CAN201 Introduction to Network

Coursework 1 Report

Name: Weijie Xie

ID: 1927806

Abstract:

In this information era, almost all fields are using the network, such as remote teaching, online office and online information query. The Internet is often used as a medium for transferring files and information. More and more advanced network technologies such as 5G make file transfer and sharing more and more efficient and accurate. To get a better acknowledge of how network works in application layer, in this project, a simple file transfer system that can be used to synchronize different files on two computers will be established, in which the socket will be employed

Introduction:

Requirement:

The project is aim to build up a link between two virtual machines with python, which should meet the following requirements:

1. The program can transfer various file regardless of its type or size.
2. Try to make the transfer rate as fast as possible, besides, the files should synchronize automatically.
3. The program should have the possibility to deal with some unexpected issue and the IP address of the peer should be an argument.
4. The file should be exactly the same between the sender's side and the receiver's side.

From the requirements above.

Background:

Nowadays, there are many mature software that can meet the above requirements, such as Baidu Netdisk, Thunder, Google Drive, etc. Behind the smooth operation of these software is a large number of reliable software and hardware support, as well as the design and application of many sophisticated transmission protocols. The transmission rate of these

software can reach an amazing upper limit meanwhile ensure the security and accuracy of file transfer.

Literature Review:

Reliable file transfer protocol is very important in the working process, it is an important guarantee to ensure the accurate and safe transmission of files. The network protocol is constantly updating and iterating. Its reliability and efficiency have gradually improved substantially. For example, a file transfer protocol with producer anonymity is recently proposed by researchers[1]. Researchers claim that this new network transmission protocol can greatly improve the security of information transmission under existing network conditions.

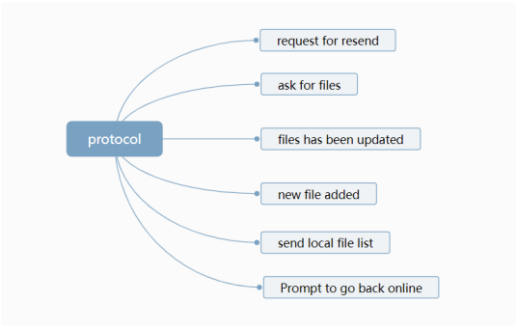
In addition to topics related to network protocols, Others have proposed a series of inspection methods for testing transmission stability.[2] Related researchers pointed out that a reasonable ratio of test files should be selected when testing the transmission effect, so that the data obtained when testing the transmission effect is more meaningful.

Main project mission:

In this project, a program that can be used at the same time as the client and server will be written. And this program can automatically check and send updated files or folders (file sizes ranging from 1kB to 500MB), and can decode and download files smoothly when used as a receiver. And it can be reconnected in the event of a network disconnection. A simple network protocol will also be designed.

Methodology:

Proposed protocol:



Proposed functions:

Function name (variables)	Return value	usage
_argparse()	none	to import the peer's ip
create_folder(file_name)	none	to create a new folder which is detected in the received files
traverse_share(dir_path)	list	to traverse the share list and return a list of file name in share
generate_full_list()	list	to return a list of file name, modified time and size
generate_name_list(file_list)	list	get a file name list
generate_moditime_list(file_list)	list	get a modified time list
generate_size_list(file_list)	list	get a size list
connect(port)	none	used to connect and reconnect PC_A and PC_B
thread_new_file_scanner()	none	a scanner which keeps scanning the local share folder and send out message to peer
file_downloader(file_wanted_list)	none	used to receive files and save them in share folder
listener()	none	keep listening to the peer and receive messages
def file_sender(file_name)	none	used to pack files and send them out

Idea:

