Xi’an Jiaotong-Liverpool University
西交利物浦大学

CW1: Large Efficient Flexible and Trusty (LEFT) Files Sharing

Author	Jianghan Chen
ID	1929557
Module	CAN201
Teacher	Fei Cheng
Date	29th/November /2021

Abstract

The main purpose of this coursework is to synchronize files between two hosts using limited memory and computing resources. During the transfer process, chunked transfer is used to avoid memory overflow, and it also has breakpoint renewal and modification retransmission functions. In addition, the application layer protocol of this project is based on TCP.

1. Introduction

1.1 Project requirement

The requirements of this project are as follows:

1. Use python socket network programming
2. Transport layer protocol should be TCP. The architecture of this project should be C/S, P2P or mixed and the port be used should between 20000 and 30000
3. When one of the peers disconnects and reconnects, the program should be able to continue to run and transfer files that have not been transferred or have not been fully transferred
4. When certain files are modified, the program should be able to determine which files have been modified and synchronize the modified files

1.2 Background

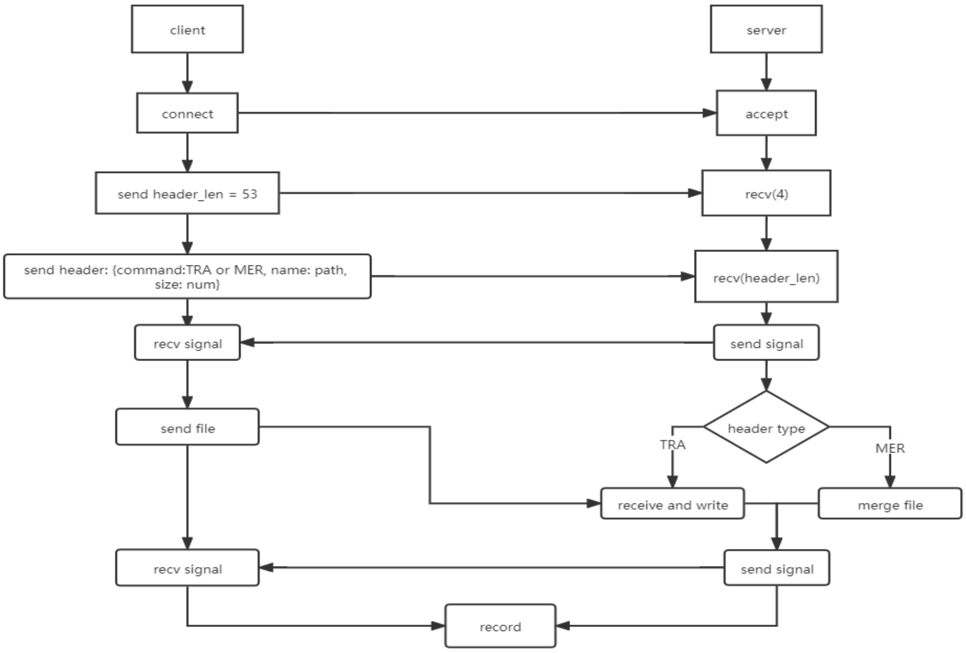
Currently, as Li, et al. pointed out, technologies such as big data cloud computing have developed rapidly in recent years [1]. At the same time people often have multiple devices and more data at the same time. Therefore, the role of high-quality applications for sharing data between multiple devices and remote backup of data on a large scale is becoming more and more important. Applications such as Baidu NetDisk and Google Drive have been a huge success, and the key technology they use is data sharing.

1.3 literature review

In 2021, Trautwein et al. introduced a peer-to-peer tool called Peer Copy [2]. They claim that the innovation of this tool lies in the data relay mechanisms that make centralized server outdated and the extensive architectural differences in peer-to-peer discovery infrastructure [2]. In addition, the verification algorithm also affects the efficiency of file transfer. Around 2020, Alhussen and Arslan came up with Fast Integrity Verification algorithm, and this algorithm can reduce the cost of end-to-end integrity verification to less than 15% [3].

