UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Distributed Systems Project Report

Peer-to-peer file transfer

Group 7

Trần Đắc Minh BI10-113

Đặng Gia Linh BI10-101

Trần Hoàng Minh BI10-119

Đoàn Đình Nam BA9-045

Lê Duy Anh BI9-034

February, 21th 2021

Table of Contents 2

1 Introduction

1.1 What is P2P ?

The concept of P2P network architecture first appeared in 1969. And the initial use of P2P networks in business was in the early 1980s. While P2P systems had previously been used in many application domains, the concept was popularized by file sharing systems such as Napster.

Early P2P networks such as Napster used client software and a central server, while later networks such as BitTorrent, Kazaa, Limewire, BearShare, Morpheus, and Acquisition worked with a central and shared server. Share shared tasks between multiple computers to free up bandwidth.

In its simplest definition, a peer-to-peer network is a simple network where each computer is seen as an access point and as a server for storing and sharing files. In the P2P network, each device is considered a peer. It is like a home network or office network. However, when a P2P network is established over the internet, the size of the files that allow for data sharing is very large.

Once connected to the network, P2P software allows you to search for files on other people's computers. Meanwhile, other users on the network can search for files on your computer, but usually only in a single folder that you have specified to share.

1.2 Why do we use P2P file transfer ?

The main purpose of peer-to-peer networks is to share resources and to help computers and devices work collaboratively, to provide a specific service or perform a specific task. However, the most common use case for peer-to-peer networks is file sharing on the internet. Peer-to-peer networks are ideal for file sharing because they allow computers connected to them to receive files and send files at the same time.

Advantages of P2P

When it comes to the advantages of a P2P file transfer, we can immediately think of the following:

- Easy to install and use.
- No need to manage complex central machines.
- No network administrator is required. Any member can be the administrator for the system.
- All users have the ability to access the sharing of resources between devices.
- Allow users to exchange large files over the internet, usually music and movies.

Disadvantages

Although P2P networks make file sharing easy and convenient, it has also led to many software piracy and illegal music downloads. Therefore, it is best practice to be safe and only download software and music from legitimate websites and copyrighted content.

Furthermore, the distributed nature of P2P networks makes them relatively difficult to control and regulate. Some applications and companies use P2P (peer to peer) networks to engage in illegal and pirated non-sharing activities.

1.3 How P2P file transfer works ?

To connect peers in a peer-to-peer network, three methods exist.
Universal Serial Bus (USB), Wire and Protocols.

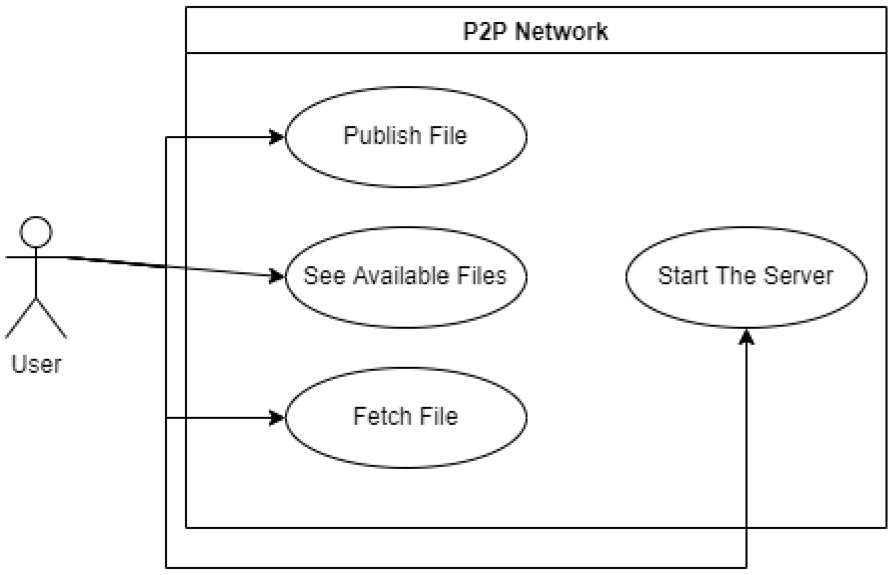
In essence, a peer-to-peer network is maintained by a distributed network of users.

On a P2P network, devices use software applications designed to mediate data sharing. When you want to find and download files, you can send search requests to other devices on the network. And once they've downloaded a file, they can act as the source of that file.

In other words, when downloading a file from node A, node B will act as client. When node A downloads a file from node B, node B will act as the server.

