Lab #2
COMPSCI 677 - Fall 2022

Due 11/18/2022 at 11:00 PM EST. Upload a zip to Gradescope and the PDF to Gradescope.

Asterix and the Trading Post

General Instructions:

- You may work in groups of two for this lab assignment.
- This project has two purposes: to familiarize you with some of the important canonical problems you have studied in this course-- synchronization using logical clocks, and leader election algorithms.
- You may find that the evaluation criteria are less rigid than you're used to from other programming assignments. This is because ultimately we're just trying to make sure you've adequately grappled with and understood the concepts above, and do so in a way that mimics a research environment appropriate to a 600-level course. Therefore, you can be creative with this project. You are free to use any programming languages (C, C++, Java, python, etc) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. You have considerable flexibility to make appropriate design decisions and implement them in your program: just be sure you can justify your decisions to your ~~reviewers~~ graders.
- Feel free to make use of online documentation for any languages or libraries you make use of, however, be sure to run any non-standard libraries you use by us, and be sure to package and include them in your submission if necessary. You should also be clear about what tools you make use of in your write-up.
- You should read this entire document before you begin: you'll be expected to document parts of your process of completing this assignment, so you'll have to know which parts before you complete them.

A: The problem
A1: The story (useful for contextualizing)

At the very beginning, the bazaar served the Gauls well. However, with the growing popularity of the bazaar, there were too many whispers that got lost in the din. After a town meeting, Vitalstatix, the village chief, decreed that henceforth the bazaar will adopt a trading post model. The Guals will use a leader election algorithm to elect a trader, who will be in charge of the trading post. All sellers will deposit their goods when the

market opens with the trader. Buyers will buy from the trader and the trader will pay the seller after each sale (after deducting a suitable commission).

A2: The technical requirements

A trader can resign her post at any time; or may call in sick. In this case, the market should elect a new leader. The trader maintains meticulous accounts, so the current state of the market (number of goods available for sale, pending sales) are not lost when a new trader is elected. Use a disk file to maintain these accounts.

While in the bazaar model, concurrent buy requests for the last item on sale were resolved arbitrarily, the trading post has adopted a charter to be fair to all buyers. Accordingly, all buyers and sellers have a logical clock (digital timepieces are yet to be invented, so logical clocks are the only option). Logical clocks are used to order buy events, and if there ~~is only one item~~ are too few items remaining, then concurrent buy requests are resolved using clock values. You need to decide whether to equip each Gaul with a Lamport or Vector clock. If you use Lamport's clocks, be sure to use the multicast version that allows for timestamps to be compared.
Sellers can offer new items for sale any time: like before, the item type (salt, fish, boar) and the number of items available is specified to the trader. Buyers need to specify what item and how many they wish to buy. As per the fair trading charter, purchase prices are determined a priori and are not negotiable. Upon each sale, the appropriate seller is credited with the purchase price. You don’t need to worry about buyer budgets, unless you’d like to include them as part of your design.
Assume that the number of people in the trading post N is specified beforehand (N should be passable as a command line argument into your parent program).

First construct a connected network. No neighbor discovery is needed; you may pass a list of all N peers and their addresses/ports to the peers from their parent program, or specify them in a file with shared read access, or share them in any other way.

Once the network is formed, randomly assign a seller or buyer role to each peer following the description of project 1. In this lab, a peer can be both a buyer or a seller, or only one of the two. Peers will then elect a coordinator using any election method mentioned in the class. The peer does not sell or buy anything once it is elected as the coordinator. If the current coordinator resigns, a new round of election will be triggered for a new coordinator to be chosen. We can assume that the last coordinator will write all the useful information to a file so that the new coordinator can read such information from the file.

Each peer maintains a clock. The clock is used to decide which seller to do business with, once multiple responses are received. In this assignment, you may implement either Lamport logical clocks or vector clocks.

As in Project 1, each peer is both a client and a server.

Each peer should be able to accept multiple requests at the same time. This could be easily done using threads. Be aware of the thread synchronization issues to avoid inconsistency or deadlock in your system.

No GUIs are required. Simple command line interfaces are fine.

B. Evaluation and Measurement

- Deploy at least 6 peers. They can be set up on the same machine or different machines.
- Do a simple experiment study to evaluate the behavior of your system in each circumstance in which you'd expect very different results. Compute the average response time per client search request in each circumstance by measuring the response time seen by a client for, say, 1000 sequential requests. Also, observe the performance change due to the coordinator re-election. Make necessary plots to support your conclusions.
- Optional: Deploy the peers on at least two separate machines. You may deploy and configure them manually for this part of the assignment. Once they're set up, observe the latencies, and compare latencies between local and remote communication and record your comparisons. Do you observe that the ordering of lookups, replies, and buys is as consistent as when peers are all deployed on the same machine? What about when only the leader is deployed on one machine, and the other peers are deployed on other machine(s)? While we won't be awarding more than 100% on this assignment, successfully completing this can make up for lost points elsewhere in this lab.

C. What you will submit

When you have finished implementing the complete assignment as described above, you will submit your solution in the form of a zip file that you will upload to Gradescope and a PDF document uploaded to a separate Gradescope assignment.

Each program must work correctly and be documented. The zip file you upload to Gradescope should contain:

- An .txt file containing the output generated by running your program. When it receives a product, have your program print a locally timestamped message such as "11.01.2022 16:54:32.10 bought product_name from peerID". When a peer issues a query, having your program print the returned results in a nicely formatted manner including the local time that the result is received. When a new coordinator is elected, print a message like "11.02.2022 16:54:32.10 Dear buyers and sellers, My ID is ..., and I am the new coordinator". You may find the '>' or '>>' linux/mac command line operator useful for generating this file.
- Your source code, containing clear in-line documentation.
- A copy of the PDF you'll also be submitting to the other Gradescope assignment.
- A separate README file with instructions on how to run your program

You'll also be submitting a PDF to a separate Gradescope assignment. Be sure to submit one assignment per group of two on this assignment. You may want to use this as an opportunity to learn LaTeX if you haven't already: if you ever plan on writing scientific papers, you'll be using LaTeX to do so. The PDF you upload to Gradescope should contain all the following:

- No more than two or three pages, including diagrams, describing the overall program design, a description of "how it works", and design tradeoffs considered and made. If this document says "do X", you don't need to tell us that you did X: instead focus on *how* you did X, if it isn't too straightforward. We're looking for high-level design, such as files, classes, interesting methods, and maybe a couple of code snippets if you're particularly proud of them, plus anything you had difficulty with or didn't manage to accomplish. Also describe possible improvements and extensions to your program (and sketch with words and/or diagrams how they might be made). You also need to describe clearly how we can run your program - if we can't run it, we can't verify that it works.
- On a separate page, a description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
- On another separate page, include your performance results.
- Finally, if you completed the optional part of this assignment, include your observations on another separate page.

D. Grading policy

Program Listing
    works correctly ---------- 50%
    in-line documentation -------- 15%
Design Document

quality of design ---------- 15%
understandability of doc ------ 10%
Thoroughness of test cases ---------- 10%
Optional separate-machine deployment - 5%
Grades for this assignment will be capped at 100%. This cap applies before any late penalties.

After using your allotted late days for the course, grades for late submissions will be lowered 4 percent per 8 hours late. See the syllabus for the extraordinary circumstances late policy to see if that applies for your circumstances.