Programming Assignment 2

Instructions

- Due by 11:59pm of Sunday Oct. 18, 2020.
- 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 **Blackboard** 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.

Overview

In this assignment, you will combine the server and client programs previously made into one single node. Each node now can send and receive files. A node can download all the files from another node and send all the files it has to the other node. You will also create a DHT, a Distributed Hash Table, a designated node that provides a list of all the nodes and the files they can provide. So that a node can ping the DHT for who has the file and go to that node to request for it.

Overview of Changes

Here you are changing the server client model made in the first Programming assignment to a very naive P2P architecture. You are merging their capabilities to allow a single node to download and upload data. You are also adding a centralized file-node lookup with the DHT. You are allowed and expected to make changes to your expected server client architecture implemented in the first programming assignment.

Detailed Description of Assignment

The node hosts files in its directory, these files will be requested by another node and the node sends it. A node listens for a connection on a specific port.

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

- Listens for connections from a port which can be specified either by:
    - Configuration file
    - Passed Arguments
- On starting the node, it connects to the DHT and uploads its file list to it.
- Hosts files in its directory
- When another node connects to it, it must send a list of files it hosts
    - It maintains a list of files in its directory (it can do either of the following)
        - It can update this periodically

        - Update its file list on connection
        - Any other method to achieve this
    - If there is a change in its file list it subsequently updates the DHT
- On getting a request for a file it sends the MD5 hash of the file and then the file to the node.
- When the node needs a new file, it pings the DHT, gets the node associated with the file needed. It can naively get the list of files from the DHT and go to the node associated with a specific file.
- After downloading the file, the DHT needs to be updated.
- After the node gets an exit signal it must notify the DHT that it is inactive.
    - Either it deletes its file entries or the DHT keeps track of active and inactive nodes and ignores files hosted by inactive nodes.
- **Node output**: The node maintains a log of the nodes that it connects to and connected to it, the files it sends to/gets from which node, the amount of data sent and the time it took and possible errors faced.

The DHT performs **at least** the following tasks:

- It maintains the list of nodes and the files they serve
- Nodes should be able to send its current file list to the DHT
    - It must also be able to do the following operations
        - Activate node
        - Deactivate node
        - Update includes the following operations
            - Add file
            - Remove file
        - **Note:** Activate and deactivate are just flags on the DHT which help it infer which nodes are online and offline, it does not control any aspect of the operation of a node.
- Nodes availability must be inferred either:
    - explicitly send active or inactive status to the DHT on start and end
    - Implicitly deduce if a node is active or inactive by pinging them at regular intervals and updating if the node is active or inactive.
- When a node connects to it, it sends it either a file to node mapping or the entire list of files or for a specific file which the node can use to deduce which node to ping for a file
- Currently, the DHT can be either a separate program or process or thread running on one of the nodes,
    - All nodes know which node is the DHT
- **DHT output**: Log all operations done to the DHT. It must track the nodes that connect to it, and what operation they do, an operation is either to get the file list or to do one of the update operations mentioned above.

Additional Requirements

- The Node must be able to connect to multiple Nodes simultaneously. You **may** achieve this using threads. You **may** also use other means to achieve this goal.

- Nodes need not always be connected to other nodes; a connection needs to happen only when it needs to upload or download a file
- The Node must be able to download files while it is simultaneously uploading files to other nodes.
- The folder being watched by the different nodes need to be different and can be specified either by:
    - Configuration file
    - Passed Arguments

Evaluation

1. Start the nodes. Ensure you can transfer one file properly
2. Scale that to 8 nodes that can simultaneously upload and download files.
3. Find out how many node file transfers can be handled simultaneously and how the transfer speed changes with the number of nodes for a fixed number (set by student) of file downloads per node. (start: 8, end: 64, step: 4)
    a. Graph your results.
    b. What differences do you notice with the previous approach as done in the previous programming assignment?

Submission Information

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

1. Source Code (25 points): You must hand in all your source code, including with in-line documentation.
2. Makefile/Ant/requirements (5 points): 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 (10 points): You must provide a deployment script for the different nodes.
4. Readme (10 points): 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 (10 points): Your code must be compiled in a Linux environment.
6. Output files & Performance Evaluation (25 points): A copy of the output generated on running your programs for each of the evaluations. For (3) in evals, provide the output for each level scaled. Write a report with your findings along with the graph.
7. Design Doc (15 points): 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:
    a. 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.
    b. Docs: for all written documents, report, readme, design doc etc.
    c. Out: for all output files from your nodes and DHT, named indicating which specific evaluation.
    d. Misc: for other files not mentioned specifically.
9. Please put all of the above into **one** .zip file and upload it to the **blackboard**. The name of .zip should follow this **format**: “Section_LastName_FirstName_PA1.zip”

Submission checklist:

- Your source codes
- Makefile or equivalent specified above
- Deployment scripts
- Readme
- Your Evaluation output
- The output files of your nodes and DHT programs as specified above
- A Design Document