2 Analysis and Design

2.1 Use case

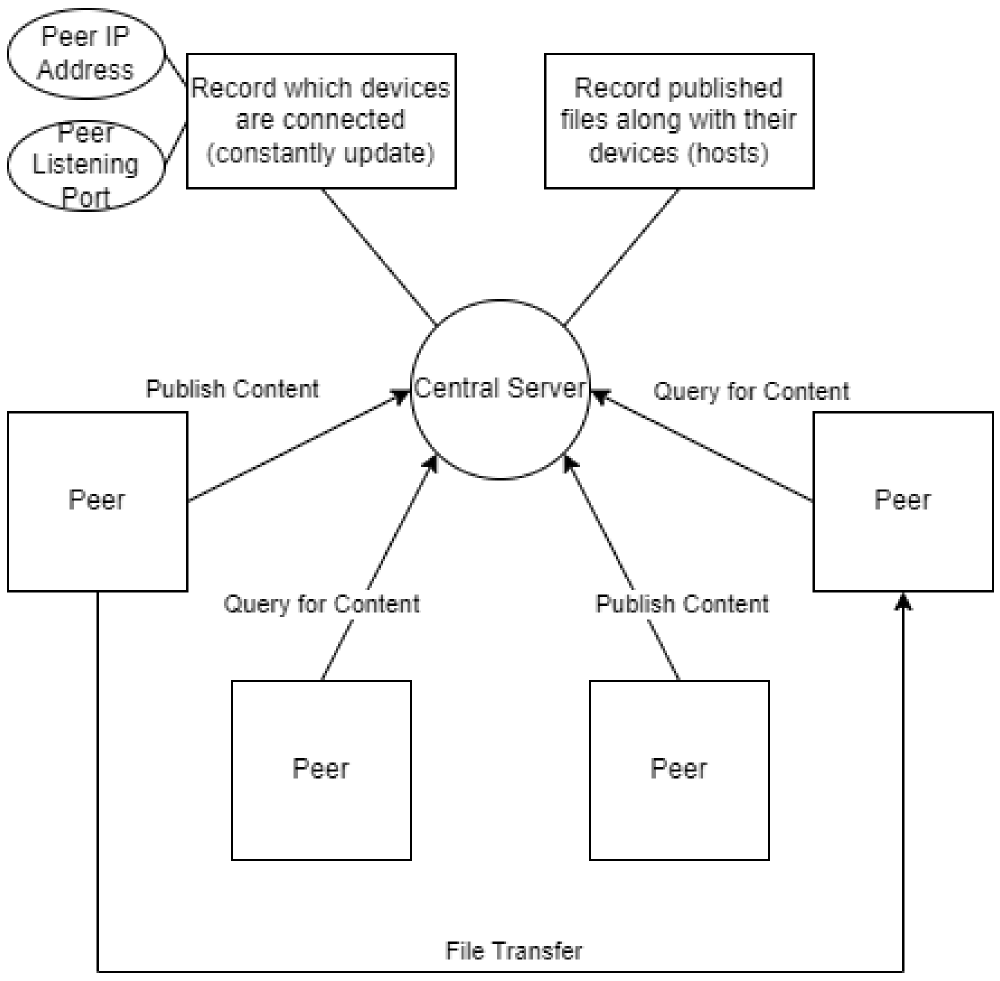


The use case diagram shows the interactions between an User and the P2P network. As shown above, the Users must be able to Publish their files to share it across the network, see available files (published files) and fetch their desired file. Moreover, any member of the network must be able to host the server.

2.2 Architecture - Centralized Directory

- Somewhat similar to client-server architecture in the sense that it maintains a central server to provide directory service.
- All the peers inform this central server of their IP address, listening port and the files they are making available for sharing (published).
- The server queries the peers at regular intervals to make sure if the peers are still connected or not.

2.3 Diagram



2.4 Operations

- Whenever a requesting peer comes in, it sends its query to the server.
- The server has all the information of its peers, so it returns the IP addresses, listening port and connecting status of all the peers and published files to the requesting peer.
- The file (published file) transfer takes place between these two peers. Therefore each peer will act as both a server and a client.

Basically, the central server acts as a receptionist who gives information when asked and the peers are the customers that ask for the other's contact information.

In a regular use:

The peer (client) connects to the server (1) and then either publishes the files (2) or searches for the other peers' information and available files (3). After that, it disconnects from the server (4) and connects to the desired peer using that peer’s information. The file transfer then happens between the two peers.

During this, the server will :

- Record the connected peer information and connecting status for (1)
- Record the published file(s) and its source to the database for (2)
- Get the published files information list with connecting status of the peers, display it and disconnect the client for (3)
- Record the connecting status of the peer for (4)

3 Implementation

- ★ We build our peer nodes and central server using C language.
- ★ The C language has powerful libraries supporting networking. Especially Socket Programming.
- ★ The peer nodes and central server are connected using TCP connections.

3.1 Prerequisites

- Unix based systems environment
- gcc compiler

3.2 Libraries

- ★ Necessary Libraries

#include<stdio.h> <stdlib.h> <string.h> <sys/types.h> <sys/socket.h> <netinet/in.h> <netdb.h> <errno.h> <arpa/inet.h> <signal.h> <unistd.h>
These libraries provide constants, structures and functions for implementing and debugging network sockets.

