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

NETWORK COMPUTING (CO3094)

ASSIGNMENT

FILE SHARING APPLICATION

Semester 231 - Class CN01

INSTRUCTOR: Bùi Xuân Giang

STUDENTS: Lê Hồng Minh – 2152170
Hoàng Hải Phương – 2152250
Phạm Nguyễn Nam – 2152184
Lê Minh Chánh – 2152445

HO CHI MINH CITY, NOVEMBER 2023

Contents

Introduction

1 Motivation & Objective

1.1 Motivation

In today's dynamic work and learning environments, collaborative tasks are predominantly assigned to teams, necessitating seamless and interactive communication among team members. One of the most time-consuming aspects of collaboration involves the transfer of files between individuals.

Recognizing this challenge, our team has developed a peer-to-peer (P2P) file-sharing system. This system allows users to upload their files to the platform for storage and facilitates quick access to shared files from other users. Users can effortlessly retrieve and share files, fostering efficient and flexible collaboration within the group.

This innovative file-sharing system aims to streamline the exchange of information, reducing the time spent on manual file transfers. By leveraging a P2P architecture, users can contribute, access, and disseminate files with ease, enhancing the overall collaborative experience.

1.2 Objective

- Understanding the functioning of a file-sharing system through the application of a Peer-to-Peer (P2P) architecture.
- Acquiring a comprehensive knowledge of the protocols studied and the specific applications of these protocols.
- Demonstrating the ability to implement a simple file-sharing system.

2 Requirement Analysis

- A centralized server keeps track of which clients are connected and storing what files.
- A client informs the server as to what files are contained in its local
- When a client requires a file that does not belong to its repository, a request is sent to the server. The server identifies some other clients who store the requested file and sends their identities to the requesting client. The client will select an appropriate source node and the file is then directly fetched by the requesting client from the node that has a copy of the file without requiring any server intervention.
- Multiple clients could be downloading different files from a target client at a given point in time. This requires the client code to be multithreaded.
- The client has a simple command-shell interpreter that is used to accept two kinds of commands.
  + **publish *lname fname***: a local file (which is stored in the client's file system at ***lname***) is added to the client's repository as a file named ***fname*** and this information is conveyed to the server.
  + **fetch *fname***: fetch some copy of the target file and add it to the local repository.
- The server has a simple command-shell interpreter
  + **discover *hostname***: discover the list of local files of the host named hostname
  + **ping *hostname***: live check the host named hostname

3 Theoretical foundation

3.1 Peer to peer Architecture (P2P)

- A type of network protocol that allows devices to communicate with each other on a nearly equal basis without the need for a central server. In a P2P network, each device or node acts as both a server and a client, providing and receiving files, with bandwidth and processing power distributed among all members of the network.

- In a P2P network, each node is connected to other nodes in the network, forming a mesh-like structure. This enables data to be transmitted directly between nodes without the need for a central server to control communication within the network. P2P networks can be structured or unstructured.

- Some advantages of P2P:
  - No reliance on a central server.
  - Each computer device is a self-managed user.
  - P2P networks are suitable for home and small business environments.
  - Utilizes minimal network access traffic.

3.2 TCP Protocol

The **TCP(Transmission Control Protocol)** is the standard protocol on the Internet used to ensure the successful exchange of data packets between network devices.

3.3 RPC Protocol

The **RPC (Remote Procedure Call)** protocol which is a set of rules and conventions used to enable communication between different processes or applications in a distributed computing environment. RPC allows one program to execute code (procedures or functions) on another remote machine or process as if it were a local procedure call.

- Communication Protocol: RPC relies on a communication protocol to facilitate the exchange of data and messages between the client and server. Common protocols include HTTP, XML-RPC, and custom binary protocols.

- Message Format: Messages exchanged between the client and server typically have a defined format. This format includes information about the procedure to be executed, parameters, and any additional data needed for the remote invocation.

- Marshalling and Unmarshalling: RPC involves the conversion of data and parameters into a format suitable for transmission (marshalling) and then reconstructing them on the receiving end (unmarshalling).

