XI'AN JIAOTONG-LIVERPOOL UNIVERSITY

Name (ID Number)	Jeongyeong Park (2032801)
Programme	BSc Information and Computing Science
Module Title	Introduction to Networking
Module Code	CAN201
Assignment Title	CAN201 Networking Project
Submission Deadline	30th November 2021
Lecturer Responsible	Fei Cheng

1. Abstract

File synchronization is a method that is widely used in file-sharing applications. It updates computer files in two or more locations according to definite rules. In order to provide a file synchronization service between devices, network programming allows devices to interact. This report illustrates the implementation of a synchronizable file sharing application between two hosts by using Python network programming.

2. Introduction

2-1. Project Requirement

The purpose of this project is to develop and implement a Large Efficient Flexible and Trusty Files Sharing program by using Python Socket network programming. The application should be able to transfer files with any forma and folders and single file's size goes up to 500MB. The application should transfer file within short amount of time and files and folders on both machines should be synchronized automatically all the time. Moreover, IP addresses of peers would be set as an argument so it can be used in any PC with different IP address. It should be run properly after it is shut down and file should be transferred without any data lose.

2-2 . Background Information

-P2P architecture

The P2P architecture minimizes the dependence of data centers on dedicated servers. The application communicates directly between periodically connected pairs of hosts known as peers. Peers are user-controlled desktops or laptops that are not owned by service providers and they are mostly found at home, the company, and school.

-TCP(Transmission Control Protocol)

TCP is the Internet's transport-layer which is connection-oriented and reliable transport protocol. transport. TCP—the Internet's transport-layer, connection-oriented, reliable transport protocol.

2-3. Literature Review

Since UDP can transfer data faster than TCP, there are some research analyzing usage of UDP in file sharing application [1]. It is fact that people will be able to share files faster than using TCP. However, in file sharing service, it is really important to guarantee the file will be shared without any damage. If UPD is used, there might be data lose since it is not connection-oriented protocol.

2-4. Project Overview

In this project, P2P architecture is used for file distribution so each peer becomes a server and also become a client at the same time. TCP socket is used for the connection between two

peers to exchange files and accomplish synchronization. Therefore, it guarantees reliable transport.

3. Methodology

3-1. Class and Function

1) Main Class

Function	Description
_argparse()	Parse external parameter which is peer's ip
create_direct()	Create directory if it does not exist
file_info()	Get a dictionary for information of local files, having file name as a key and list [size, hash value] as value
subdirect_file_info(subdir_path)	Get a dictionary for information of local files in subdirectory of main directory, having file name as a key and list [size, hash value] as value This dictionary will be update to file information dictionary as sub directory name for key and dictionary containing files in subdirectory as value
get_md5(file)	Get md5 hash value of the given file
compressing_file(path)	Compress the files in the given path
decompressing_file(path)	Decompress the files in the given path

2) Server Class

Function	Description
server(self)	Call the functions in Server class
connect(self)	Create server socket, bind, and listen to client
accept(self)	Accept connection and call other functions
file_list_sender(self)	Send the information of the local files containing file names, size, and md5 hash value to the connected peer.
file_download(self)	Get the update dictionary containing information of file which is not in local directory but in peer's local directory and get the files.
folder_download(self, folder_name, folder_files)	Get the sub-directory and files inside it which are located in peer's device but not in local device.

3) Client Class

Function	Description
client(self)	Call the functions in Client class
connect_server(self)	Create client socket, send request for connection to server and call the function
peer_files_get(self)	Get the information of the connected peer's local files containing file names, size, and md5 hash value.
detect_file(self, new_dict)	Compare files in the local directory with files from the peer and find the file has to be sent
send_file(self, send_files)	With the dictionary containing files have to be send, send files to the other peer
send_folder(self, folder_name, folder_dict)	Send the sub-directory and files inside it which are located in the local device but not in the peer's device

In this program, there are two threads which are *server thread* and *client thread*.

-Server Thread

As it can be seen in the above table for Server class, *server thread* accepts client's request and sends the dictionary containing information of files in the local device to client. It also receives files data from the client.