3.3 Functions

Core functions

socket(AF_INET, SOCK_STREAM, 0);

socket(domain, type, protocol) creates a socket in the specified domain and of the specified type and returns a socket handle (unique identifier of the socket).
AF_INET: Internet domains
SOCK_STREAM: stream-oriented communication type
0: default protocol - TCP

bind(sock, (struct sockaddr *)&server, sockaddr_len);

bind allows a process (in this case the server service process) to specify the local address of the socket. This forms the set local address, local port which will be used when communicating with other processes or remote access.
sock: the socket handle
&server: struct sockaddr_in for IPv4
cast (struct sockaddr_in*) to (struct sockaddr*) => bind only take struct sockaddr*
sockaddr_len: sizeof (struct sockaddr_in)

After binding the socket, to receive a client’s connection, the server need to indicate how many connection requests can be queued, use listen:

listen(sock, MAX_CLIENTS)

sock: the socket handle
MAX_CLIENTS: number of pending connections to queue - defined as 4 in our application

Then it’s time to accept a connection, use accept:

accept(sock, (struct sockaddr *)&client, &sockaddr_len)

accept() blocks the program until receive a connection, then it returns a new socket descriptor that is connected to the requesting client
sock: the socket handle
&client: struct sockaddr_in for IPv4
cast (struct sockaddr_in*) to (struct sockaddr*) => accept only take struct sockaddr*
sockaddr_len: sizeof (struct sockaddr_in)

A client make a connection to the server using connect:

connect(sock, (struct sockaddr *)&remote_server,sizeof(struct sockaddr_in))

sock: the socket handle
&remote_server: struct sockaddr_in for IPv4
cast (struct sockaddr_in*) to (struct sockaddr*) => accept only take struct sockaddr*

connect() and **accept**() will complete a socket's association by fixing the remote half of the address tuple:

An address tuple between a server and client will have the form:
(server_address:server_port, client_address:client_port)

bzero(buffer, MAX_BUFFER): to clear (zero out) the buffer after or before transferring data, alternative can be **memset**(buffer, MAX_BUFFER, 0)
buffer: the array for storing sent or received data
MAX_BUFFER: max size of the buffer, defined as 512 in our application

recv(new, buffer, MAX_BUFFER, 0) and **send**(new, buffer, MAX_BUFFER, 0)

recv and send can be used to transfer data between network and local machine. recv is used to get the data from the network and write it to the buffer while send is used to get the data from the buffer and write it to the network, alternatives: **read**() and **write**()

close(sock): close socket connection

Handle multiple socket connections

For the server, bind(), listen(), and accept() are called on the parent socket, fork() is then used to fork the process for each separate connection with the client (peer), this will enable the server to handle as many clients as possible.

For the client, connect() is called to connect to the server then bind(), listen(), and accept() are called on the parent socket, after that, it forks one background process for listening to incoming requests of other peers.

The background process of the client:

This process uses the select() approach to handle multiple connections.

FD_SET(listen_sock, &master): add the socket descriptor to the set of descriptors

listen_sock: the socket handle (descriptor) of the client, from the function **socket**()

&master: the descriptors set of type struct fd_set

select(FD_SETSIZE, &read_fd, NULL,NULL,NULL): select() will block until there is an incoming connection, to indicates which of the file descriptor is ready for reading

FD_SETSIZE: range of file descriptors to be tested, constant, defined as 64 => max 64 concurrent socket connections

&read_fd: = &master, the descriptors set of type struct fd_set, on output indicates which of the file descriptors are ready to read

FD_ISSET(i,&read_fd): check if i is set in the file descriptors, returns 0 if FALSE, non-zero if TRUE.

i: range from 0 to FD_SETSIZE

&read_fd: file descriptors ready for reading

FD_CLR(i, &master): remove the bit of the file descriptor from the set

i: the file descriptor

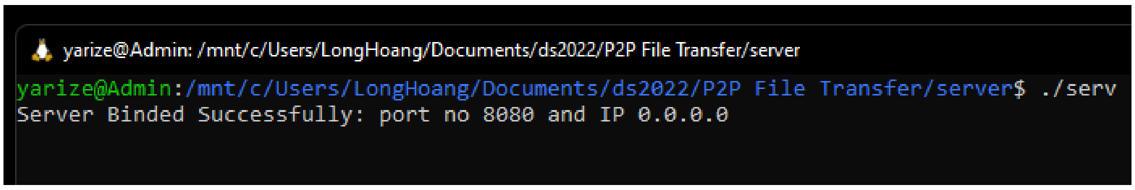
&master: the descriptors set of type struct fd_set

The other peers can connect to this peer using connect() normally like how it would connect to a server.

4 Results

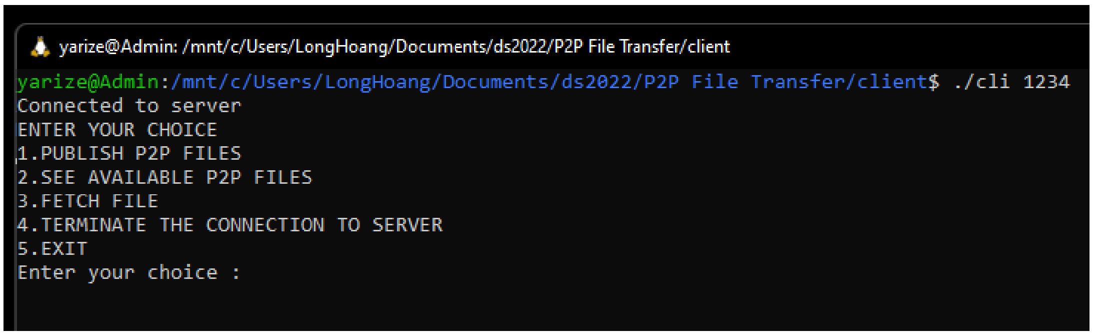
Tested on local host (unfortunately we don't have money for 2 computers)

Bind the server: we have configured the server with IP 0.0.0.0 and Port 8080. IP 0.0.0.0 means "to listen to any connection".



