CSE5306, Distributed Systems
Fall 2017, Project 2
Due date: 11:59pm Nov. 14, submission through Blackboard

Please read this:
Two students form a team and turn in one submission.
Total points possible: 100 pts.
Please add the following statement in the beginning of your submission.
I have neither given or received unauthorized assistance on this work
Signed: Date:

Introduction

In this programming project, you will develop an *n*-node distributed system that provides a totally ordered multicasting service and a distributed locking scheme. The distributed system uses logical clock to timestamp messages sent/received between nodes. To start the distributed system, each node should synchronize their logical clocks to the same initial value, based on which the ordering of events can be determined among the machines. Use the Berkeley Algorithm to obtain an initial agreement on the logical clocks. Totally ordered multicasting can be implemented using **either** Lamport's algorithm **or** the vector clocks algorithm. Suppose the distributed nodes have read and write access to a shared file. The last task is to implement a distributed locking scheme that prevents concurrent accesses to the shared file. You can use the centralized, decentralized, or the distributed algorithms to realize mutual exclusive access to the file. You are free to use **any programming language**. To simplify the design and testing, the distributed system will be emulated using multiple processes on a single machine. Each process represents a machine and has a unique port number for communication.

Assignment-1 (50pts)

Suppose the logical clock on each machine represents the number of messages have been sent and received by this machine. It is actually a counter used by the process (or the machine emulator) to count events. Randomly initialize the logical clock of individual processes and use Berkeley's algorithm to synchronize these clocks to the average clock. You can select any process as the time daemon to initiate the clock synchronization. After the synchronization, each process prints out its logical clock to check the result of synchronization.

Assignment-2 (50pts)

Implement the totally ordered multicasting for the distributed system. Create two threads for each process, one for sending the multicast message to other nodes and one for listening to its communication port. Use either Lamport's algorithm or the vector clocks to enforce the order of messages. Once a process delivers a received message to a user, it prints out the message on screen. A totally ordered multicasting requires that all process print out received messages in the same order. You can assume that the number of processes (machines) is fixed (equal to or larger than 3) and processes will not fail, join, or leave the distributed system. Implement two versions of this program, one without totally ordered multicasting and one with this feature. Compare the results of the two programs. (HINT: Each process needs to maintain a message buffer, from where messages can be sorted and delivered in order).

Deliverables

The deliverables include the source code of the programs, a README file containing instructions on how to compile and run your programs, and a report that briefly describes how you implemented the programs, what you have learned, and what issues you encountered. Put all the requirement documents into a zipped folder. Make sure you clearly list your names and student IDs in the report.