2. Methodology

2.1 protocol process



2.2 Proposed protocols

2.2.1 Header

Header contains three attributes: command, name (relative path) and size (File size)

Command	Description
TRA	Transfer a small file
MER	Merge the specific temporary files into one large file

2.2.2 Protocol Design

The application layer protocol is based on TCP, and TCP can ensure the orderliness and consistency of the data. Therefore, this application will not consider these two points. Nevertheless, the TCP protocol still has the problem of sticky packets, and therefore this protocol specifies that the socket sends a signal to the sender immediately after receiving data to indicate that it has received the information in order to solve the problem of sticky packets and maintain the orderliness of code operation (expect when accept header data, which will be explained later).

The header data is transferred in two parts, once for the length of the header data, and the second part for the header data itself. Since the length of the header data is a fixed 4-byte length, the recv method of socket accepts the size of the data explicitly and is itself blocking, and therefore it will not cause the above problem.

2.3 Proposed functions and ideas

1. Considering the memory limitation of this project, large files are transferred in chunks instead of compressed because the size of the compressed files also does not support direct transfer.

2. file_record can record the received but not finished files and the finished files, so as to avoid the retransmission of the same file and the mistransmission of the uncollected files.

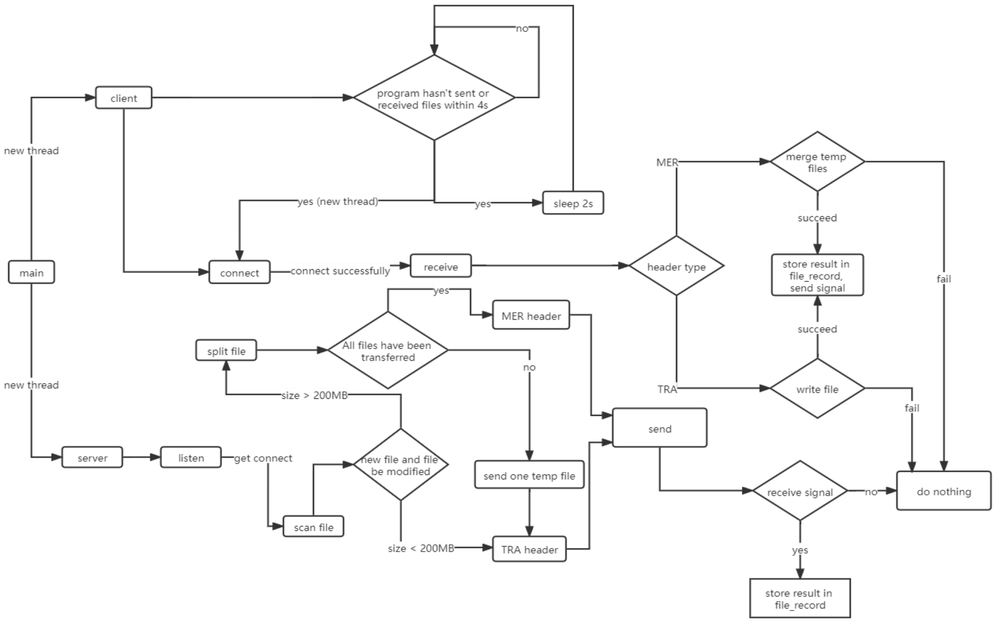
3. Implementation

3.1 Steps of implementation

First specify the assignment requirements, outline requirements, then write a stable version, test it and get the results. Second, optimize the original version, get the final version and test it. Finally, write a report

3.1.1 Program flow

Program flow chart is as follows:



The main function opens two threads for client and server respectively. When the server receives the connection request, it will query the file to be transferred and then send it to the client side. And the program will record the transfer information and file information when accepting and transferring files to prevent miscommunication and retransmission. The client side will create a thread to check if the program is still accepting or sending files, and when it finds that the program has stopped accepting or sending files for a period of time, it will send a connect request to the server so that the server will re-detect the files to be transferred and re-transmit them.

3.1.2 Main modules

1. client: Send connections request, accepting files.

2. server: Receive requests, send files.

3. utils: Methods for modifying file_record, detecting and returning modified files, chunking, merging files, recursively querying absolute paths to files, etc.

