediately to p_i;
    end if
To exit the critical section
    state := RELEASED;
    reply to any queued requests;
```

Demonstrate and discuss the results of sample runs with the TA.

Task 2: Implementing the Bully Algorithm for Leader Election (30 marks)

The bully algorithm, which allows processes to crash during an election, elects a coordinator or a leader from a group of distributed processes. The process with the highest ID number from amongst the non-failed processes is elected as the coordinator. Please check the slides to understand how the Bully Algorithm works. Your task is to develop a code to implement Bully Algorithm to elect a leader.

Your code should take the following as input:

1. Number of process.
2. Process id for each process.
3. Status of each process (Live or dead).
4. The process id who will initiate the election.

The output of your code should show all the messages sent between two processes in the correct order and finally which process is elected as the leader.

You can use implement your solution by completing the code provided in the file: BullyElection.java.

As a sample run, consider having 5 processes in total, process 5 is dead and process 3 initiates the election.

```
Enter the number of processes: 5

For process 1...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 1
For process 2...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 2
For process 3...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 3
For process 4...
Status (1 for alive, 0 for dead): 1
Process id (1, 2, 3, .., n): 4
For process 5...
Status (1 for alive, 0 for dead): 0
Process id (1, 2, 3, .., n): 5


Which process will initiate election? 3

Election message is sent from 3 to 4
4 replies to 3
```

```
Election message is sent from 3 to 5
Election message is sent from 4 to 5

Final coordinator is 4
```

Demonstrate and discuss the results of sample runs with the TA.

Lab Report (50 Marks)

1. Submit the source code you have written for Tasks (1) and (2). (10 marks)

2. For Task (1): (20 marks)
    a. How did you design and develop your solution?
    b. What challenges did you face? How did you resolve such challenges? Please discuss.
    c. How is the Ricart Agrawala algorithm different from Lamport Mutual Exclusion algorithm. Please discuss.

3. For Task (2): (20 marks)
    a. Discuss the result of your sample run.
    b. What challenges did you face and how did you solve them?
    c. What's the worst performing case and best performing case when a process initiates the election? How many messages will be sending in both cases? Discuss your answer.

Submit your source code and lab report (in Word or PDF) on Blackboard by 11:59pm on Friday, November 23. No late submissions will be accepted. Your submission should contain only one .zip folder where it will have the report and code (.java files). The name of your zip folder should be "FullName_student id".