CSE5306, Distributed Systems
Fall 2022, Project 2
Due date: 11:59pm Nov. 5, submission through Canvas

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 supports totally ordered events and a distributed locking scheme. The distributed system uses logical clock to timestamp messages sent/received between nodes. You will realize a total order of event through the totally ordered multicast and vector clocks. Suppose the distributed nodes have read and write access to a shared file. A distributed locking scheme is needed to prevent 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 languages**. 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 uses a unique port for communication. 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.

Assignment-1 (35pts)

Implement totally ordered multicasting using Lamport's algorithm. Each process conducts local operations and numbers them as PID.EVENT_ID. After each operation is done, a process multicasts the event to all other processes in the distributed system. The expected outcome of this assignment is that events occurred at different processes will appear in the same order at each individual process. To realize such a total order of events, each process maintains a buffer for received events and follow the rules on slide 19 in Lecture 6 when delivering the events. In this assignment, the delivery of events is simply printing them on screen, in the format of CURRENT_PID: PID.EVENT_ID. HINT: You may use two threads in each process to handle the communication and deal with message delivery, respectively. Refer to the multi-threaded server example discussed in class. The communication thread enqueues updates in a buffer, from where the delivery thread enforces a total order of events. The communication thread is also responsible for sending acknowledgements for received messages.

Assignment-2 (35pts)

Implement the vector clock algorithm to enable causally-ordered events in the distributed system. Similar to the previous assignment, we use multiple processes to emulate multiple nodes. Assume that all nodes are initiated to the same vector clock. After completing a local operation, each process sends its updated vector clock to all other processes. Follow the steps on slide 34 in Lecture 6 to implement the vector clock algorithm.

Assignment-3 (30pts)

Implement a locking scheme to protect a share file in your distributed system. While we use processes on a single machine to emulate the distributed system and there are shared-memory synchronization primitives available, you are required to implement one of the following distributed locking schemes: decentralized, distributed locking or the token ring algorithm. When a process acquires the lock, it simply opens the file, increments a counter in the file, and closes the file. Assume that all processes keep requesting the lock until successfully acquiring the lock for a predefined number of times.

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.