-Client Thread

As it can be seen in the above table for Client class, *client thread* requests connection to server and get the dictionary containing information of files in the peer's device and compare it with the files in the local device. After finding the files that have to be sent to the server, it sends the data to the server.

3-2. Proposed Protocols

In this program, two threads which are server and client interact with each other. Once server thread and client thread have started, they both create the TCP socket. After server socket finishes binding and starts to listen, client socket sends request for accepting. After that in order to check they are actually connected to each other, server and client send and receive messages that they are both online. If they check they are both online, server will send the dictionary containing information of files in the local device to client. Once client receives that dictionary, it starts to compare information and creates dictionary for files to be sent. Client will send that file list to server. In this step, even though dictionary containing 0 value, client will send it to server in order to let server know client is still connected. If client finds out there is any file needed to be sent, client will send it to server and finish synchronization.

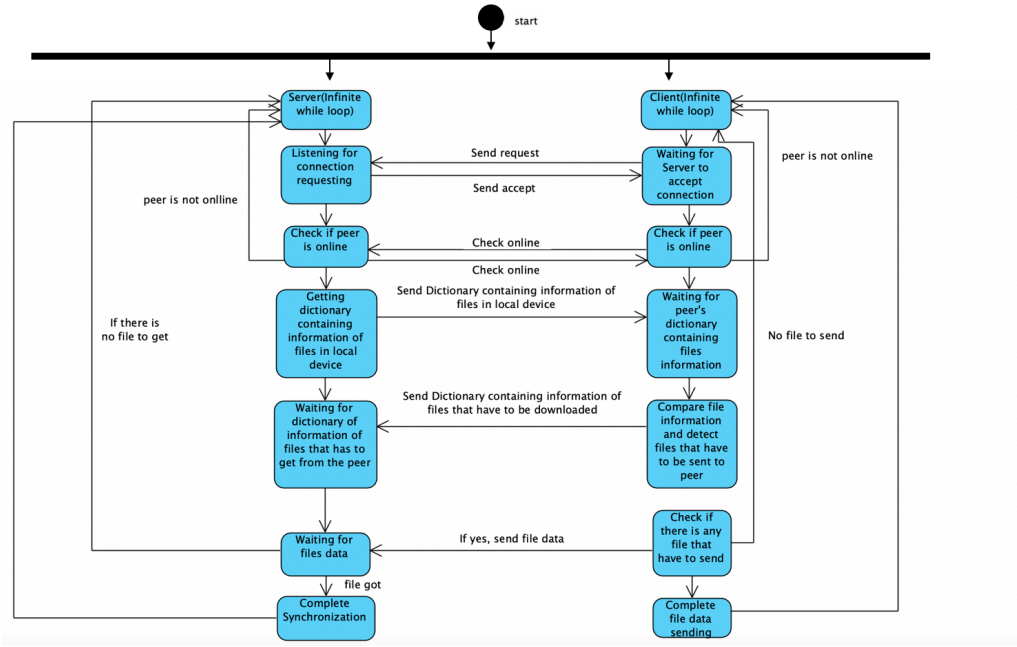

Figure1. State Machine Diagram

4. Implementation

4-1. Steps of Implementation

-Server thread

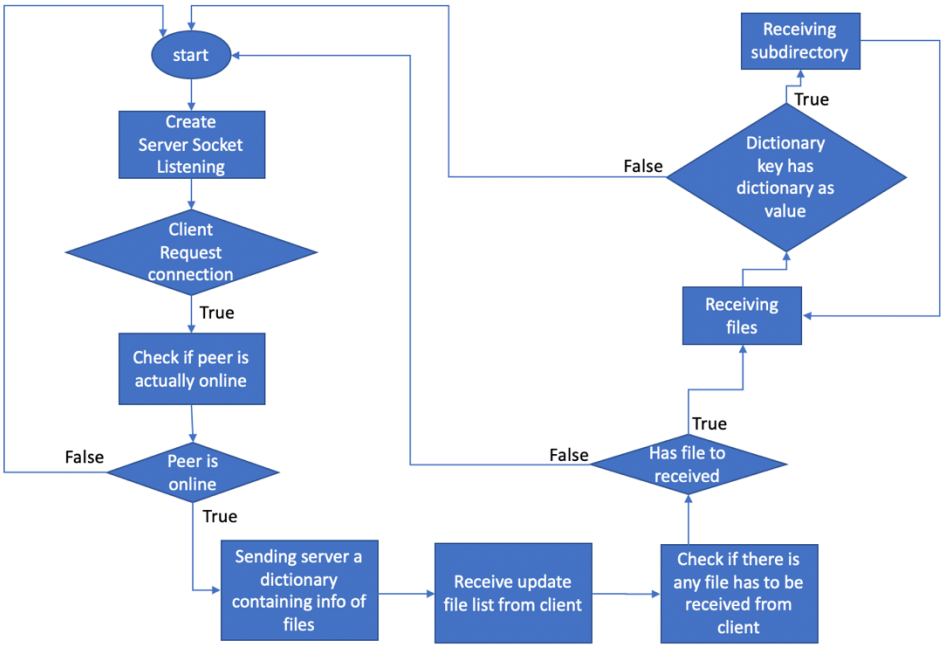

Figure 2 Flow Chart of Server thread

Once client thread has started, it creates socket, binds, and listens to client's request. Server waits until client requests for connections. After connection, server first check if client is actually online. Then server gets a dictionary of information of files in one's device with file_info() function. A dictionary will have file name as a key and [file_size,