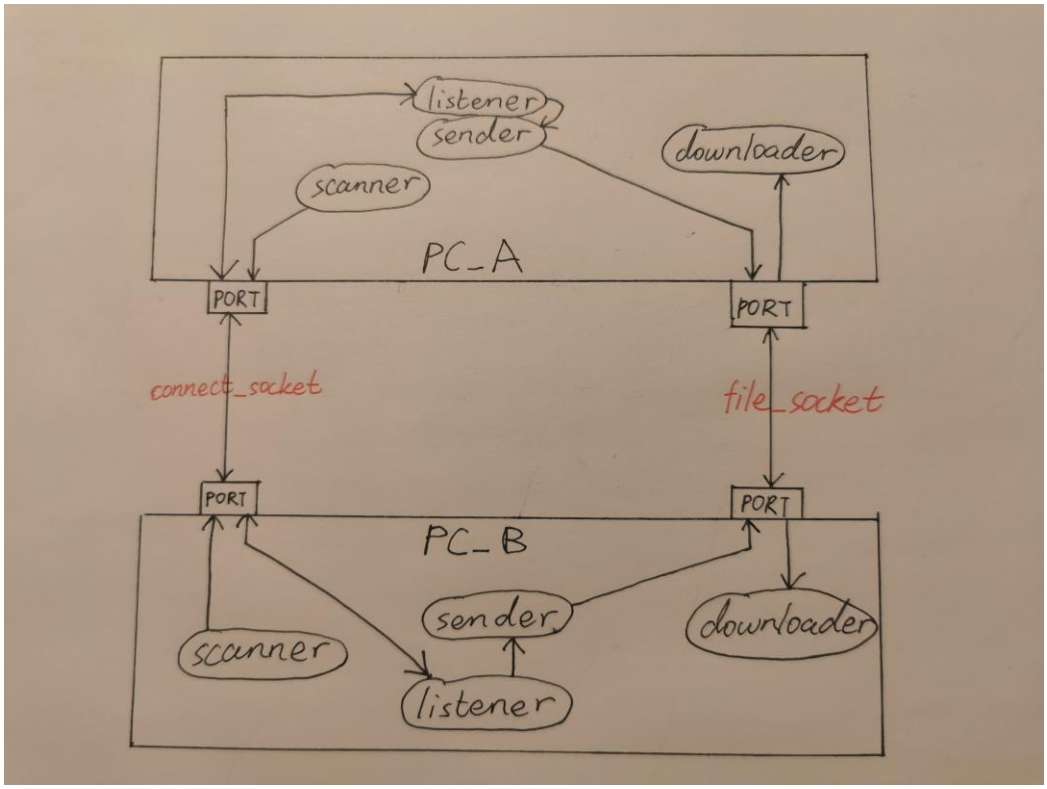
There are two types of data transferred between two virtual machines, one is messages, and the other is file data. In order to make packing and unpacking easier, and to increase the transmission speed. A double-socket method was adopted. Messages will be transmitted through message sockets and file data will be transmitted through file sockets.

Meanwhile, since the behavior of checking the new file does not conflict with sending and receiving data, the checking of the new file will be performed in a separate thread.

The methods that are responsible for different tasks will be packaged one by one, which makes repeated calls easier and more convenient and makes program maintenance easier.

Implementation:

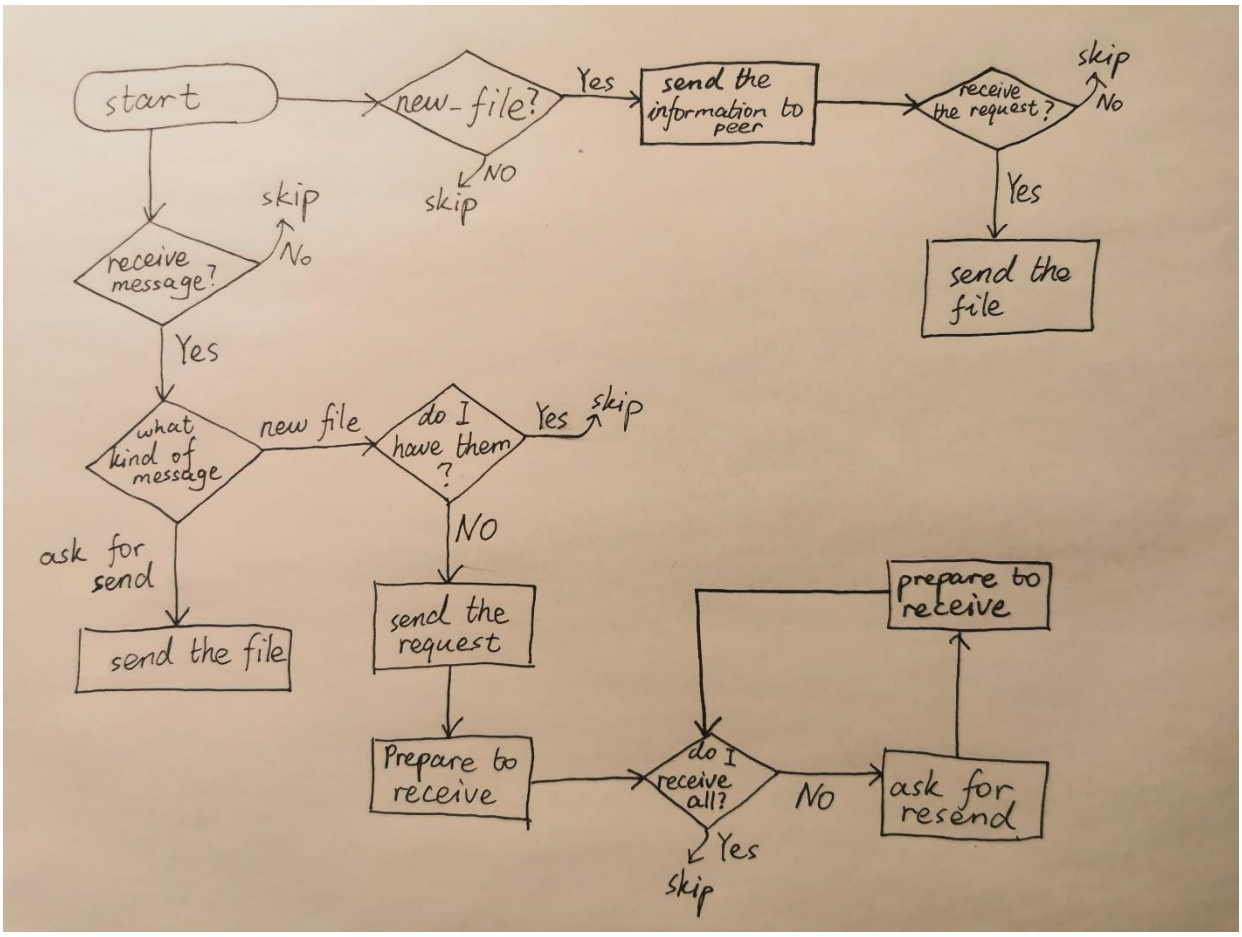
The modal of implementation is shown in the following picture:



As shown in the picture, the two big rectangles stand for the two virtual machine PC_A and PC_B. And there are two sockets between PC_A and PC_B. Since the two are exactly the same, only one of them needs to be studied.

From the picture, it can be seen that the listener is the core of the program, as it may send and receive messages from the connect_port while also give instruction to sender.

Below is the flow chart of one of the PCs:



As shown in the picture, two lines are drawn at both ends of the start symbol, which means that the program has two threads and does not interfere with each other.

In the scanner thread, when a new file (includes both update one and new added one) is detected, the scanner will send a message to ask the other virtual machine if it needs the new one, then it will wait for a request. If the scanner doesn't receive a request for new file, it will begin a new loop, else if it receives a request from the peer, it will directly send the new file to the peer.

In the listener thread, it will keep listening to all the messages send by the other virtual machine. For example, while the listener receives the message that the peer has a modified file, by comparing the file with the local files, it doesn't have such a file. Thus, it will send a

request for obtaining the file and wait for its coming. Meanwhile, the peer receives the request and send out the file immediately. However, not all the files are received. Then it will send a message to the peer to resend the file data until it gets the complete file.

Programming skills:

The most typical skill that is applied in this program is OOP. For that the user only need to enter the IP of the peer and then move the files which are need to be synchronized, the program will run automatically regardless of the size or type of the file, which shows one of the features of OOP—Polymorphism.

In addition, while programming, all methods for performing different tasks are encapsulated, leaving only specific interfaces for people to use, which is also a feature of OOP.

Difficulty during programing:

1. How to detect the link is killed?

At first, I wanted to send a heartbeat packet to inform the connection that the connection is stable, but this method has a disadvantage that it is not efficient. You have to wait at least one heartbeat cycle to know that the connection is disconnected. Finally, in order to reconnect the time, I chose to catch the exception to reconnect. This method can almost catch the anomaly at the moment the network is disconnected, and reconnect quickly.

2. File cannot be found:

```
f = open(file_name_to_download, 'ab+')
FileNotFoundError: [Errno 2] No such file or directory: 'share/folders/fxx_23.txt'
```

At first, I was very puzzled why this method of opening a file would report an error, saying that the folder could not be found, as this opening mode would automatically create a new file with the same name if it could not be found. Finally, after searching for information, it was found that a file cannot be created in this way without the folder to which it belongs. To solve the problem, these codes were added:

```
try:
    f = open(file_name_to_download, 'ab+')
    f.write(bin_file_to_download)
    f.close()
except:
    create_folder(file_name_to_download)
    f = open(file_name_to_download, 'ab+')
    f.write(bin_file_to_download)
    f.close()
```

Thus, problem solved.

Testing and results:

Testing environment:

Computer A and Computer B will be simulated by two Linux single-core virtual machines opened by VMware. The CPU model of the host running the code is i7-9750H.