4. dataheader: Wrapping header and it is used to simplify operations

5. main: Main startup function, initializing certain important parameters

3.2 Programming skills

1. Concurrent Programming

Open the client and server threads when the program starts, and check if the program stops receiving or sending messages for more than a period of time, and open the connect thread if it exceeds time limit.

2. Object-oriented programming

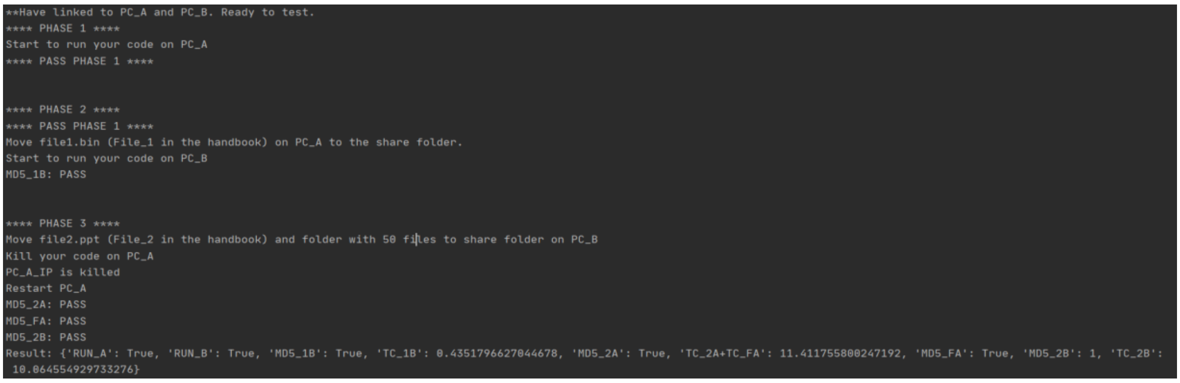
   Wrapping headers into DataHeader

3.3 Difficulties Met

I started out using multi-threaded file transfers, where one thread is opened for each file sent. Compression was done when transferring large files, but the program was not stable. After analysis, the size of the compression was unstable, which caused the file to not be transferred when it was too large. If I choose to compress the file and then slice it or slice it directly, the time consumption would be very high because of the frequent opening of threads, and even if I use thread pooling, it still does not work well. Therefore, I gave up on this transfer method. The final approach is to use a single thread for all transfers

4. Testing and results

One of my results:



```
**Have linked to PC_A and PC_B. Ready to test.
**** PHASE 1 ****
Start to run your code on PC_A
**** PASS PHASE 1 ****


**** PHASE 2 ****
**** PASS PHASE 1 ****
Move file1.bin (File_1 in the handbook) on PC_A to the share folder.
Start to run your code on PC_B
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
MD5_2A: PASS
MD5_FA: PASS
MD5_2B: PASS
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.4351796627044678, 'MD5_2A': True, 'TC_2A+TC_FA': 11.411755800247192, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B':
 10.064554929733276}
```

4.1 Testing environment

Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz   2.00 GHz

RAM: 8.0GB

Virtue machine: Linux version 5.4.3-tinycore (tc@box) (gcc version 9.2.0 (GCC)) #2020 SMP Tue Dec 17 17:00:50 UTC 2019

Memory of VM: 512 MB

4.2 Testing plan

4.2.1 Test data

1 x 10MB file, 1 x 500MB file and 50 x 10KB files

4.2.2 Test content

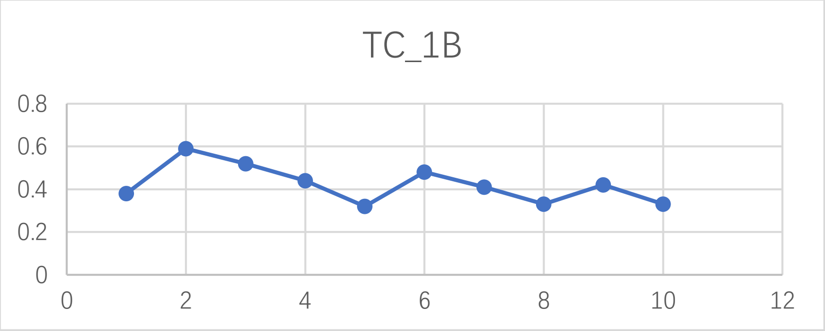
The test will use different parameters to test the 3 parts of single file transfer, breakpoint transfer and modified retransmission at the same time, but the results of the test will be analyzed separately.

