Socket Programming Assignment: P2P Application (DRAFT)

Due Date: April 10th , 2022 : 11pm

Instructions:

1. **<u>This lab is to be done in groups of two.</u>** Every single line of code should be your own.
2. Note evaluation of this lab will be via an autograded script. These details will be updated later including sharing some test cases and auto-evaluation script, for now get started.
3. The program should be written in phases, with increasing levels of functionality added in each phase. Each phase requires submission of a separate client-phaseX.cc, which will be auto-evaluated for that phase functionality.
4. You will require some files and folder structure for this lab. The same is available as a zip for download (https://drive.google.com/file/d/1xillzjSsyuUxHgGDPhH2UcxMG7sYx3Bd/view?usp=sharing). Explanation is given below. NOTE: The content within the provided files may not be the same as one used by us during auto-evaluation but the general structure will be the same.
5. Use loopback interfaces for initial testing (i.e. all processes run on the same machine). Once it works fine, you should run the processes on different machines and make it work. In our grading, we may run some processes on the same and some on different machines.

In this project you will implement a P2P network for searching and downloading files. The overview is as follows

- There is a network of clients which are interconnected with each other based on a specified topology.
- A client (which you will code) will be provided with the following information as arguments. Note the same client code will be run multiple times with different arguments to mimic different clients.
    - A directory path: this is specific to this client. The client is the owner of all files present in that directory. Another client may be searching for these files to download.
    - A path to a configuration file, which will include the following. See sample file contents below.
        - Client ID and port on which it is ready to listen for incoming connections
        - A list of immediate neighbors and the ports they are listening on. The client is supposed to have direct connections with these neighbors for search purposes.
        - A list of files which the client should search for in the network. These files the client does not possess originally, hence it wants to search and finally

download them. The downloaded files should be stored in the above mentioned directory path under a folder “Downloaded”. These files should NOT be made available to others who are searching ,since this client is not the owner.

- Each of the n clients upon initialization should process its arguments; setup connections with its immediate neighbors and print relevant output (see details below).
- After this, each client should search for certain files as specified in the configuration file. It can search only upto a specified depth d. For example d==2 means, it will search clients upto 2 hops away from it.
- If a file is found at another client, it should setup a separate tcp connection (if not already present) with that client and receive the file. After reception, this connection must be closed. Note File search should happen only via the initial connections specified.

Sample configuration file (e.g. client1-config.txt)

```
1 7000 3765
2
3 5000 4 6000
2
bar.pdf
file.cpp
```

The first line is the id of the client (1) and port (7000) it has to listen on for incoming connections and a unique-id private to it (3765). The second line is the number of immediate neighbors (2). The third line is a list of neighbors, id followed by the port the client is listening on. The fourth line is the number of files this client should search for (2 in this case). The following lines have names of the files it should search.

Directory path (e.g. files/client1)

```
foo.txt          random.jpg      Downloaded/
```

Usage: (executable argument1-config-file argument2-directory-path)

./client-phaseX client1-config.txt files/client1/

(client1 is owner of files foo.txt and random.jpg; and has to search for bar.pdf and file.cpp. And whatever files client-1 downloads as part of its search should be saved in the Downloaded folder.)

Phase 1

Submit: client-phase1.cpp

In this phase, the client should just process the input arguments, establish connections with its immediate neighbors and output the following.

- It should print the name of all files in its directory, one file per line. These are the files owned by this client.
- It should set up connections with its neighbors, and print the unique ID associated with the immediate neighbors. “Connected to <neighbor-ID> with unique-ID <unique ID of that neighbor> on port X” (without quotes and <>).

Example sample output: (assuming 3 and 4 have unique ids 4526 and 7291)
foo.txt
random.jpg
Connected to 3 with unique-ID 4526 on port 5000
Connected to 4 with unique-ID 7291 on port 6000

(Note: if we run 4 clients, each client should generate relevant output based on the provided configuration)

Phase 2

Submit client-phase2.cpp

In this phase, you should implement file searching to a depth of just 1, i.e. check if file is present among immediate neighbors. Each client, for each file it is supposed to search for, it must ask its immediate neighbors and find out if the neighbor is the owner. In case there are multiple owners, choose the one with the smallest unique ID. Now for each file (in ascending order of file name), the client must print “Found <filename> at <client-unique-ID> with MD5 <hash> at depth <depth>”. In case no owner is found, print 0 in <client-unique-ID> and <depth>. For this phase, just set <hash> to be 0. The depth here is 1.

Example sample output: (assuming 3 has bar.pdf)
Found bar.pdf at 4526 with MD5 0 at depth 1
Found file.cpp at 0 with MD5 0 at depth 0

(Note: if we run 4 clients, each client should generate relevant output based on the provided configuration)

Phase 3

Submit client-phase3.cpp

Very similar to Phase2, except that in this phase, we will actually transfer the file if the file is found, else nothing happens. The file transfer will happen over the connection that has already been set up in phase 1. The file which is received should show up in the "Downloaded" directory of the receiver. You must then calculate the MD5 hash of the received file and report it in the <hash> field defined in phase 2.

Example sample output: (assuming 3 has bar.pdf)
Found bar.pdf at 4526 with MD5 d076ccf1bd9eedc74b461e83d08a7df3 at depth 1
Found file.cpp at 0 with 0 at depth 0

(Note: if we run 4 clients, each client should generate relevant output based on the provided configuration)

Phase 4

Submit client-phase4.cpp

This phase is similar to Phase2 except we will increase the search depth to 2. How to implement depth of 2 is up to you. No need to transfer the file yet. The depth should be reported accordingly. You should go to depth 2 only if the file is not found at depth 1. For a given depth, the tie-breaking rule is the same as phase 2. Note this phase does not involve file transfer, only search. Hence no need for MD5 sum value.

Example sample output: (assuming 3 has bar.pdf, 2 has file.cpp)
Found bar.pdf at 4526 with MD5 0 at depth 1
Found file.cpp at 1294 with MD5 0 at depth 2

(Note: if we run 4 clients, each client should generate relevant output based on the provided configuration)

Phase 5

Submit client-phase5.cpp

In this phase, we will transfer the file at depth 2 also much like Phase3. Again details of the implementation is up to you, except that if the file is not at your immediate neighbor, you should use a separate connection to get the file from the node. If a new connection is created for transfer, it must be closed after the transfer is complete and must not be used for searching files in future. All searching must strictly happen on the initial connections between immediate neighbors only. Much like before all downloaded files should be listed in the "Downloaded" folder.

Example sample output: (assuming 3 has bar.pdf, 2 has file.cpp)
Found bar.pdf at 4526 with MD5 d076ccf1bd9eedc74b461e83d08a7df3 at depth 1
Found file.cpp at 1294 with MD5 ab0126af65877bbb17afa6f81eb80f76 at depth 2

(Note: if we run 4 clients, each client should generate relevant output based on the provided configuration)