VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER NETWORK - CO3093

Assignment 1

Develop A Network Application

Supervisor: Nguyen Manh Thin
Class: CC04

HO CHI MINH CITY, NOVEMBER 2023

Contents

1 Member list & Workload

No.	Fullname	Student ID	Tasks	Percentage
1	Ho Cong Gia Bao	2152414	Define and describe functions	24%
2	Le Duc Hoang Nam	2111795	Application design	36%
3	Nguyen Xuan Thanh	2152285	Code and interpret result	40%

2 Application functions and protocols

2.1 Exchanging client - server messages

There are 5 scenarios where client and server need exchanging messages for opposing information and handling requests from the other. They are

- When the client initiates its connection with the server. In other words, the client registers to be tracked by the server for its hostname, address, current status and local repository.
- When the client publishes one of its local file under a name to the server.
- When the client lists the published files in it repository.
- When the client searches for a filename.
- When the client requests a file residing at another host's repository.

The first message is sent only when the client starts, where as others depend on user preference. As a result, we consider using UDP as the only protocol for delivering client - server messages. This prevents us from the overhead of TCP connection and speeds up the communication process. The downside is that UDP message is not guaranteed to be deliverable. As a result, at the client side, we will try to resend several UDP messages in case there is no response for the previous one within a hard-coded timeout. In the other hand, the server will never wait for client's response. To conclude, we implement a generic function for sending and receiving UDP messages and a pair of handlers for each of the above scenarios under client's and server's perspective.

2.2 Transmitting peer to peer file

When the client choose a source node - a peer holding the requested file to download, it needs to know the specific address of that peer, and more importantly, the peer needs to be ready for any incoming connection apart from that of the server. Here, TCP protocol is chosen as we aims at file integrity and don't want the transmitting process to be interrupted by unreliable delivery of UDP, especially with large media file. The fetching function can be described briefly as

1. The requesting client connects to a particular port on the peer's computer. This port is chosen by the server in the initiation phase and must be open to accept any incoming TCP connection.
2. The peer reads the name of the requested file, search in its local repository for a reference to one of its local file and write this to the connecting socket.
3. On the other side, the client's code creates the saving file (if needed) and forwards any input from the socket to it.

2.3 Interacting with user

The GUI of both client and server includes several listing boxes for displaying commands, results and networking status. Also, the GUI needs to cover the required functions in the specification. Here is the demographics of how the GUIs for client and server are like

