CS/CE/TE 6378: Project I

Instructor: Ravi Prakash

Assigned on: February 1, 2020
Due date and time:February 19, 2020, 11:59 pm

This is an individual project and you are expected to demonstrate its operation to the instructor and/or the TA. Sharing of code among students, or reusing code from previous semesters or from other sources without the permission of the instructor is prohibited. Any violation of this policy can result in the student(s) being reported to appropriate authorities for academic misconduct.

1 Requirements

- Source code must be in the C /C++ /Java programming language.
- The program must run on UTD lab machines (dc01, dc02, ..., dc45).

2 Client-Server Model

In this project, you are expected to implement *Lamport's Mutual Exclusion Algorithm.* Knowledge of threads and socket programming and its APIs in the language you choose is expected. Each process (server/client) must execute on a separate machine (dcxx).

3 Description

Implement a solution that mimics a replicated file system. The file system consists of four text files: $f1$, $f2$, $f3$, *and* $f4$. All four files are replicated at three file servers. To achieve file replication, choose any three of the dcxx machines. Each of these machines will store a copy of all the files. Five clients, executing at different sites (and different from the three sites that replicate the files), may issue *append to file* requests for any of these files. The clients know the locations of all the other clients as well as the locations of all file servers where the files are replicated. All replicas of a file are consistent, to begin with. The desired operations are as follows:

- A client initiates at most one *append to file* at a time.
- The client may send a *append to file* REQUEST to any of the file replication sites (randomly selected) along with the text to be appended. In this case, the site must report a *successful* message to the client if **all** copies of the file, across the all sites, are updated consistently. Otherwise, the append should not happen at any site and the site must report a *failure* message to the client. We do not expect a *failure* to happen in this project unless a file is not replicated at exactly four different sites.
- On receiving the *append to file* REQUEST, the receiving site initiates a REQUEST to enter critical section, as per Lamport's mutual exclusion algorithm. Obviously each REQUEST should result in a critical section execution regarding that particular file. In the critical section the text provided by the client must be appended to all copies of the file.
- Concurrent appends initiated by different clients to different files must be supported as such updates do not violate the mutual exclusion condition.

- Your program does NOT need to support the creation of new files or deletion of existing files. However, it must report an error if an attempt is made to append to a file that does not exist in the file system.
- All clients are started simultaneously.
- Each client loops through the following sequence of actions 100 times: (a) waits for a random amount of time between 0 to 1 second, (b) issues an *append to file* REQUEST, (c) waits for a *successful* or *failure* message. So, for this project you need to worry about concurrent read/write requests for the same file but issued by different clients. Your client should gracefully terminate when all 100 REQUESTS have been served (either successfully or unsuccessfully).

The teaching assistant will upload a sample configuration file for the servers. You must follow that standard. Your code is evaluated in terms of functionality, readability and documentation (comments and README file). A separate file describing your code is appreciated but is not mandatory.

4 Submission Information

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

- File(s) containing the source code.
- The README file, which describes how to run your program.
- DO NOT submit unnecessary files.