file_md5_value] as a value. Since client is online, server send a dictionary to client. It will later get a new dictionary containing file information that has to be downloaded from client and according to that dictionary, server will receive the files. In this receiving process, if the dictionary has value as dictionary that means server will receive sub-directory, so folder_download(self, folder_name, folder_files) function will be called.

-Client thread

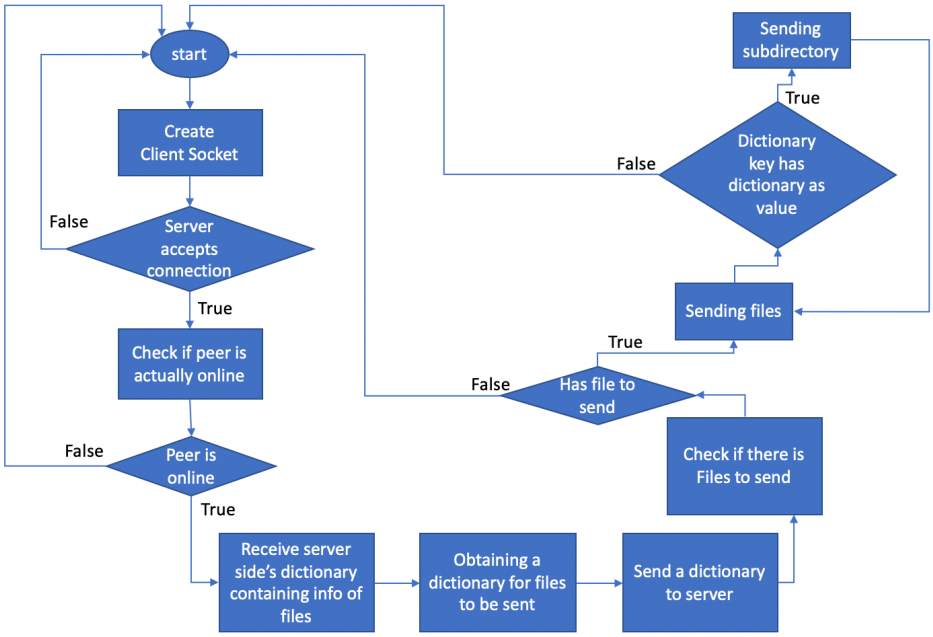

Figure 3. Flow Chart of Client thread

Once client thread has started, client socket will be created and client will request connection to the server. After it is accepted by server, client will check again if server is actually online. When they check they are connected, it will receive a dictionary containing information of files in the server's device. Once client receives that dictionary, client will get a dictionary of information of files in one's device with file_info() function. Then, it will start to compare that dictionary with one's dictionary containing the file information of local device and create a new dictionary containing information of files that has to be synchronized. If client checks there is any file that has to be sent, it will start send the files. In this sending process, if the dictionary has value as dictionary, that means client will send sub-directory, so send_folder(self, folder_name, folder_dict) will be called. After sending all files, client will close the socket and go back to the beginning of while loop.