- Stub Code: Client-side and server-side programs often use stub code. The client-side stub looks like the server-side procedure but packages the procedure's parameters and sends them over the network to the server-side stub.

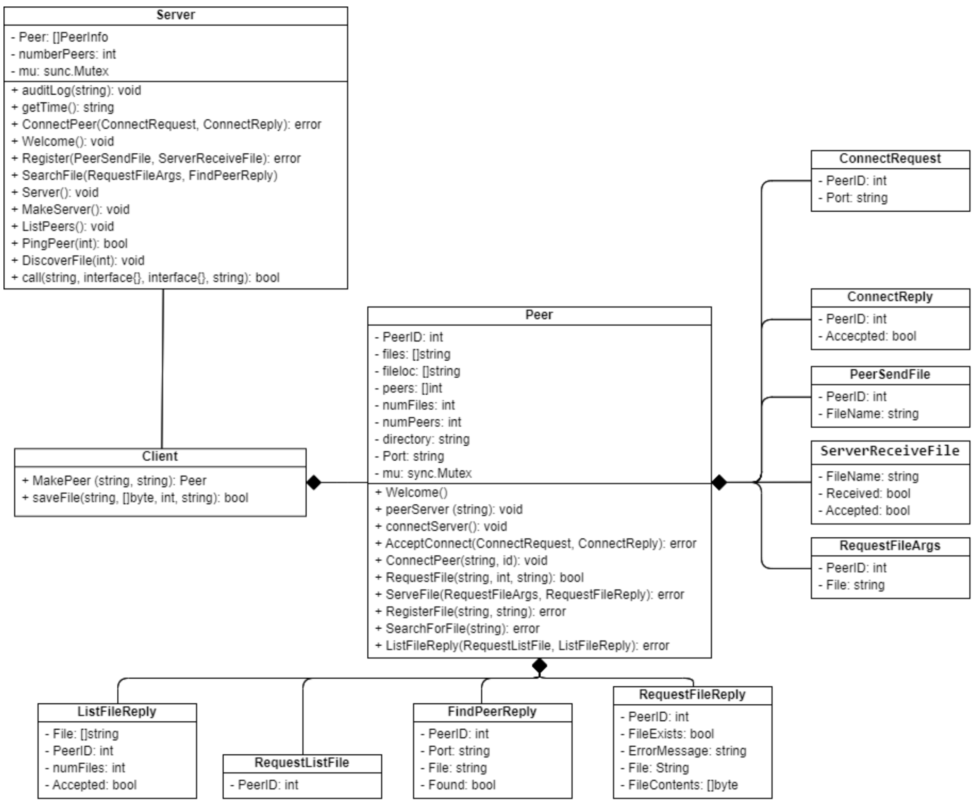
- Binding: Binding is the process of associating a specific procedure call with the code to execute at runtime. It can be done either at compile time or dynamically at runtime.

- Synchronous and Asynchronous RPC: RPC calls can be synchronous, where the client waits for a response before continuing, or asynchronous, where the client can continue with other tasks while waiting for a response.
- Idempotence: RPC systems often aim to be idempotent, meaning that if the same request is made multiple times, the result is the same as if it were made once

3.4 ICMP Protocol

The **ICMP (Internet Control Message Protocol)** is a protocol used for reporting errors, notifying the sender that there is an issue with the data being sent, similar to how a router notifies errors to the source IP address when network issues prevent the distribution of IP packages. ICMP creates and sends messages to the source IP address when there is a network gateway to the Internet that cannot be accessed. Every IP network device has the capability to send, receive, or process ICMP messages.

4 Class Diagram



5 Function

5.1 Client functions

5.1.1 Publish

The client must include the file path on their machine and a new name for the file system (if not specified, it defaults to the name on the client's machine) when using this command. The server will retain this information along with the sender's address.

5.1.2 Fetch

The client uses this command to search for a file that is not yet on their machine or to request the download of a specific file. The server will search the list of published files and return the address of the file owner along with instructions to connect to them through a P2P port for direct file exchange if the file is found. The server will not directly access files on the client's system for security reasons.