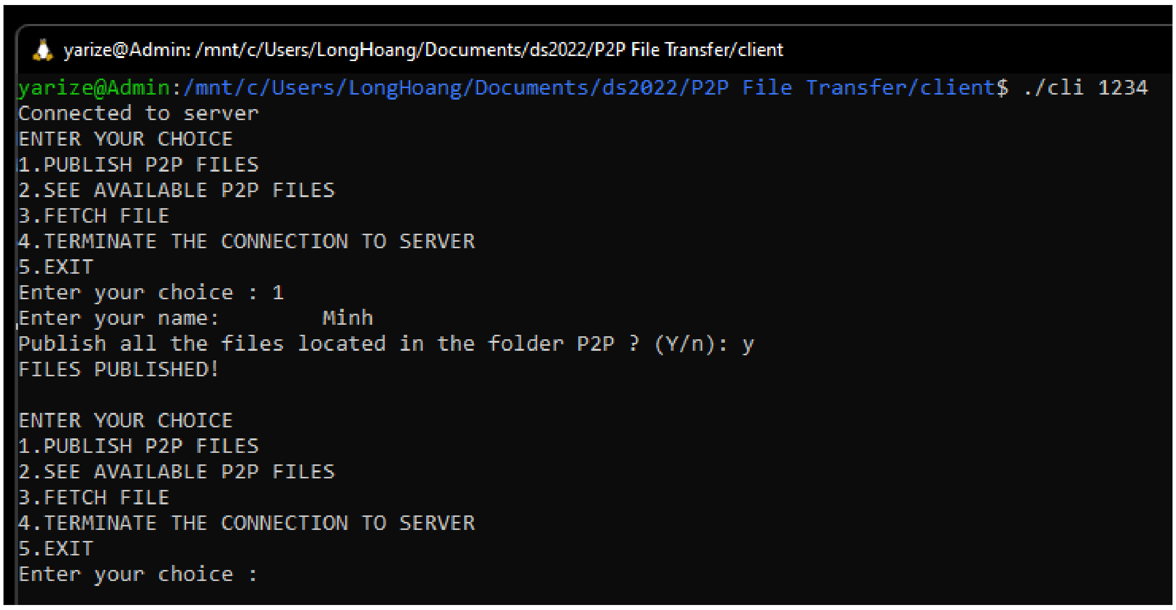
```
yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/server
yarize@Admin:/mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/server$ ./serv
Server Binded Successfully: port no 8080 and IP 0.0.0.0
```

Connect to the server from the client, from here user can interact by input choice of action, the three most important ones are the first three:



```
yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client
yarize@Admin:/mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client$ ./cli 1234
Connected to server
ENTER YOUR CHOICE
1.PUBLISH P2P FILES
2.SEE AVAILABLE P2P FILES
3.FETCH FILE
4.TERMINATE THE CONNECTION TO SERVER
5.EXIT
Enter your choice :
```

