Xi'an Jiaotong-Liverpool University

西交利物浦大学

Large Efficient Flexible and Trusty Files Sharing program Report

Author Mingzirui WU

Module CAN201

Teacher Dr. Fei CHENG

Date 28^{th} / Nov / 2021

Abstract

The aims of this project are to use Python Socket Programming to design and implement a LEFT file sharing program between two hosts A and B. Block transfer and own application layer protocol design is needed for the coursework requirement and the application layer protocol is supported by the transmit layer protocol TCP.

1 Introduction

A college student called Shawn Fanning, start a project called Napster which start the file-sharing revolution in 1999. Unfortunately, record labels and musicians indicted the online file-sharing service in 2000 [1]. Although record companies who as the first batch to feel file-sharing's impact tried to block it, file-sharing still as strong as before under the lead of BitTorrent [2].
Requirements of this coursework:

1. The program can transmit any kinds of files with the maximum of size is up to 500 MB, and also can transmit the folder and the files in the folder.
2. The program should synchronize any files/folder (including new and changed) in the '/share' folder in host A and B automatically.
3. The program can run in any host, which mean the IP address should be set as an argument
4. The transmitted file and folder should have no error.

Base on the requirements, a new application layer protocol to synchronize the two folders in the two hosts is designed. It uses the C/S the TCP as transport layer protocol to guarantee the data have no error during the transmit. The C/S architecture with two port number, which are 25000 and 25001 were used to transmit semaphore information and file data respectively.

2 Methodology

Considering the C/S architecture, separate parts of client and server are applied (There is no difference between host A and host B, both of them have same code). The obligation of server is to send file to client, it also judgement whether the file should be sent and client part is responsible for receiving file and folder. They have different operation in the application layer protocol (Figure 1).
Server first checks the **sending_log.txt** to guarantee no file transmit failed because of unknown reason. If exists file in the **sending_log.txt**, it will be retransmit first. Different transmit functions were designed for server which can be recognized by client via scan the type message. And the type message is a part in header data. Server checks file name while traverse share folder, file in the **check_log.txt** will not be transmit. Server and client maintain **check_log.txt** together to

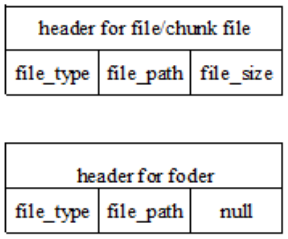

Fig. 1: Header Information

make sure they both know which file is sent by server, and which file is received by client. After traverse share folder, server checks **all_file_dir** (This dictionary will record each file name with their final modify time when server traverse share folder). If any file has different final modify time in the dictionary, this file will be retransmit immediately (Figure 2).
Client scan the file type message first, call different type of function to receive item. Because server send different type of header data, client has different method to read the header data (Figure 3).
Header data is a string converted from a dictionary by the **JSON** module. For folder has no the attribute called **file_size**, because my client only needs to know the folder path from server.

Table 1: How the file type message work in client

File_type	Client Operation	Function name in code
0	Get folder path (folder_name) and create new folder (if not exists)	rcv_folder(folder_name)
1	Get file path (file_name), file size and start to receive file data.	rcv_file(file_name)
3	Get file path (real_name), file size(>10MB) (real_size) and start to receive file chunks.	rcv_chunk(real_name, real_size)

3 Implementation

To implements this coursework, I went through the following steps:

1. List requirements of the coursework.

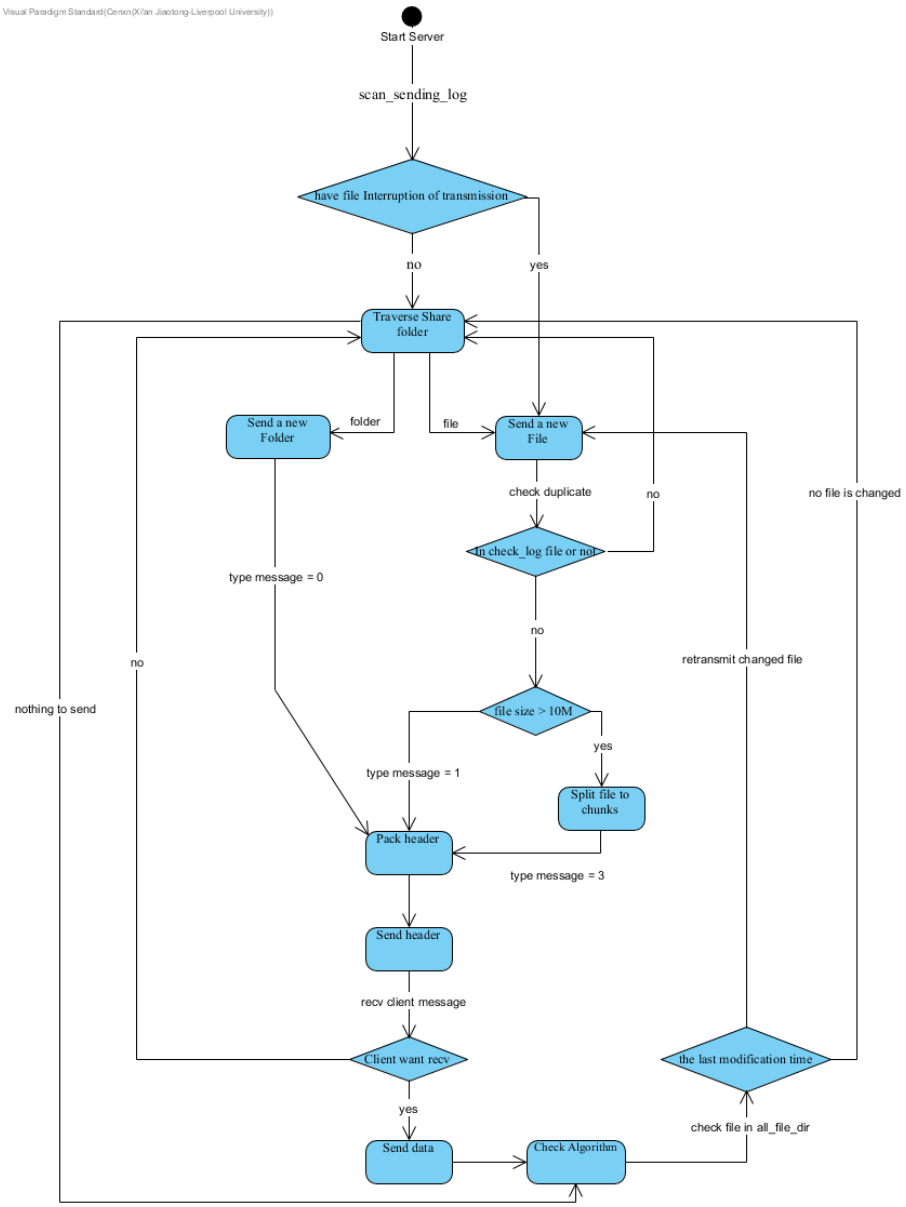

Fig. 2: FSM of server

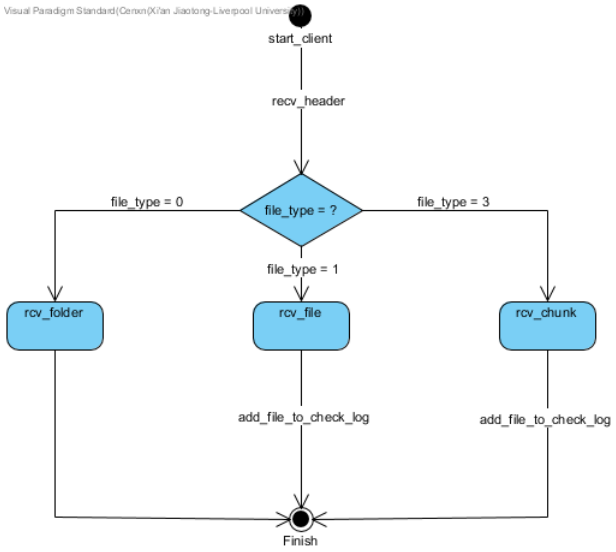

Fig. 3: FSM of client

2. Design server and client logic base on the requirement.
    - Use client maintain the function of receive item
    - Use server maintain the function of send item
3. List function respectively base on the logic of my client and server. Each function has different capabilities, it will be put in the public area if both server and client need its function.
   Needed public functions:
    - Get IP address from user.
    - Check whether a file is recorded by the log text.
    - Delete file from log text.
    - Scan log text as list.
    - Split big file into stable chunks.
4. Make sure server and client has their own function which cannot be called by the other.
   Needed server function:
    - Send normal file (size $\leq$ 10MB).
    - Send large size file (size = 500M), split the file to chunk and send.
    - Send folder.
    - Traverse share folder (find sending file).
    - Check all_file_dir find changed files.

   Needed client function:
    - Receive normal file (size $\leq$ 10MB).
    - Receive large data chunks and join chunks.
    - Receive folder.
5. Make the start operation of server and client become a function, so that they can be called as multi-process.
6. Use multi-process to maintain client and server, ensure both host A and B have ability to received or send items.

3.0.1 Programming skills and difficulties encountered:

As implementation process above, different functions for different ability requirements for server and client are designed which could reduce code redundancy.
Problem: Stick package problem happen when sending all the data in that file at once.

Solution: send file data line by line, which can avoid this problem.

Problem: PC_A starts before PC_B, client of PC_A meets **ConnectionRefuseError** when test.
Solution: Add while True sentence out of client connection sentence, if no server is on, client wait until connection success.