5.2 Server function

5.2.1 Discover

The administrator will use this command to request the main server to send information about the files published by a specific hostname.

5.2.2 Ping

The administrator will use this command to request the main server to check the online status of a hostname and return the result.

6 Protocols

6.1 Client functions

6.1.1 Publish

The server uses sockets to listen for information about the file that the client wants to publish, thus utilizing the TCP protocol.

6.1.2 Fetch

The server uses sockets to listen for the filename that the client is searching for, employing the TCP protocol. Additionally, the server supports the establishment of peer-to-peer connections between two machines, utilizing the RPC protocol.

6.2 Server functions

6.2.1 Discover

The admin uses the TCP protocol to command the server to use TCP in retrieving information about files published by the client back to their computers.

6.2.2 Ping

The admin uses the TCP protocol to instruct the server to send ICMP for pinging the desired machine.

7 Manual

In the search bar, type "cmd" to open the Command Prompt.

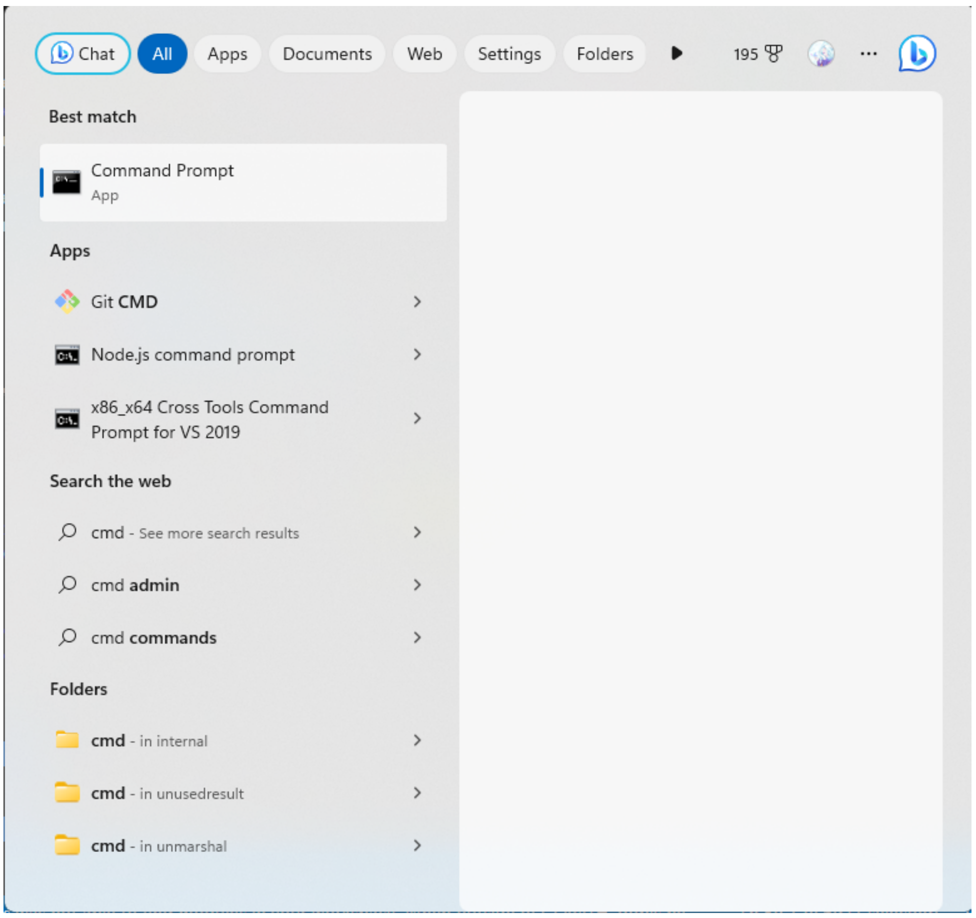
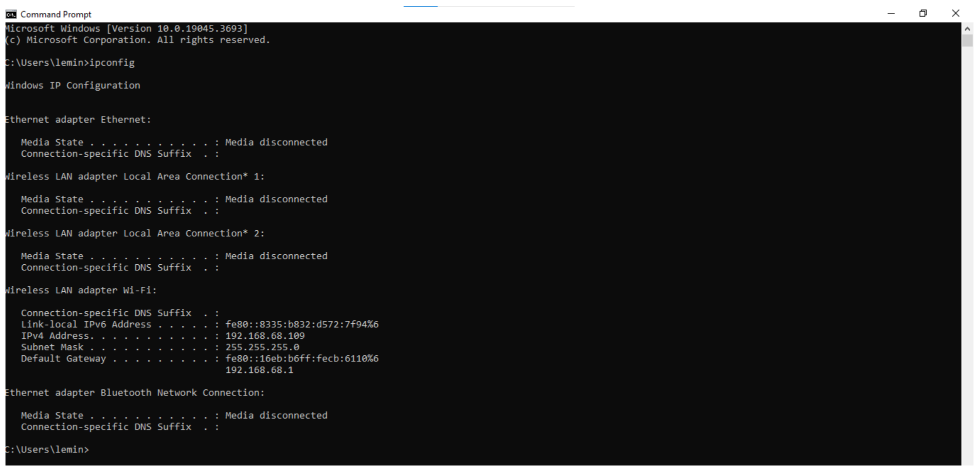


