Lab #1
COMPSCI 677 - Fall 2024

Due 10/21/2024 at 11:59 PM EST. Upload a zip and a PDF to separate Gradescope assignments.

**Asterix and the Bazaar**

**General Instructions:**

- You may work in groups of two for this lab assignment.
- This project has two purposes: first to familiarize you with sockets/RPCs/RMIs, processes, threads; second to learn the design and internals of a Bazaar-style peer-to-peer (P2P) marketplace.
- 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. 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. Feel free to make use of online documentation for any languages or libraries you make use of. To be clear, the intent is for you to use sockets/RPCs/RMIs in this project, so don't use any libraries that abstract these away!
- 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)***

The year is 50BC. Gaul is entirely occupied by the Romans. Well, not entirely... One small village of indomitable Gauls still holds out against the invaders. The Gauls love to trade goods in their weekly bazaar. The bazaar contains two types of people: buyers and sellers. Each seller sells one of the following goods: fish, salt, or boars. Each buyer in the bazaar is looking to buy one of these three items.

While previously buyers meet sellers by screaming for what one wished to buy or sell, the village chief Vitalstatix has decreed that henceforth, all bazaar business shall strictly follow the peer-to-peer model. Each buyer shall gently whisper their needs to all her

neighbors, who will then propagate the message to their neighbors and so on, until a seller is found or the maximum limit on the number of hops a message can traverse is reached.

If a seller is found, then the seller sends back a response that traverses in the reverse direction back to the buyer. At this point, the buyer and the seller directly enter into a transaction (without using intermediate peers).

### *A2: The technical requirements*

In this assignment you will be simulating the entire bazaar in your chosen computing environment. The participants, i.e. the buyers and the sellers, will be represented by separate processes that you fork from a parent program. The participants (processes) will communicate with each other using the methods you've learned about in lecture: RPC, RMI, and/or sockets. You should recognize this structure as a p2p network, with the participant processes as peers.

Assume that the number of people in the bazaar N is specified beforehand (N should be passable as a command line argument into your parent program).

First construct a p2p network such that all N peers form a connected network. You can use either a structured or unstructured P2P topology to construct the network. All peers should have no more than three direct neighbors, and should only communicate directly (make RPCs, RMIs, or use the sockets of) direct neighbors during the simulation. You should also ensure that the network is fully connected: i.e. there exists a path, if not a direct connection, between any two peers. 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 write access, or share them in any other way.

Once the network is formed, assign each peer a random role: fish seller, salt seller, boar seller, or buyer. You may find it expedient to assure that there is at least one buyer and at least one seller, but make sure you cover the corner cases in which not all types of sellers are represented. Once the roles have been assigned, each buyer randomly picks an item and attempts to purchase it using the interfaces specified below; it then waits a random amount of time, then picks another item to buy and so on. Each seller starts with n items (e.g., n boars) to sell; upon selling all n items, the seller picks another item at random and becomes a seller of that item.

**The peers in your system should implement the following interfaces and components (depending upon their roles):**

- lookup(buyerID,product_name,hopcount[,search_path]): this procedure, executed initially by buyer processes then recursively between directly connected peer processes in the network, should search the network; all matching sellers respond to this message with their IDs. The hopcount, which should be set lower than the maximum distance (of all the shortest paths between any two peers in the network, the length of the longest one) between peers in the network (after the topology is defined), is decremented at each hop and the message is discarded when it reaches 0. This means that your network cannot use a fully connected topology, as the hopcount would never be greater than 0.
- reply(sellerID[,reply_path]): this is a reply message with the peerID of the seller, called by the seller process to execute along the reverse of the path lookup used.
- buy(peerID[,path]): if multiple sellers respond, the buyer picks one at random, and contacts it with the buy message. A buy causes the seller to decrement the number of items in stock.
- Note that the above function signatures are suggestions: as long as you have the specified functionality available, you may use whatever parameters you need to achieve them.
- A peer is both a client and a server.
- As a client, the buyer specifies a product_name using "lookup". Upon receiving replies, the buyer peer picks one matching seller and then connects to that peer to finish the transaction.
- As a server, a seller waits for lookup requests from other peers and sends back a response for each match. Each lookup request is also propagated to all its neighbors. While lookup requests use flooding, a reply/response should traverse along the reverse path back to the buyer without using flooding (one way to meet this requirement is to have each peer append its peerID to a list which is propagated with the lookup message [the optional [search_path], [reply_path], and [path] arguments above]; the list yields the full path back to the buyer; other techniques are possible which involve stateful peers).

**Other requirements:**

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. For instance, a seller should be able to process multiple concurrent buy requests and decrementing the number of items in

stock should be done using synchronization. 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 (different directories) or different machines.
- Do a simple experiment study to evaluate the behavior of your system. Compute the average response time per client search request by measuring the response time seen by a client for, say, 1000 sequential requests. Also, measure the response times when multiple clients are concurrently making requests to a peer, for instance, you can vary the number of neighbors for each peer and observe how the average response time changes, 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? 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 timestamped message such as "10.10.2022 16:54:32.10 bought product_name from peerID". When a peer issues a query (lookup), having your program print the returned results in a nicely formatted and similarly timestamped manner. 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.

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 and creativity --- 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.

See the syllabus for the extraordinary circumstances late policy to see if that applies for your circumstances.