4 Test and results

Test environment:

Fig. 4: Virtual Machine configuration

Test plan:

1. Check whether the server and client can start successfully.
2. Check whether a file (size = 10MB) can be transmitted between two hosts.
3. Check whether a file (size = 500MB) can be transmitted between two hosts.
4. Check whether a folder with 50 files (size = 1MB) can be transmitted between two hosts.
5. Check whether the modified file (has been transmitted before) can be transmitted between two hosts.

Test result (11 success record):

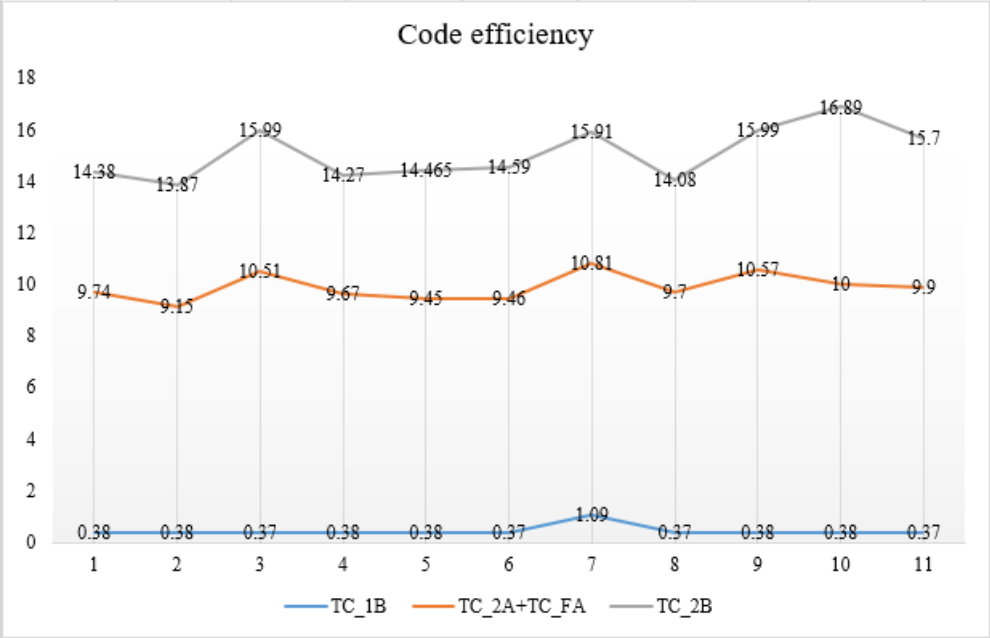

Fig. 5: Test record

Because the client restart code is set in the mian.py file, the restart time vary depending on the PC configuration environment. Restart time is 15sec which means test1 to test4 should finish their work in the 15 seconds. This means that the code has a chance of failing and is unstable for the result for the last test.

```
MD5_1B: PASS


**** PHASE 3 ****
Move file2.ppt (File_2 in the handbook) and folder with 50 files to share folder on PC_B
Kill your code on PC_A
PC_A_IP is killed
Restart PC_A
'./share' folder exists!
'./chunk' folder exists!
'./temp_chunk' folder exists!
check_log.txt on set.
./sending_log.txt on set.
Server want accept 192.168.244.129
Info and file client connection success: 192.168.244.129Server is ready: 192.168.244.131

Server want accept 192.168.244.131
MD5_2A: PASS
MD5_FA: PASS
Server is ready: 192.168.244.129
Info and file client connection success: 192.168.244.131
Info and file client connection success: 192.168.244.129
Server is ready: 192.168.244.131
/home/tc/workplace/cw1/share/folders already exist!
/home/tc/workplace/cw1/share/folders already exist!
********** re-transmit changed: ./share/file2.ppt **********
MD5_2B: PASS
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.3278350830078125, 'MD5_2A': True, 'TC_2A+TC_FA': 8.416890621185303, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 16.990864038467407}
```

Fig. 6: test printing record

After test 3 finished, **client of PC_B** will restart to avoid **server of PC_A** keeping waiting accept message. The **check algorithm in PC_A** checked the modification of **file2.ppt** and print message: **re-transmit changed file2.ppt**.

5 Conclusion

In this coursework, I designed and implement own application layer protocol for two hosts file synchronization which passed the test in Python. The program can split large file into chunks with same size and send it to the object host while breakpoint re-transfer of file are allowable. In the future, the check change file function should be improved. This application layer protocol is not stable and efficiently enough, client should request files from the server pro-actively.

6 Reference

[1] L.Sydell, ”Napster: The File-Sharing Service That Started It All?,” National Public Radio, 21 12 2009. [Online]. Available: https://www.npr.org/2009/12/21/121690908/napster-the-file-sharing-service-that-started-it-all. [Accessed 27 11 2021].
[2] R.Levi, ”The History of File Sharing,” Curious Minds, [Online]. Available: https://www.cmpod.net/all-transcripts/history-file-sharing-part-1-2-rise-fall-napster_text/. [Accessed 27 11 2021].