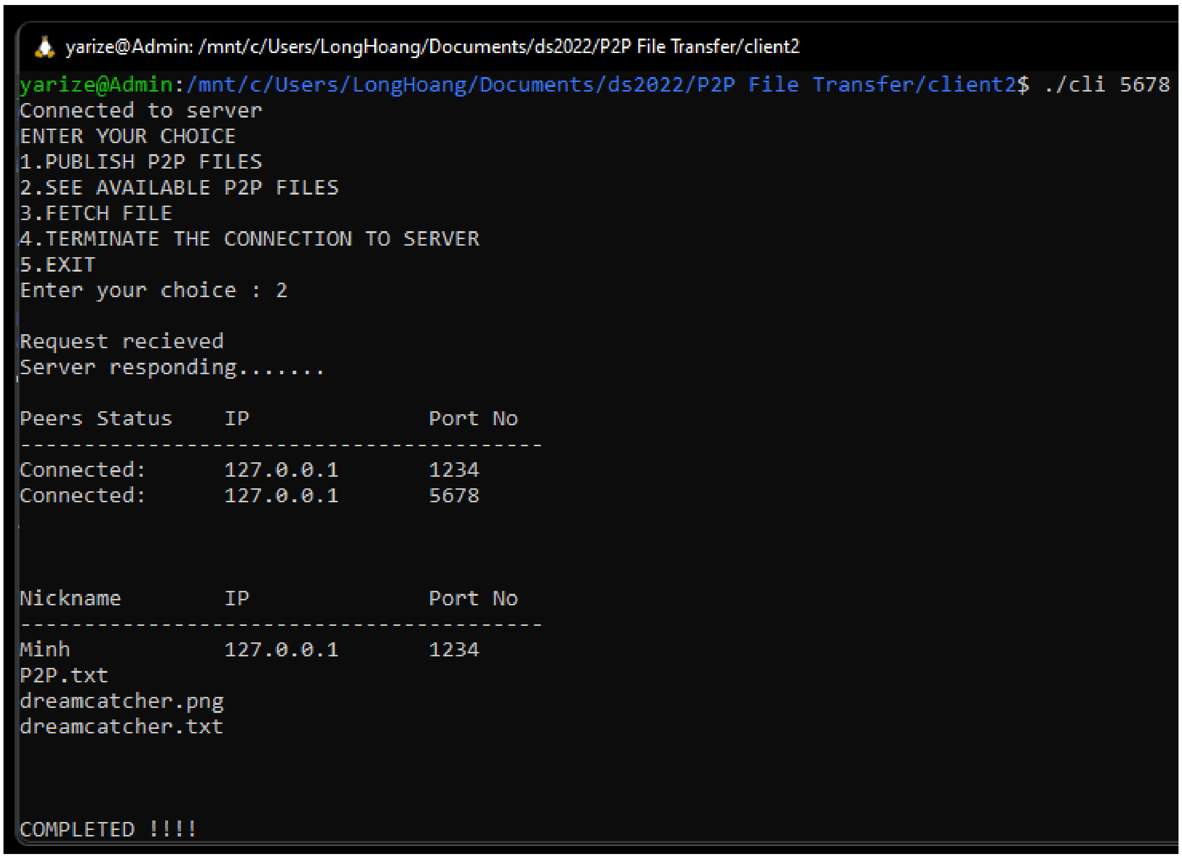
Client (peer) can choose to publish files for sharing, choosing this option will asks the user if he/she wants to publish all the files in the P2P folder, if yes then the operation proceeds :



```
yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client
yarize@Admin:/mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client$ ./cli 1234
Connected to server
ENTER YOUR CHOICE
1.PUBLISH P2P FILES
2.SEE AVAILABLE P2P FILES
3.FETCH FILE
4.TERMINATE THE CONNECTION TO SERVER
5.EXIT
Enter your choice : 1
Enter your name:        Minh
Publish all the files located in the folder P2P ? (Y/n): y
FILES PUBLISHED!

ENTER YOUR CHOICE
1.PUBLISH P2P FILES
2.SEE AVAILABLE P2P FILES
3.FETCH FILE
4.TERMINATE THE CONNECTION TO SERVER
5.EXIT
Enter your choice :
```

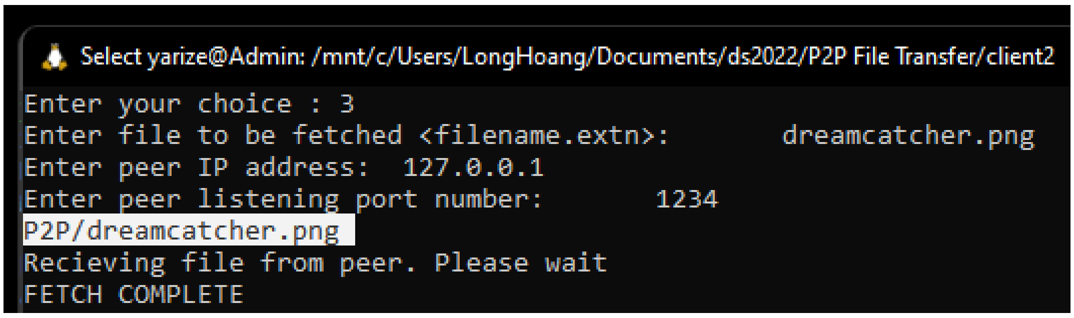
Users don’t have to specify the name before publishing, but everybody would want to exchange files with someone who has a name rather than just an IP address. So specifying a name is advised.

After the files have been published, it will be recorded in the database and other peers can get information about the files through option 2:

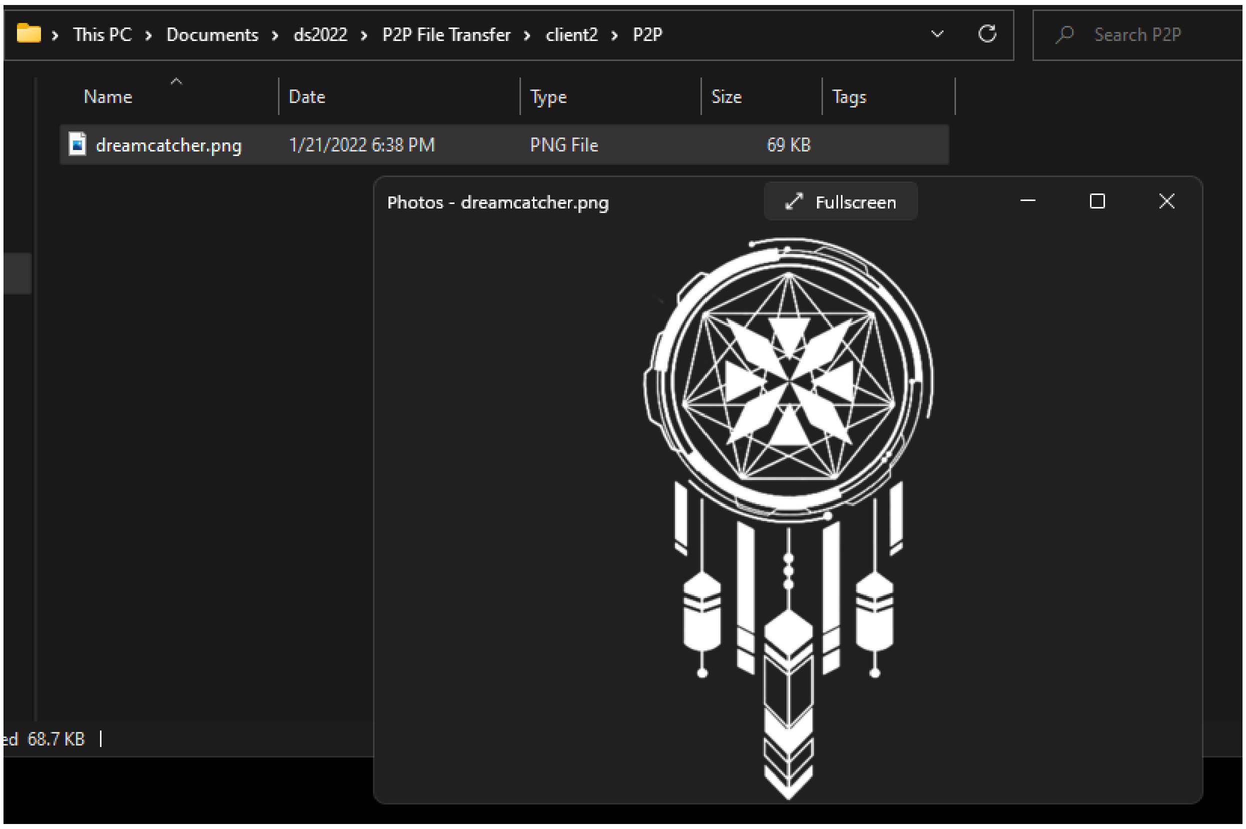

```
yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client2
yarize@Admin:/mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client2$ ./cli 5678
Connected to server
ENTER YOUR CHOICE
1.PUBLISH P2P FILES
2.SEE AVAILABLE P2P FILES
3.FETCH FILE
4.TERMINATE THE CONNECTION TO SERVER
5.EXIT
Enter your choice : 2

Request recieved
Server responding.......

Peers Status    IP              Port No
-----------------------------------------
Connected:      127.0.0.1       1234
Connected:      127.0.0.1       5678



Nickname        IP              Port No
-----------------------------------------
Minh            127.0.0.1       1234
P2P.txt
dreamcatcher.png
dreamcatcher.txt



COMPLETED !!!!
```

Here the server will give the user both information about other peers and the available files for fetching.

User can then choose option 3 to fetch the desired file, this will proceed after user has input the necessary information:

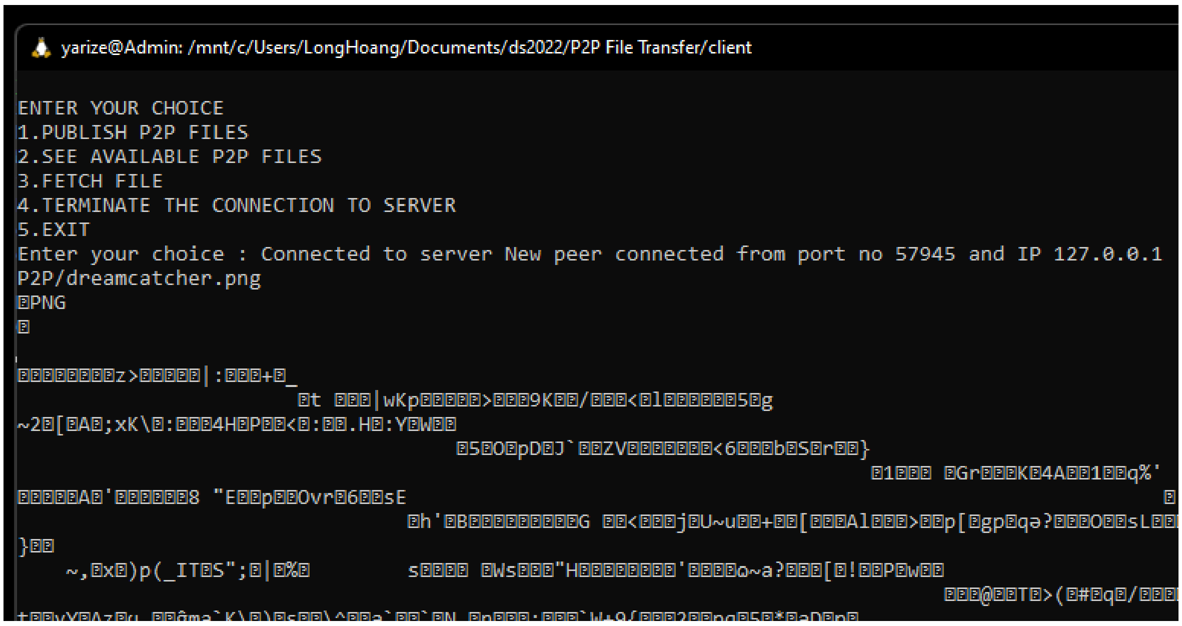

```
Select yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client2
Enter your choice : 3
Enter file to be fetched <filename.extn>:       dreamcatcher.png
Enter peer IP address:  127.0.0.1
Enter peer listening port number:       1234
P2P/dreamcatcher.png
Recieving file from peer. Please wait
FETCH COMPLETE
```