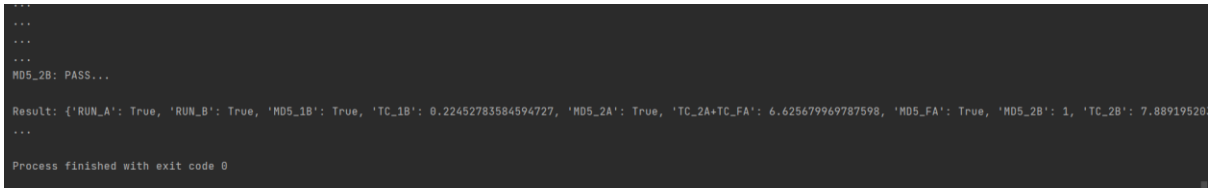
Python 3.9 is applied.

Test plan:

When the program is running the screen will keep printing "…" to inform that the program is not stuck. The program will be run for 5 times and the screen will be recorded later the running time will also be recorded and obtain the average running time.

Test result:

Test 1:



Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.22452783584594727, 'MD5_2A': True, 'TC_2A+TC_FA': 6.625679969787598, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 7.889195203781128}

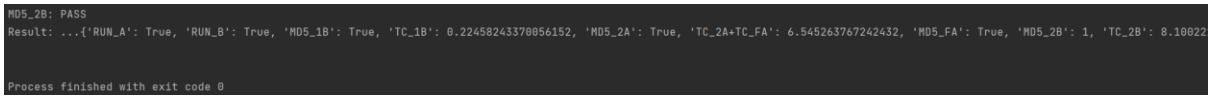
Test 2:

Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.22488927841186523, 'MD5_2A': True, 'TC_2A+TC_FA': 6.502185344696045, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 8.115586042404175}

Test 3:

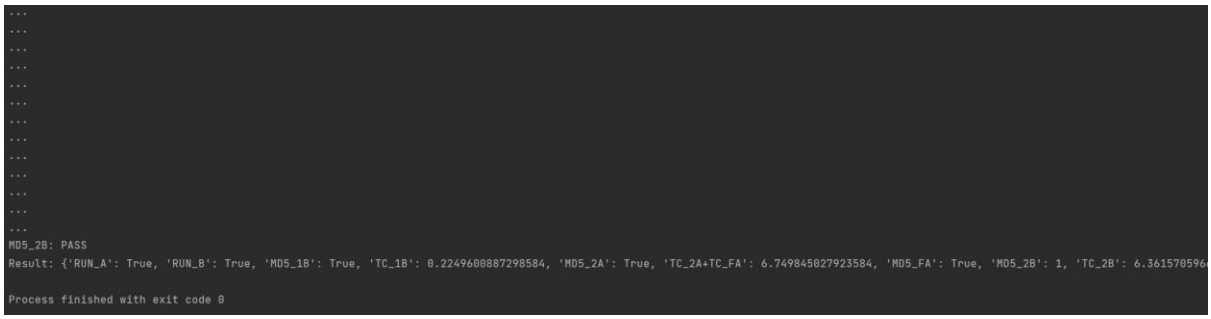
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.2241830825805664, 'MD5_2A': True, 'TC_2A+TC_FA': 6.834656238555908, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 6.004943609237671}

Test 4:



Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.22458243370056152, 'MD5_2A': True, 'TC_2A+TC_FA': 6.545263767242432, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 8.10022234916687}

Test 5:



Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.2249600887298584, 'MD5_2A': True, 'TC_2A+TC_FA': 6.749845027923584, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 6.361570596694946}

The result table:

	TC_1B	TC_2A+TC_FA	TC_2B
Test1	0.22452783584594727	6.625679969787598	7.889195203781128
Test2	0.22488927841186523	6.502185344696045	8.115586042404175
Test3	0.2241830825805664	6.834656238555908	6.004943609237671
Test4	0.22458243370056152	6.545263767242432	8.10022234916687
Test5	0.2249600887298584	6.749845027923584	6.361570596694946

Average time	0.22	6.65	7.29

Conclusion:

In this project, I designed and established a simple network connection to connect two virtual machines through IP addresses and make the data on the two virtual machines transfer and share with each other. And by designing the protocol, the data transmission is accurate and fast. In the future, this file sharing program can be improved a lot, such as multi-threaded multi-port synchronous transmission, multiple computers interconnect and transfer files to each other to simulate P2P mode, etc. This project is generally very successful.

[1] C.-I. Fan, A. Karati, and P.-S. Yang, “Reliable file transfer protocol with producer anonymity for Named Data Networking,” *J. Inf. Secur. Appl.*, vol. 59, p. 102851, Jun. 2021, doi: 10.1016/j.jisa.2021.102851.
[2] N. Mills, F. A. Feltus, and W. B. Ligon III, “Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks,” *Future Gener. Comput. Syst.*, vol. 79, pp. 190–198, Feb. 2018, doi: 10.1016/j.future.2017.04.030.