Programming Assignment 3

**Instructions**

* Due by 11:59pm of 11/04/2024.
* Late penalty: **10%** penalty for **each day late**.
* This is an individual assignment, although you may work in groups to brainstorm on possible solutions; your code implementation, report, and evaluation must be your own work.
* Upload your assignment on the **Canvas** with the following name: **Section\_LastName\_FirstName\_PA2.zip**
* Please do **NOT** email your **assignment** to the **instructor** or **TA**!
* Use any popular language, if in doubt ask the TA ASAP.
* All of the submitted document files should be **PDF/Doc format** files. The **handwritten file is not accepted**.

## Overview

In this assignment, you need to change the centralized P2P architecture from Programming Assignment 2 to a de-centralized P2P system. This system requires only one kind of node, the peer node. All topics shall be stored on a distributed hash table (DHT). Each peer node contains a non-overlapping portion of the DHT. And the network topology connecting the peer nodes shall be a hypercube.

1. **Peer nodes**: Same as PA2, each peer node can host topics and serve publish/subscribe to topics from its clients. There will be multiple peer nodes executed by separate processes.
2. The topics hosted by different peer nodes are not overlapping.
3. All the APIs in PA2 are supported in PA3.
4. Communications among the peer nodes will be via asynchronous IO calls.

## Detailed Description of Assignment

A distributed hash table (DHT) is a hash table distributed among multiple nodes. Each node stores a subset of the table. One key property of DHT is all nodes collectively form the system WITHOUT any central coordination. In this assignment, you need to design a hash function that each node can utilize to compute the location of any given topic. Refer to <https://en.wikipedia.org/wiki/Distributed_hash_table> for more information about DHT. To make this assignment simple, you can assume there is no topology changes during execution.

Hypercube is a topology where each node has a unique binary identifier (e.g. 0010). Two nodes are connected if and only if their identifiers differ in one and only one digit (e.g. 0010 and 0000 are connected. 0000 and 0011 are not). In this assignment, the network topology should be a hypercube. To make it simple, you can assume there are 8 nodes.

Since each node can only communicate with its neighbors, you need to design and implement a request routing mechanism so each node can access any topic on any nodes.

The Peer node performs **at least** the following tasks:

* Listens for connections from a port which can be specified either by:
  + Configuration file
  + Passed Arguments
* Hosts topics.
* Forward requests & replies.
* As a server, it could support all the server’s APIs implemented in PA1.
* When its client wants to create/subscribe/… to an existing topic, it should compute the location of that topic using your hash function. Then it connects to the target peer node and performs the action as described in PA1.
* **Peer Node output**: The node maintains a log of the events that happened. E.g, which node it connects to and which nodes it is connected to. What message it sent/received from which node and which topic. What API it has called. All events should have a timestamp.

### Additional Requirements

* All nodes should be able to handle multiple requests simultaneously. You **may** achieve this using threads. You **may** also use other means to achieve this goal.
* The peer node must be able to host topics while it is simultaneously interacting with other topics.

## Evaluation

1. Deploying 8 peers. They can be set up on the same machine or different machines.
   1. Ensure all APIs are working properly.
   2. Ensure multiple peer nodes can simultaneously publish and subscribe to a topic.
2. Similar to PA2, you need to benchmark the latency and throughput of each API.
   1. Deploy 8 peers. Benchmark each API on each node using randomly generated workload.
   2. Graph your results
3. Design & conduct experiments to answer questions about your hash function below.
   1. Time complexity and average time cost at runtime.
   2. Whether it can evenly distribute topics among all nodes.
4. Design & conduct experiments to answer questions about your request forwarding mechanism.
   1. Prove it can work properly. Each node should be able to access topics on all nodes.
   2. Average response time.
   3. Max throughput.

## Extra credits

Up to 10 points. Describe what you are curious about DHT, conduct experiments to solve/verify your questions on your own.

## Submission Information

When you have finished implementing the complete assignment as described above, you should submit your solution to the Canvas. Each program must work correctly and be well documented. You shall submit:

1. **Source Code:** You must hand in all your source code, including with in-line documentation.
2. **Makefile/Ant/requirements**: You must use Makefile or Ant to automate your programming assignment compilation or a requirements file to download any dependencies. If using external libraries (mainly for compiled languages) put them in a folder marked external. **Note:** The external library cannot do the heavy lifting of the assignment tasks for you. If in doubt reach out to the TA.
3. **Deployment scripts**: You must provide a deployment script for the different nodes.
4. **Readme**: A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step.
5. **Compiles Correctly**: Your code must be compiled in a Linux environment.
6. **Report:** Write a report with your findings along with the graphs.
7. **Design Doc**: You must write about how your program was designed, what tradeoffs you made, etc. Also describe possible improvements and extensions to your program (and sketch how they might be made). Separate from report.
8. Please structure your assignment root folder as follows:
   1. Code: for your source code and make files and deployment scripts. Your file structure in here is up to you but please make sure it is clean.
   2. Docs: for all written documents, report, readme, design doc etc
   3. Out: for all output files from your nodes and Indexing server node, named indicating which specific evaluation.
   4. Misc: for other files not mentioned specifically.
9. **Screen shoots:** Submit the screen shoots for your test scenarios.
10. Please put all of the above into **one** .zip file and upload it to **Canvas** The name of .zip should follow this **format**: “Section\_LastName\_FirstName\_PA2.zip”

Grading policy:

* Working code: 50%. You need to demonstrate the following:
  1. Client APIs work with local and remote peer nodes.
  2. Peer nodes work with one another.
* Good testing: 20%.
* Documentation 15% including README and Make scripts and manual pages for the APIs
* Discussion: 15%. You should graph your benchmark results in the report. You should also discuss the design of your code. E.g. Do you think the APIs are sufficient? Where are the bottlenecks? You don’t have to answer exactly these questions. We look for any opinions/ideas/discussion that show you are actively thinking about this project is fine.

Submission checklist:

* Your source codes
* Makefile or equivalent specified above
* Deployment scripts
* Readme
* Your Evaluation output
* The output files of your peer nodes and Indexing server node as specified above
* Screen shoots of your testing
* A Design Document