CMSC 621: PROJECT 2

ASSIGNED DATE: 10/17/2017
DUE DATE: 11/07/2017, 11:59 PM

CLOCKS, MULTICAST, AND COMMIT

In this programming project, you will develop an *n*-node distributed system that provides a causally 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. Causal ordered multicasting can be using the algorithm discussed in class. 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. 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 (60PTS)

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 (40PTS)

Implement the causal 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 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. 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 causally ordered multicasting and one with this feature. Compare the results of the two programs.

BONUS ASSIGNMENT (20PTS)

Add the feature of distributed locking to your program to protect a shared file. The file only contains a counter that can be read and updated by processes. Implement two operations: *acquire* and *release* on a lock variable to protect the file. At the beginning, each process opens the file and tries to update the counter in the file and verifies the update. Thus, the critical section includes the following operations: (1) point the file offset to the counter; (2) update the counter; (3) read and print out the counter value. You can use any of the mutual exclusion algorithms learned in class to implement the locking. The expected result is that the read of the counter value always matches the updated value by a process if locking is enabled.

DELIVERABLES

The following are the deliverables for the project

(a) The source code of the programs that you will submit
(b) A Makefile that compiles the code based
(c) A README file containing instructions on how to compile and run your programs.
(d) A report that briefly describes how you implemented the programs, what you have learned, and what issues you encountered.
(e) Put all the requirement documents into a zipped folder. Make sure you clearly list your names and student IDs in the report.
(f) Submit the zip file to the TA over slack.