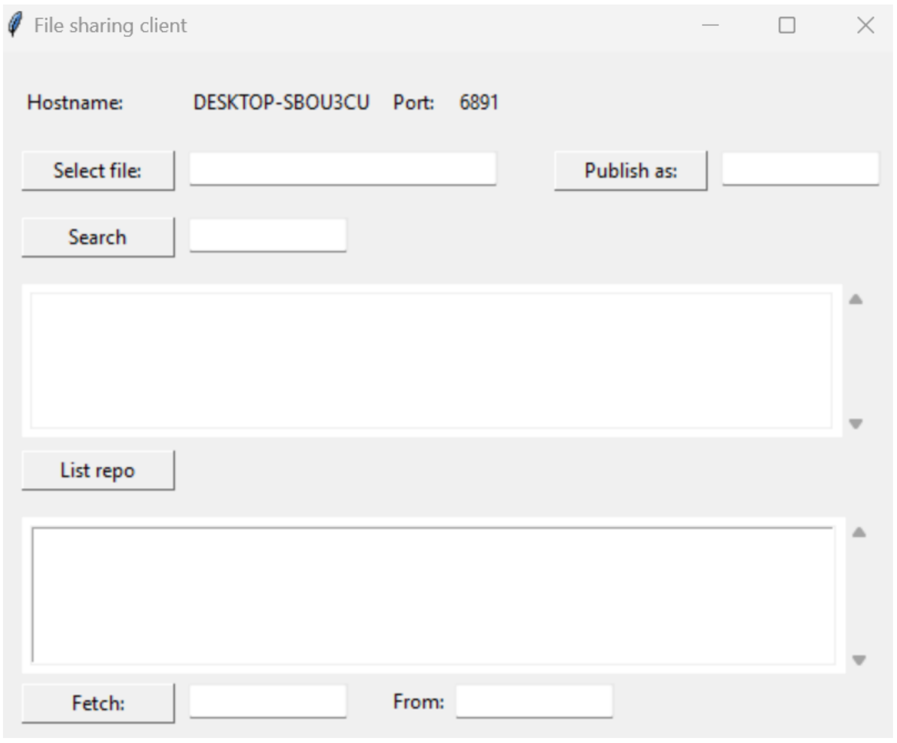

Figure 1: Client GUI

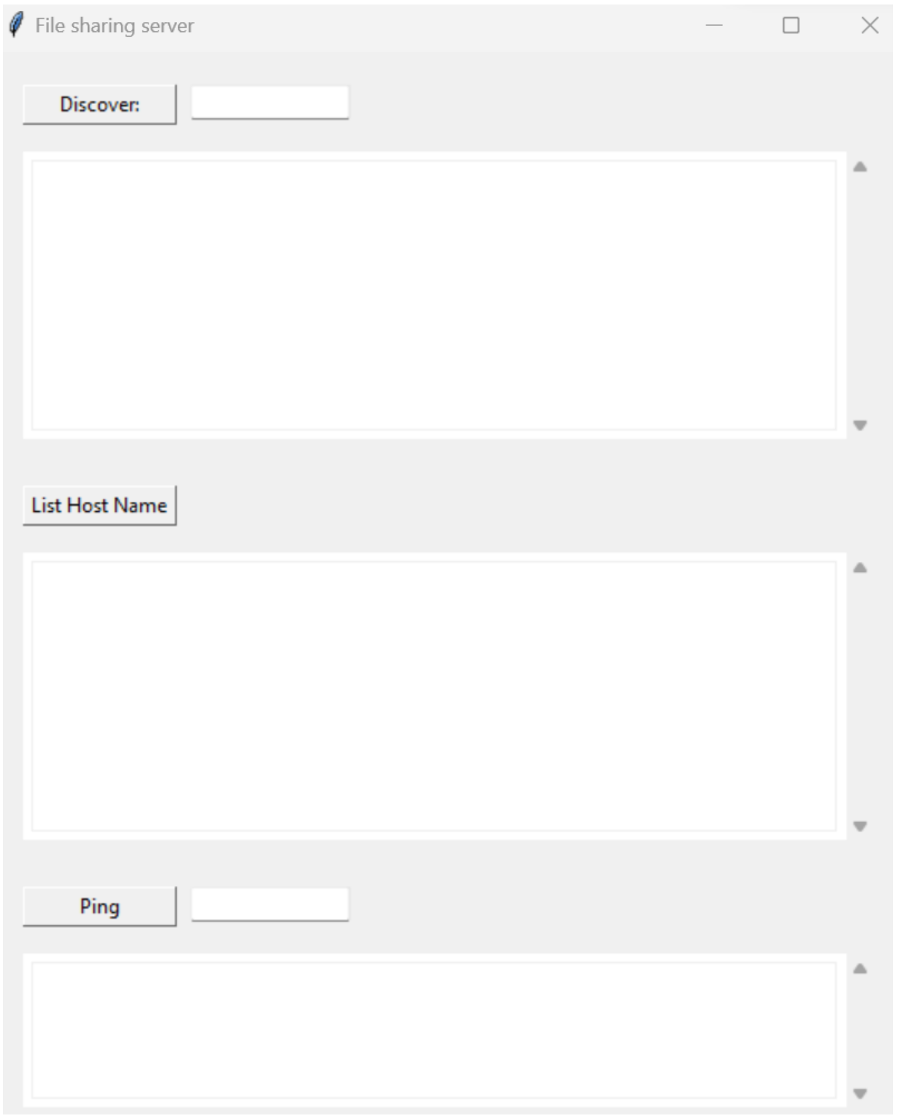

Figure 2: Server GUI

3 Design

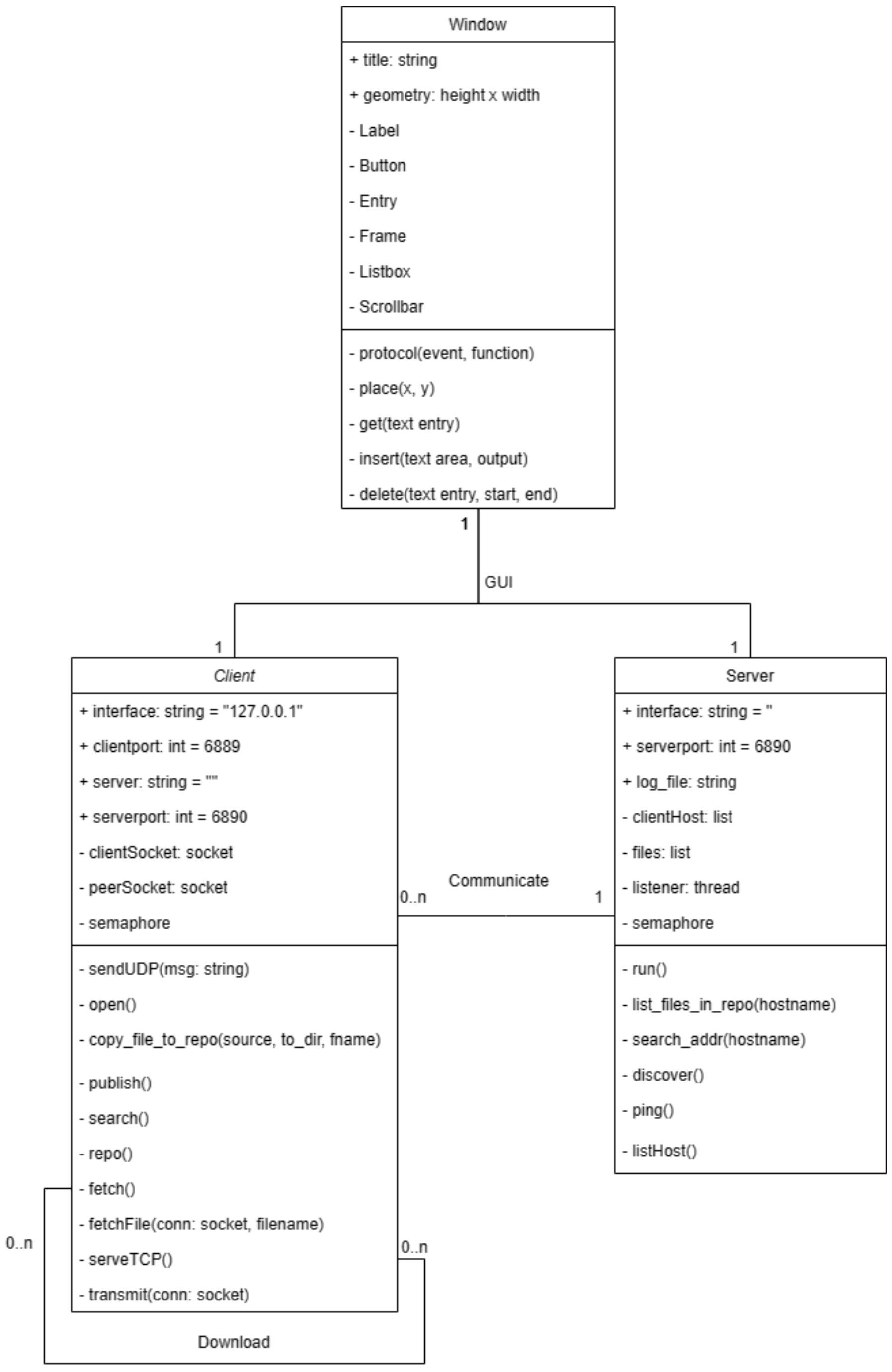

Figure 3: Class diagram

3.1 Classes

- **Window**
  - **Attributes**: tittle(string), gemoetry: height x width
  - **Operations**: `delete(text entry, start, end)`, `protocol(event, function)`, `place(x,y)`, `get(text entry)`, `insert(text area, output)`
  - **Components**: Label, Button, Entry, Frame, Listbox, Scrollbar
- **Client**
  - **Attributes**: interface(string), clientport(int), server(string), serverport(int)
  - **Operations**: `sendUDP(msg: string)`, `open()`, `copy_file_to_repo(source, to_dir, fname)`, `publish()`, `search()`, `repo()`, `fetch()`, `serveTCP()`, `transmit(con: socket)`.
  - **Other Elements**: clientSocket(socket), peerSocket(socket), semaphore.
- **Server**
  - **Attributes**: interface(string), serverport(int), log_file(string), clientHost(list), files(list)
  - **Operations**: `run()`, `list_files_in_repo(hostname)`, `search_addr(hostname)`, `discover()`, `ping()`, `listHost()`.
  - **Other Elements**: listerner(thread), semaphore

3.2 Relationship

- Window is the GUI of Client and Server with a one-to-one association
- Client and Server commumnicate through a many to one association, what means multiple client can communicate with server at a same time.

3.3 Key Points

- The **Client** class represents the client-side, which connect to the server located at ⟨**server**:**serverport**⟩. It also attaches to the **interface** and opens a TCP listening socket at a specified port for peer requests. The **Client** also possesses several functions described above for communicating with the server and handling file operation over TCP.