Figure 1: *Command Prompt*

In the Command Prompt, type "ipconfig."

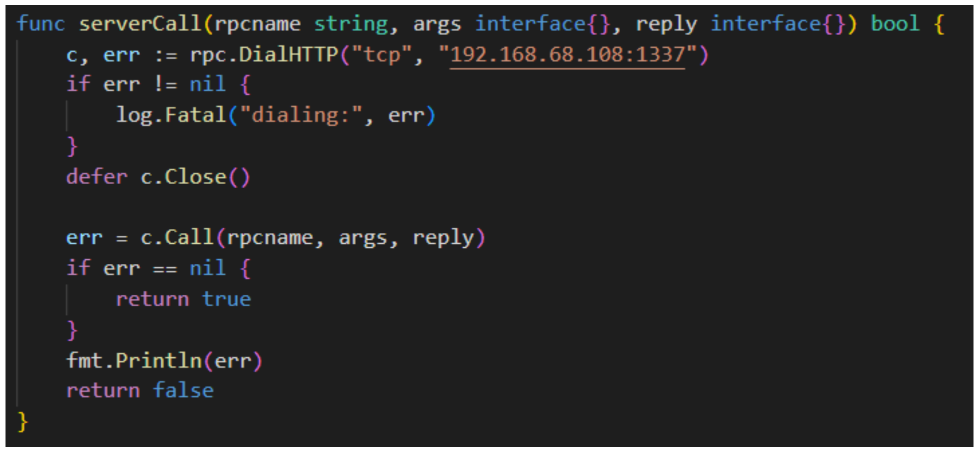


Command Prompt

```
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\lemin>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::8335:b832:d572:7f94%6
   IPv4 Address. . . . . . . . . . . : 192.168.68.109
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::16eb:b6ff:fecb:6110%6
                                       192.168.68.1

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\lemin>
```