4-2 Programming Skills

1) OOP (Object Oriented Programming)

In this program, there are two class Server, and Client and as it is stated above, each class has own functions. Therefore, it increases re-usability and decreases data redundancy.

2) Multi-threading

Each Server thread and Client thread are running within one process. It allows multiple tasks to be performed concurrently in one program. Therefore, two peers can be a server and a client at the same time.

4-3 Difficulties

The concept that a machine should be a server and a client at the same time made it difficult to create the flow of code at first. Since one machine is a server and client at the same time, interference occurred when reading and writing file. In order to solve this problem, lock.acquire() and lock.release() are used in order to prevent interference when each thread is reading and writing data.

5. Testing and results

5-1. Testing in Testing Environment

```
**Have linked to PC_A and PC_B. Ready to test.
**** PHASE 1 ****
Start to run your code on PC_A
Parsing argument...
>>>Share folder already exists
...Listening...
**** PASS PHASE 1 ****
```

Figure 4. Phase 1 Testing Result - Pass

```
**** PHASE 2 ****
Move file1.bin (File_1 in the handbook) on PC_A to the share folder.
Start to run your code on PC_B
Parsing argument...
>>>Share folder already exists
...Listening...
...Connection request has sent to ip 192.168.56.10 ...
>>>Server is connected to 192.168.56.11
...Connection request has sent to ip 192.168.56.11 ...
>>>Server is connected to 192.168.56.10
dictionary sent
dictionary sent
No file to download, already synchronized
...Connection request has sent to ip 192.168.56.11 ...
>>>Download Completed
>>>Server is connected to 192.168.56.10
No file to send, already synchronized
...Connection request has sent to ip 192.168.56.10 ...
>>>Server is connected to 192.168.56.11
MD5_1B: PASS
```

Figure 5. Phase 2 Testing Result - Pass

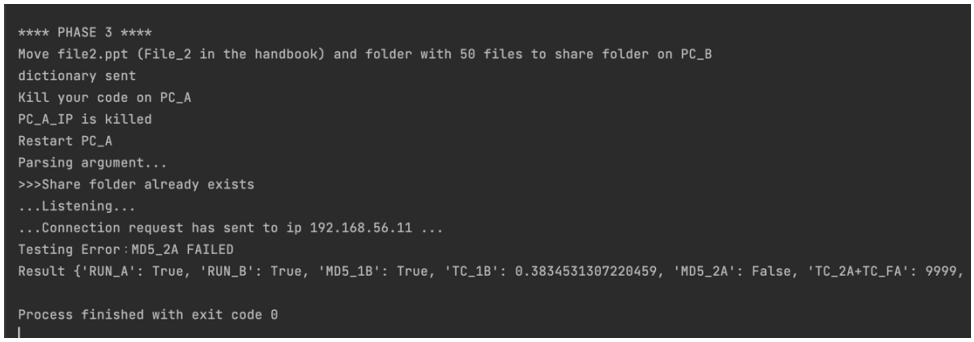

Figure 6. Phase 3 Testing Result – Failed

5-2. Result

Phase 1 is passed and Phase 2 is also passed with about 0.38345 sec. The result of Phase 2 indicates that data is well sent and received between two peers. Since TCP socket is used, as long as server and client are connected well, reliable transport is guaranteed. However, the program was not able to pass Phase 3. Shutting down PC_A and restarting cause the error. Since they only check once if they are online at the very first time they cannot know if they are still online if the program is shut down at one side. By constantly exchanging messages with each other, peers have to check if they are still online state or not. This program requires a part that disconnects and returns to connecting part when detecting one peer is not online.

6. Conclusion

The result tells TCP guarantees data reliability since, it send and receive data without error when two peers are well connected. Therefore, TCP is suitable for the application that synchronizes the file to the particular machine people want without data lose.
However, in order to detect the situation which is one peer suddenly become offline, each peer should send and receive message to keep track the other peer is still online.

7. Reference

[1] Y.-N. Lien and H.-Q. Xu, “A UDP based protocol for distributed P2P file sharing,” *Eighth International Symposium on Autonomous Decentralized Systems (ISADS'07)*, 2007.