- The **Server** class represents the server-side, which processes any UDP messages sent to the hard-coded address. **log_file** is the file location where server options are saved under JSON format. The **Server** keeps track of the host names, addresses and corresponding repositories using **files** and **clientHost**. Apart from responding to clients, it can also **ping** or **discover** a host of interest.
- The **Window** class represents the graphical user interface. It consists of an output frame containing a text widget that informs user about the application status and a scroll bar for history tracking. The buttons and input entries depend on whether they are for the clients and servers. However, methods such as binding a function to a button on pressed or getting and modifying entry content are generic.

4 Validation and evaluation

4.1 Testing environment

The testing involves three machines A, B and C in the same subnet. The first one hosts the server and a client process while others run client processes in order to interact with the first host.

4.2 Testing procedure

- **Step 1**
  - **Machine A**: Deploy the server and start a client process on the same interface connecting to the subnet.
  - **Machine B and C**: Start the other client on appropriate interface. Verify that the hostname is identical to that of the machine and a port for serving TCP is displayed.
- **Step 2**
  - **Machine A**: Server list all hostname connecting to it. It should display two distinct rows for two clients. Try pinging the hosts.
  - **Machine B**: Client list files in its repository, which must be empty. Try searching a random filename and fetching it, which results in an empty list box and an error message box.
- **Step 3**
  - **Machine A and B**: Each select a file and publish it with the same name as the other, then list the repository and check for the file. Also, check the ”repo” directory created where the code reside to verify that a copy of that file is there.