Figure 2: *ipconfig*

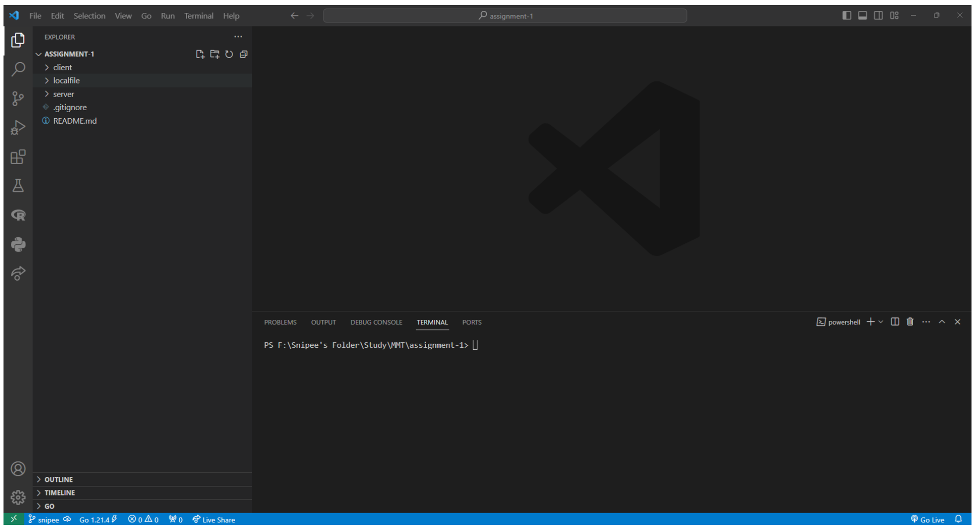
After running ipconfig, check the IPv4 address under the Ethernet adapter (for wired network users) or Wireless LAN (for wireless network users) and copy the IPv4 address. For Ubuntu users utilizing WSL, use the command "ip a" to determine your own IP address. For client users, navigate to the "client" folder. Inside the "client.go" file, modify the serverCall function to use your current IPv4 address. For example, if the current address is "192.168.68.108:1337," change it to "xxx.xxx.xx.xxx:1337."



```go
func serverCall(rpcname string, args interface{}, reply interface{}) bool {
    c, err := rpc.DialHTTP("tcp", "192.168.68.108:1337")
    if err != nil {
        log.Fatal("dialing:", err)
    }
    defer c.Close()

    err = c.Call(rpcname, args, reply)
    if err == nil {
        return true
    }
    fmt.Println(err)
    return false
}
```

Figure 3: *Change IPv4 in client.go*

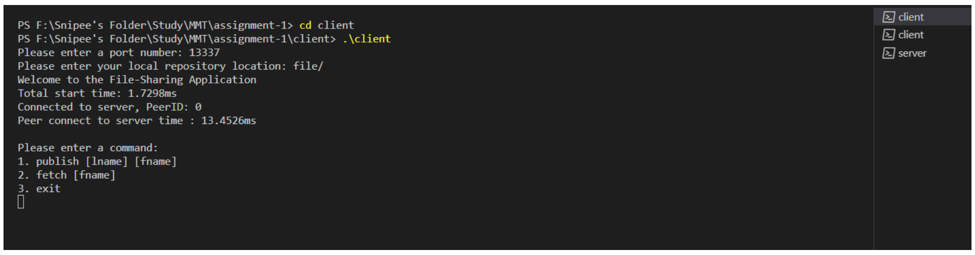
In this case, our team uses Visual Studio Code for ease of operation. Below is the Visual Studio Code interface.In Visual Studio Code, select the "Run" menu, then choose "Terminal" to open the terminal directly at your current folder location.



```
PS F:\Snipee's Folder\Study\MMT\assignment-1>
```

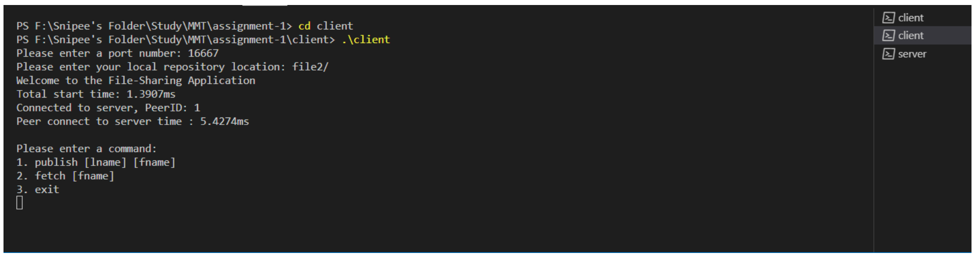
Figure 4: *Visual Studio Code*

From here, create separate terminals, symbolizing one for the server and several for individual clients.
For clients, enter `.\client.exe` to initiate the program.
For the server, enter `.\server.exe` to start the server.
After the client runs `.\client.exe`, the client will enter the port and address to store files. Perform this action in a different terminal. At this point, we now have two clients (in this example, with ports 13337 and 16667, and addresses to save files in the "file/" and "file2/" directories) connected to our server.



```
PS F:\Snipee's Folder\Study\MMT\assignment-1> cd client
PS F:\Snipee's Folder\Study\MMT\assignment-1\client> .\client
Please enter a port number: 13337
Please enter your local repository location: file/
Welcome to the File-Sharing Application
Total start time: 1.7298ms
Connected to server, PeerID: 0
Peer connect to server time : 13.4526ms

Please enter a command:
1. publish [lname] [fname]
2. fetch [fname]
3. exit
```