The file fetched can be found in the P2P folder:


This PC › Documents › ds2022 › P2P File Transfer › client2 › P2P
Search P2P
Photos - dreamcatcher.png
Fullscreen
68.7 KB

Name	Date	Type	Size	Tags
dreamcatcher.png	1/21/2022 6:38 PM	PNG File	69 KB	

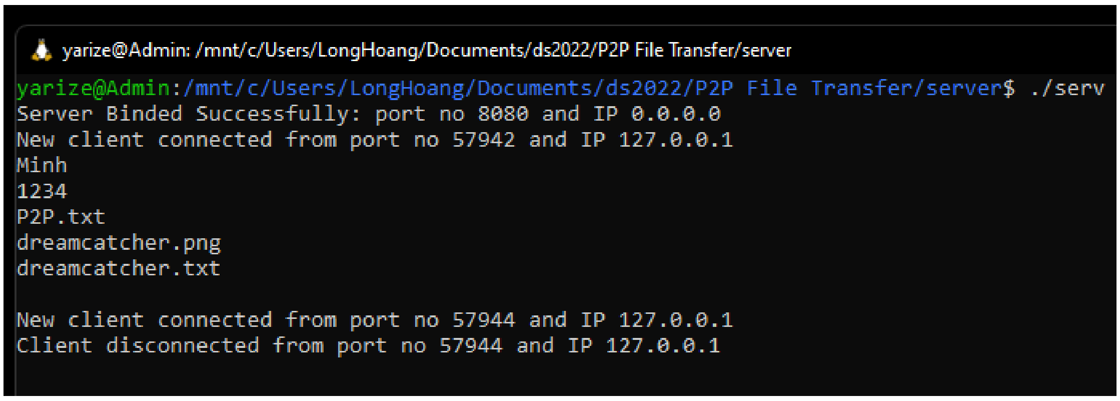
On the host peer side:



```
yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/client

ENTER YOUR CHOICE
1.PUBLISH P2P FILES
2.SEE AVAILABLE P2P FILES
3.FETCH FILE
4.TERMINATE THE CONNECTION TO SERVER
5.EXIT
Enter your choice : Connected to server New peer connected from port no 57945 and IP 127.0.0.1
P2P/dreamcatcher.png
▯PNG
▯

▯▯▯▯▯▯▯▯z>▯▯▯▯▯|:▯▯▯+▯_
                       ▯t ▯▯▯|wKp▯▯▯▯▯>▯▯▯9K▯▯/▯▯▯<▯l▯▯▯▯▯▯▯5▯g
~2▯[▯A▯;xK\▯:▯▯▯4H▯P▯▯<▯:▯▯.H▯:Y▯W▯▯
                                    ▯5▯O▯pD▯J`▯▯ZV▯▯▯▯▯▯▯▯<6▯▯▯b▯S▯r▯▯}
                                                                  ▯1▯▯▯ ▯Gr▯▯▯K▯4A▯▯1▯▯q%'
▯▯▯▯▯A▯'▯▯▯▯▯▯8 "E▯▯p▯▯Ovr▯6▯▯sE                                                             ▯
                                ▯h'▯B▯▯▯▯▯▯▯▯▯▯G ▯▯<▯▯▯j▯U~u▯▯+▯▯[▯▯▯Al▯▯▯>▯▯p[▯gp▯qə?▯▯▯O▯▯sL▯▯
}▯▯
    ~,▯x▯)p(_IT▯S";▯|▯%▯        s▯▯▯▯ ▯Ws▯▯▯"H▯▯▯▯▯▯▯▯▯'▯▯▯▯ɑ~a?▯▯▯[▯!▯▯P▯w▯▯
                                                                              ▯▯▯@▯▯T▯>(▯#▯q▯/▯▯▯
```

All of the print is just for debugging purposes, as shown above, the requesting peer will send the file name to the host peer and this peer will read the file content and send it back.

On the server side:

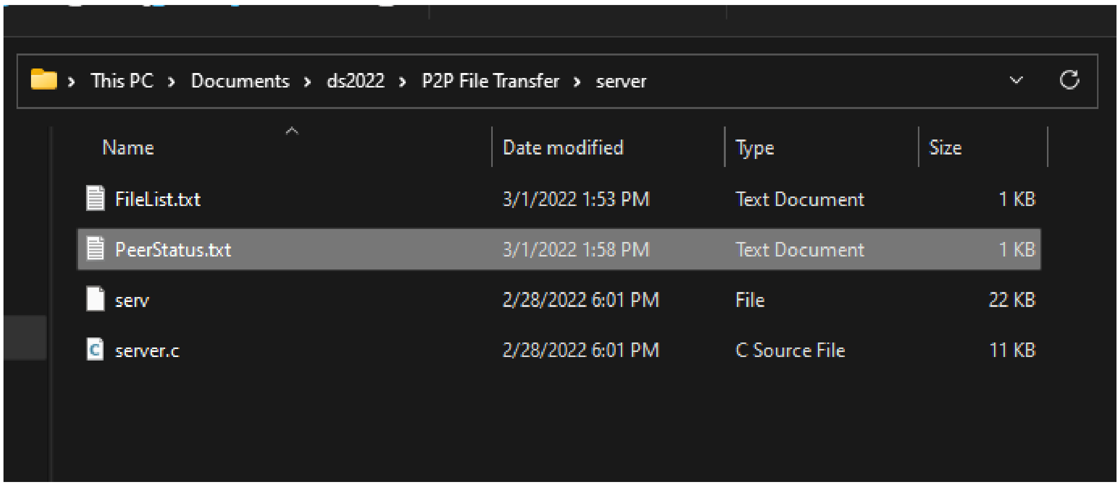


```
yarize@Admin: /mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/server
yarize@Admin:/mnt/c/Users/LongHoang/Documents/ds2022/P2P File Transfer/server$ ./serv
Server Binded Successfully: port no 8080 and IP 0.0.0.0
New client connected from port no 57942 and IP 127.0.0.1
Minh
1234
P2P.txt
dreamcatcher.png
dreamcatcher.txt

New client connected from port no 57944 and IP 127.0.0.1
Client disconnected from port no 57944 and IP 127.0.0.1
```

Again, the prints are just for debugging purposes, everytime a client connects to the server, it will automatically send its information to the server to record into the database. In the picture, it can be seen that the client will send what information when publishing files to the server: name, listening port and files list.

The database:

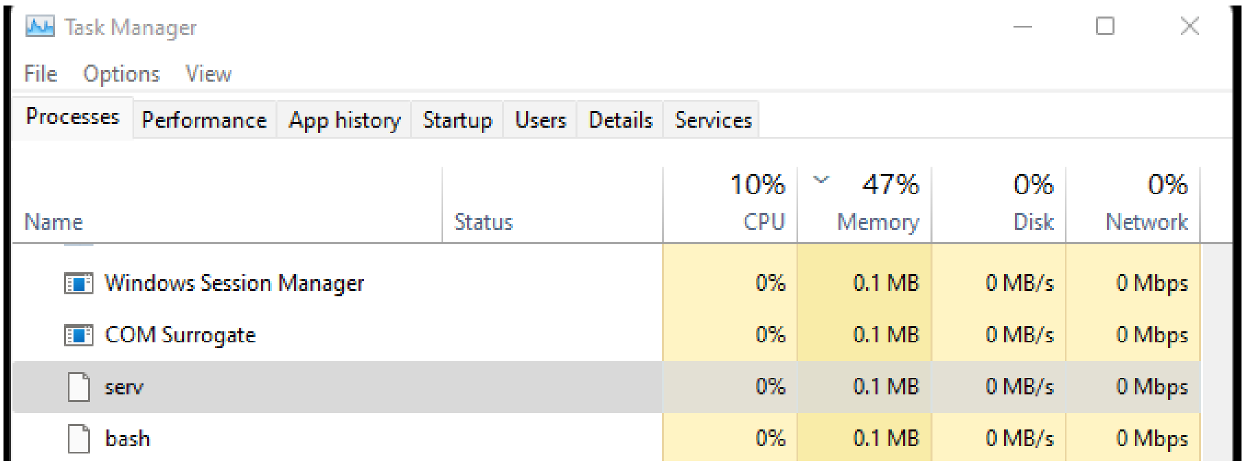
They are just 2 text files: FileList.txt and PeerStatus.txt



Name	Date modified	Type	Size
FileList.txt	3/1/2022 1:53 PM	Text Document	1 KB
PeerStatus.txt	3/1/2022 1:58 PM	Text Document	1 KB
serv	2/28/2022 6:01 PM	File	22 KB
server.c	2/28/2022 6:01 PM	C Source File	11 KB

When a peer chooses option 2, the server will read from these 2 files and send its contents to the client.

Additional note:


Task Manager
File Options View
Processes Performance App history Startup Users Details Services

Name	Status	10% CPU	47% Memory	0% Disk	0% Network
Windows Session Manager		0%	0.1 MB	0 MB/s	0 Mbps
COM Surrogate		0%	0.1 MB	0 MB/s	0 Mbps
serv		0%	0.1 MB	0 MB/s	0 Mbps
bash		0%	0.1 MB	0 MB/s	0 Mbps

The memory (RAM) usage of the server is very small for almost all of the modern computers, and it’s 0.1 MB (+ 0.1 MB for every connection) so users can run it all day all year without any worries.

5 Conclusion and Future work

5.1 Conclusion

● Testing the P2P file transfer on a local machine has given out some successful results.

● The P2P file transfer is easy to use.

● To deploy it to a real network is still a long road to go, especially considering the security aspect.

5.2 Future work

★ Add more error and exception handlers in the code to make the system more stable.

★ Maybe add more options for users to interact with the system other than just query and transfer files.

★ Maybe make a GUI for easier use and better experience.