CS 6378: Advanced Operating Systems
Programming Assignment 2

Instructor: Ravi Prakash

Assigned on: March 1, 2024
Due date: March 21, 2024

This is an individual project. Sharing of code among students or using fragments of code written by others is strictly prohibited, and will be dealt with as per the university's rules governing academic misconduct. You are expected to demonstrate the operation of your project to the instructor or the TA.

Requirements

1. Source code must be in the C/C++/Java programming language.
2. The program must run on UTD lab machines (`dc01, dc02, ..., dc45`).
3. You will need to know thread and socket programming and its APIs for the language you choose. It can be assumed that each process (server/client) is running on its own machine (`dcXY`), where $01 \leq XY \leq 45$, and two processes communicate through a reliable socket connection between them. Please get familiar with basic UNIX commands to run your program on *dcXY* and UNIX/Linux system calls for directory and file operations.

Project Description

Consider a set of four servers, each of them maintaining a replica of the file system. There are multiple clients who may wish to perform writes to the file system. A client's write is routed to the closest server that has a replica. It is that server's responsibility to ensure that the update happens locally and also at the other three servers. To perform such updates, the server that received client request broadcasts the update to all servers (including itself). Some of these client requests may have causal dependencies among them. Hence, messages carrying updates that are causally related should be delivered in the right causal order at all servers. Additionally, to ensure consistency of replicas, messages carrying updates that are mutually concurrent should be delivered in the same order at all servers. What we have described above is totally ordered message delivery. In this project, you can ignore the clients in the sense that each client's requests have been received by the closest server. The only communication you have to deal with it that between the four servers.

In this project you will extend Project 1 to implement totally ordered message broadcasting, as defined below, among the four processes:

1. Each message is broadcast to all processes. So, if process P_x is the source of a broadcast, the message is sent to all processes, including P_x.
2. If $Send(m_a) \rightarrow Send(m_b)$, then at all four destinations, $Delivery(m_a) \rightarrow Delivery(m_b)$.
3. If $Send(m_a) \parallel Send(m_b)$ then m_a and m_b are delivered at all nodes in the same relative order.

For the purpose of total ordering you can use any algorithm/protocol described in scientific literature. Make sure in your report you briefly describe the algorithm you used (including how it guarantees message delivery in the desired order), and provide appropriate citations.

You are prohibited from using a fifth process to act as the coordinator for message ordering. All ordering decisions must be made among the four processes.

Execution terminates when each process has generated at least one hundred messages for broadcasting, and all those messages have been correctly delivered to all destinations.

Submission Information

The submission should be through eLearning in the form of an archive consisting of:

1. File(s) containing the source code.
2. The makefile or corresponding instructions used for compilation purposes.
3. The directories and files you used to test the execution of your code.
4. A PDF document containing description of the totally ordered delivery solution you have used, along with relevant citations.

Your source code must have the following, otherwise you will lose points:

1. Proper comments indicating what is being done
2. Error checking for all function and system calls