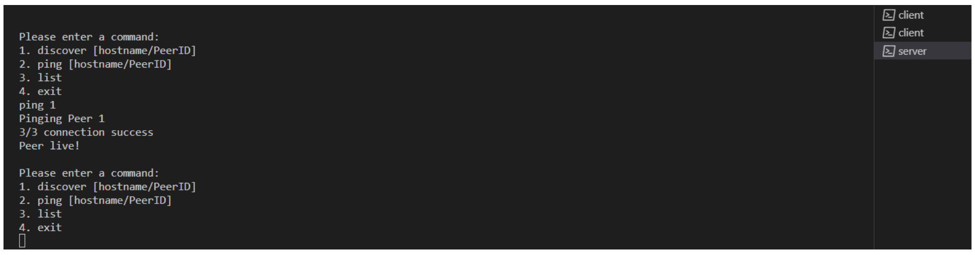
Figure 5: *Run Client 1*



```
PS F:\Snipee's Folder\Study\MMT\assignment-1> cd client
PS F:\Snipee's Folder\Study\MMT\assignment-1\client> .\client
Please enter a port number: 16667
Please enter your local repository location: file2/
Welcome to the File-Sharing Application
Total start time: 1.3907ms
Connected to server, PeerID: 1
Peer connect to server time : 5.4274ms

Please enter a command:
1. publish [lname] [fname]
2. fetch [fname]
3. exit
```

Figure 6: *Run Client 2*

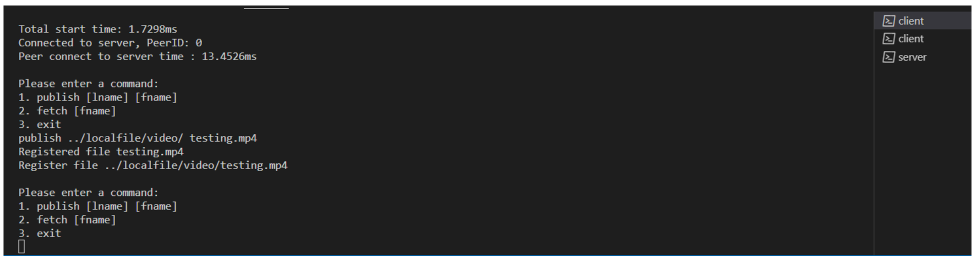
For the server, we can check if the current client with the corresponding PeerID can be reached using the "ping" command. In this case, when we ping with PeerID 1, we observe that Peer 1 is currently reachable.



```
Please enter a command:
1. discover [hostname/PeerID]
2. ping [hostname/PeerID]
3. list
4. exit
ping 1
Pinging Peer 1
3/3 connection success
Peer live!

Please enter a command:
1. discover [hostname/PeerID]
2. ping [hostname/PeerID]
3. list
4. exit
```