  - **Machine C**: Search for the published filename to verify that both hosts have the same filename in their repositories. Fetch them from A’s and B’s clients. Check the ”downloads” directory created where the code reside to verify that two files are downloaded and are named differently.
- **Step 4**
  - **Machine A**: Select the previous file and publish it again with overriding selected. Run the discover command **before** and **after** the publishing to see that the date added field has been changed.
  - **Machine B**: Select a random file but publish it with the same name as previous without overriding. Run the discover command on the server again to verify A’s repository contains two files with different names.
- **Step 5**
  - **Machine A**: Restart the client to see that all the information is retained by the server. Publish a media and a large file.
  - **Machine B and C**: Fetch the two files simultaneously: while B is downloading the large file, let C fetch the media one. Verify that A is multi-threaded and the files are intact.

4.3 Testing proof

The whole testing procedure can be viewed at https://drive.google.com/file/d/1Zw9cMI5qhtkcKyaf0DdeBbaPj6
view?usp=sharing.

5 Manual document

5.1 Deploying the server

To deploy the server, one only needs to run

```
python deploy.py
```

To manually select the IP address and port for the server, fill in the options '–server' and '–serverport'. By default, the server binds to all interface.

5.2 Starting the client

Similar to the server, client application can be started as

```
python start.py
```

The '–interface' option directs the code to use the specified interface, while the '–clientport' option indicates which port the UDP socket should be opened to communicate with the server. If the server is not deployed as default, the user need to specify its address by filling in '–server' and '–serverport' options.