CS 171 Spring 2019
Programming Assignment II

Assigned: April 21st, 2019
Due On Demo Day: May 10th 2019

PLEASE NOTE:

- Solutions have to be your own.
- It is an **INDIVIDUAL** programming assignment.

1 Introduction

In the first programming assignment, you successfully implemented the clock synchronization in a bank system. In this programming project, you will explore how to use the distributed mutual exclusion problem to maintain the replicated ledger of a bank system using Lamport's algorithm.

The bank system consists of three servers which maintain a fully replicated shared resource, an append-only data ledger of client account information, ie, all transactions the clients executes. To ensure consistency and prevent interference when accessing this resource, the bank system uses Lamport's protocol as the underlying mechanism to ensure mutual exclusion.

2 Implementation details

This project has following important components:

Shared Ledger: For simplicity of implementation we will use a simple text file containing the client ledger information as the shared resource. This file can be operated on by the server using simple *open, write* and *close* operations. All updates will be **append** operations to the end of the ledger.

Client: For simplicity, we will assume three clients, Alice, Bob and Carol, each communicating with one of the given servers. Assume each client starts with $100. A client can issue a transaction at any time, and then waits for the completion of this transaction before issuing the next transaction. A transaction consists of a single money transfer to another

client, eg, Alice sends $10 to Bob. A transaction can be initialized by entering a command from the client command line.

Servers: There will be three server nodes. Each server maintains a copy of the shared (replicated) ledger. Every server also maintains a queue, consisting of the list of pending transactions to be appended to the ledger. All items in the request queues are sorted by message timestamps. Before appending a transaction to the local copy of the ledger, the server must first verify that the transaction is *legal*, ie, the server must check that the client actually has the resources to execute the transaction, eg, if Alice wants to send Bob $50 transaction, the server will need to check that Alice actually has the funds in her account. This requires checking the initial amount ($100) + all the transfers **to** Alice - all the transfers **from** Alice.

Messages: All messages (i.e. REQUEST, REPLY, RELEASE) are sent reliably and in FIFO order. Each message is time stamped with unique Lamport timestamps (i.e. logical Lamport time + proc id).

Channels: The channels in this protocol have to be **directed**. (i.e. there should be 2 edges between any two sites. Please see the network communication example below.)

Network: As in Programming Assignment 1, a network process will be added to simulate a random network delay. All the requests that are sent between the servers will be sent to this network process first and then the network process would add a delay and send the request to the server.

The flow of algorithm will be as follows:

- The server that wants to execute a transaction sends a timestamped REQUEST for the resource to other servers and puts the request in its local queue.
- Every recipient puts the received request in its local queue, which is sorted in timestamp order, and sends back REPLY to the server who requested the resource.
- The server (that wants to use the resource) can access the shared resource if it receives a reply from all servers and its own request is at the head of the queue. At this point, the given server can access its copy of the shared ledger. It first verifies that the transaction is legal to execute. If it is *not legal*, it sends a message to the client informing it of the reason, eg,*insufficient funds*. If *legal*, it appends the new transaction to its local copy of the ledger.
- Once the server is done with the resource, it will send a RELEASE message to all servers and remove the request from its local queue. If the transaction was legal and was appended to the local ledger copy, the RELEASE message will contain the transaction. On receipt of a RELEASE message by another server, this server appends the attached transaction (if one is attached) to the local copy of the ledger and then removes the corresponding request from their local queue.

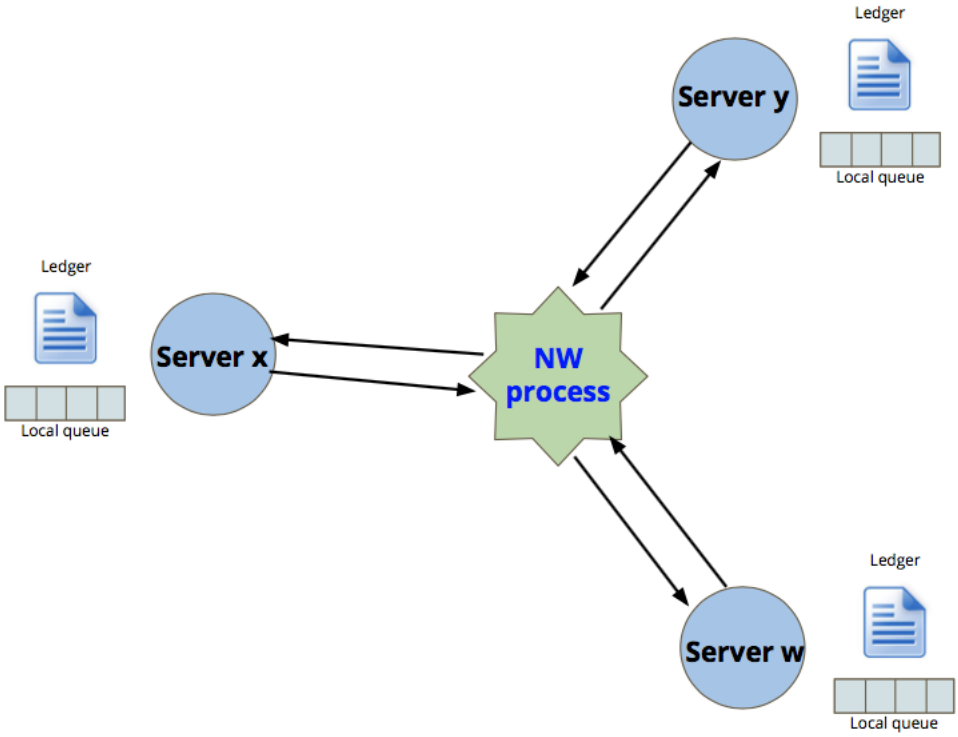

Figure 1: Example

NOTE:

1. Next week, we will send you more information on what to print during your demo.
2. You should have a configuration file that contains all the IP and ports, so when starting a server, it should know the IP and port numbers of other servers.
3. We do not want any front end UI for this project. Your project will be run on the terminal and the input/output for the demo will use stdio.
4. Use message passing primitives TCP/UDP. You can decide which alternative and explore the trade-offs. We will be interested in hearing your experience.

3 Demo

We will have a short demo for this project. It will be on **May 10th, 2019** in CSIL. Time details will be announced later. Please be ready with the working program at the time of your demo.