Figure 7: *Ping Peer*

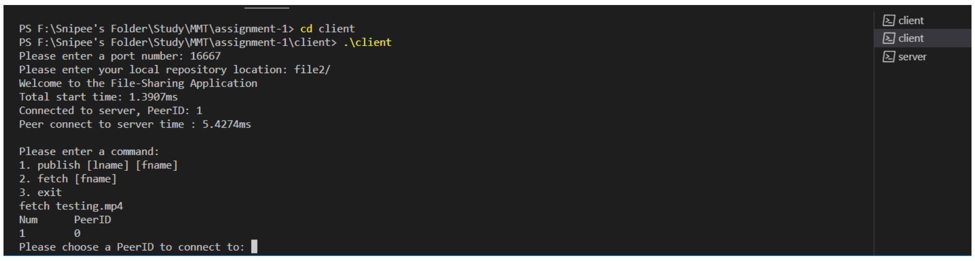
In the scenario where a client wants to upload a file to the server, the client uses the "publish" command with "lname" as the location name (file address to upload) and "fname" as the filename (name of the file to upload).



```
Total start time: 1.7298ms
Connected to server, PeerID: 0
Peer connect to server time : 13.4526ms

Please enter a command:
1. publish [lname] [fname]
2. fetch [fname]
3. exit
publish ../localfile/video/ testing.mp4
Registered file testing.mp4
Register file ../localfile/video/testing.mp4

Please enter a command:
1. publish [lname] [fname]
2. fetch [fname]
3. exit
```

Figure 8: *Publish file*

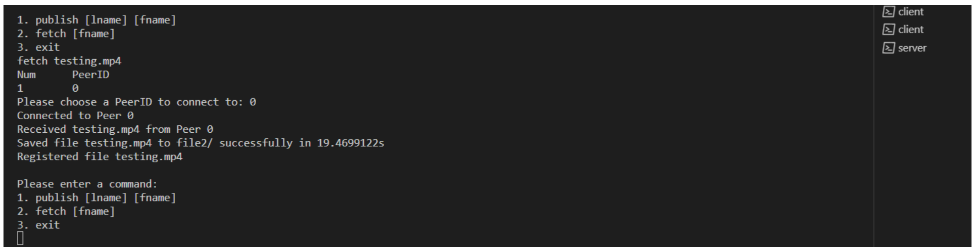
In this example, we want to upload a file named "testing.mp4" with the given address. To do this, we input a command similar to the one above.
After the upload, when Client 2 wants to retrieve the "testing.mp4" file to their machine, the client will use the command "fetch testing.mp4."



```
PS F:\Snipee's Folder\Study\MMT\assignment-1> cd client
PS F:\Snipee's Folder\Study\MMT\assignment-1\client> .\client
Please enter a port number: 16667
Please enter your local repository location: file2/
Welcome to the File-Sharing Application
Total start time: 1.3907ms
Connected to server, PeerID: 1
Peer connect to server time : 5.4274ms

Please enter a command:
1. publish [lname] [fname]
2. fetch [fname]
3. exit
fetch testing.mp4
Num      PeerID
1        0
Please choose a PeerID to connect to:
```

Figure 9: *Fetch file*

After using the "fetch" command, the server will display a list of clients who have uploaded files with names similar to the one on the server. The client then inputs the corresponding PeerID they want to retrieve. The server will initiate the file retrieval process and transfer it to the respective folder for the client. You can immediately see the results in the folder where you want to download it.



```
1. publish [lname] [fname]
2. fetch [fname]
3. exit
fetch testing.mp4
Num      PeerID
1        0
Please choose a PeerID to connect to: 0
Connected to Peer 0
Received testing.mp4 from Peer 0
Saved file testing.mp4 to file2/ successfully in 19.4699122s
Registered file testing.mp4

Please enter a command:
1. publish [lname] [fname]
2. fetch [fname]
3. exit
```

Figure 10: *Fetch file from PeerID 0*

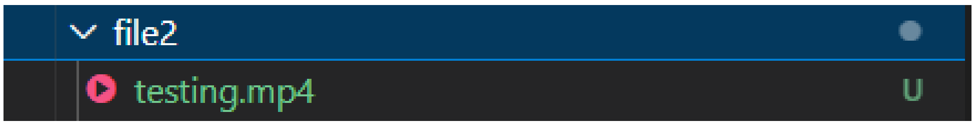
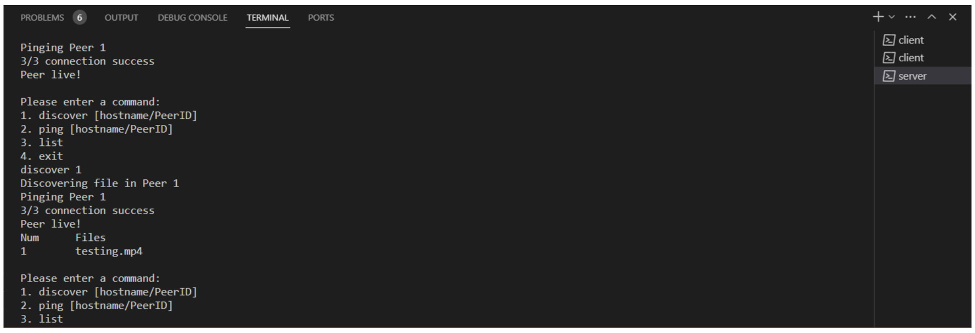


