CS 6378: Programming Project I

Instructor: Ravi Prakash

Assigned on: September 9, 2019
Due date: September 30, 2019

This is an individual project and you are required to demonstrate its operation to the instructor and/or the TA to get credit for the project.

In this project, you are required to implement the Ricart-Agrawala algorithm for distributed mutual exclusion, with the optimization proposed by Roucairol and Carvalho, in a client-server model.

1 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/or socket programming and its APIs for the language you choose. It can be assumed that each process (server/client) is running on a single machine (`dcXY`). Please get familiar with basic UNIX commands to run your program on *dcXY*.

2 Description

1. There are three servers in the system, numbered from zero to two.
2. There are five clients in the system, numbered from zero to four.
3. Assume that each file is replicated on all the servers, and all replicas of a file are consistent in the beginning. To host files, create separate directory for each server.
4. A client can perform a READ or WRITE operation on the files.
   (a) For READ operation, one of the servers is chosen randomly by the client to read from it.
   (b) For WRITE operation, the request should be sent to all of servers and all of the replicas of the target file should be updated in order to keep them consistent.
5. READ/WRITE on a file can be performed by only one client at a time. However, different clients are allowed to concurrently perform a READ/WRITE on different files.
6. The supported operations by servers are as follows:
   (a) ENQUIRY: A request from a client for information about the list of hosted files.
   (b) READ: A request to read last line from a given file.
   (c) WRITE: A request to append a string to a given file.

   The servers must reply to the clients with appropriate messages after receiving each request.
7. Assume that the set of file does not change during the program's execution. Also, assume that no server failure occurs during the execution of the program.

8. The client must be able to do the following:
   (a) Gather information about the hosted files by querying the servers and keep the metadata for future.
   (b) Append a string $\langle client\_id, timestamp \rangle$ to a file f_i during WRITE operation. *Timestamp* is the value of the clients' local clock when the WRITE request is generated. This must be done to all replicas of f_i.
   (c) Read last line of a file f_i during READ.
9. Write an application that periodically generates READ/WRITE requests for a randomly chosen file from the set of files stored by the servers.
10. Display appropriate log messages to the console or to a file.
11. Consider this project as having two phases:
    (a) **Phase One**: Entering *Critical Section*.
    (b) **Phase Two**: Performing *Read/Write* operations on servers.

2.1 Phase One

It does not concern servers, it is just between clients. Each client upon requesting a file operation, regardless of the operation type, sends its request to other four clients, and waits for their replies. Each client upon receiving a request, sends its reply to the requesting client if (i) it has no request to send, or (ii) its own request has higher timestamp than the received request.

The requester can enter its critical section when it receives all four replies. Upon entering its critical section, it will not send any reply to any other client, and will keep them in a queue. Whenever it exits its critical section, it sends all delayed replies to their requesters.

In order to ensure the mentioned conditions, you must implement Ricart-Agrawala algorithm for distributed mutual exclusion, with the optimization proposed by Roucairol and Carvalho, so that no READ/WRITE violation could occur. The operations on files can be seen as *critical section* executions.

2.2 Phase Two

It is the critical section part of each client. Each client upon entering its critical section, sends its operation to all three servers. Each server upon receiving an operation request from a client, performs the requested operation and sends an appropriate message to the requester client, should it finishes the operation. The client upon receiving all three messages from three servers, will exit its critical section and if it has delayed any reply to the requests of other clients, it will send them.
Keep in mind to close all open sockets when your program exits/terminates.

3 Submission Information

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

1. File(s) containing the source code. Your source code must have the following, otherwise you will lose points:
   (a) Proper comments indicating what is being done.
   (b) Error checking for all function and system calls.
2. The README file, which describes how to run your program.