Figure 11: *Folder that contains "testing.mp4" file*

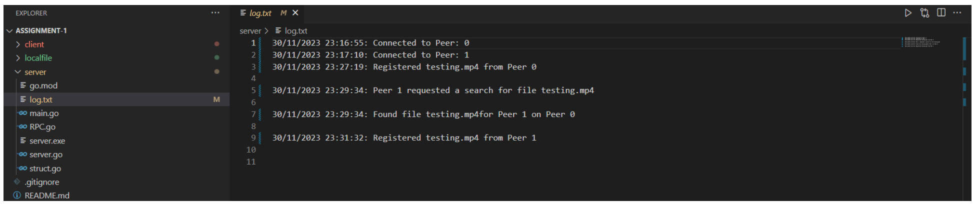
After any client uploads a file, the server can check how many files that client has uploaded with the corresponding "discover PeerNum" command. In this example, we see that client-1 has uploaded the file "testing.mp4" to the server, and the returned result shows 1 file with the name "testing.mp4."
It's worth noting that the process of uploading and downloading files is quite fast (taking around 20 seconds to download a video file with a size of 986MB) and efficient.



```
Pinging Peer 1
3/3 connection success
Peer live!

Please enter a command:
1. discover [hostname/PeerID]
2. ping [hostname/PeerID]
3. list
4. exit
discover 1
Discovering file in Peer 1
Pinging Peer 1
3/3 connection success
Peer live!
Num      Files
1        testing.mp4

Please enter a command:
1. discover [hostname/PeerID]
2. ping [hostname/PeerID]
3. list
```

Figure 12: *Discover*

And on the server side, there is a continuous and updated log to easily monitor clients in the process of uploading and downloading files.



```
30/11/2023 23:16:55: Connected to Peer: 0
30/11/2023 23:17:10: Connected to Peer: 1
30/11/2023 23:27:19: Registered testing.mp4 from Peer 0

30/11/2023 23:29:34: Peer 1 requested a search for file testing.mp4

30/11/2023 23:29:34: Found file testing.mp4for Peer 1 on Peer 0

30/11/2023 23:31:32: Registered testing.mp4 from Peer 1
```

Figure 13: *Server log*

8 System Evaluation

8.1 Advantages

- File can transfer to another person without any addition step.
- When requesting any file, only the name of the requesting file is required, not where the file come from.
- Can handle multiple file types.
- Multithread. Also Golang can easy handled multithread effiecency.
- Can handle very big file and transfer at short time. (Golang)

8.2 Disadvantages

- Hard to use, must required some specific pre-download such as Golang and must follow setup step to get everything read to use.

Closing remarks

Through this semester's assignment, our group has not only accumulated more academic experience in implementing the content given by our instructor, but also practiced communication skills and effective teamwork.

However, during the implementation of the major assignment project, it is inevitable that there will be mistakes. At this point, we hope that our instructor will consider overlooking them. At the same time, due to limited theoretical knowledge and practical experience, the report cannot avoid shortcomings. We hope to receive feedback from our instructor so that we can accumulate more experience and perform better in upcoming major assignment projects.

To conclude this assignment, our group would like to express our sincerest thanks to our instructor, seniors, and fellow students in the National University of Ho Chi Minh City community in general and Ho Chi Minh City University of Technology students in particular for helping us successfully complete this semester's Network Computing assignment.

References

[1] Computer Networking: A Top-down Approach 8th Edition, Pearson - James F. Kurose, Keith W. Ross. (2020).