4.3.3 Test Parameters

The parameters are the sleep time for the loop to check if the transfer is completed and the block size for the chunk transfer, respectively.

4.3 Test result

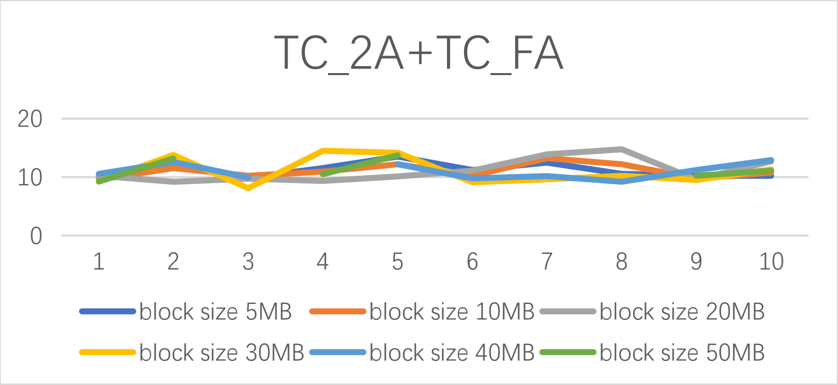
4.3.1. TC_1B



The average runtime for transferring single file is 0.442s

The results of TC_1B are not affected by any of the above parameters and therefore will not be explored further.

4.3.2 TC_2A+TC_FA

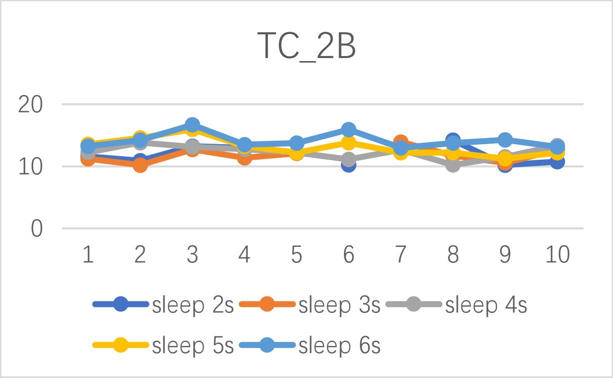


5, 10, 20, 30MB four parameters corresponding to the average runtime and standard deviation of the breakpoint transmission are as follows:

Block size (MB)	Average runtime (s)	Standard deviation
5	11.230	1.236878
10	11.048	1.232214
20	11.035	1.216301
30	10.992	2.362561

The errors that occur when the block size is 40MB and 50MB, and it is assumed that this instability will increase when the block size continues to increase. Then, according to the data and image analysis, the average running time does not change much when the block size is 20MB or 30MB, but the standard deviation of the former is smaller, so I choose 20MB as the final parameter

4.3.3 TC_2B



3s, 4s, 5s three parameters corresponding to the average runtime of the modify and retransmit are as follows:

Sleep time (s)	Average runtime
3	12.288
4	13.165
5	14.141

Since the program is not stable when the sleep time 2s and 3s are selected, it is not possible to select them. This situation may be caused by the fact that the last file transfer has not yet succeeded and the next transfer has already started. And it is easy to prove that the final running time increases proportionally with the increase of the sleep time. Although the program is stable and takes the shortest time when it sleeps for 3 seconds, I choose 4s for insurance purposes

5. Conclusion

In this coursework, I implemented a file inter-transfer system in python socket programming with functions of breakpoint transfer, and retransfer after modification. To implement this project, I designed the TCP-based application layer protocol and eventually refined the parameters of this application through testing.

However, file transfers are all in cleartext, so the security of the data transferred is extremely low. And my application does not consider the scenario of running on multi-core CPU, so I can't take advantage of multi-core CPU, these are the directions that can be studied in the future

Reference:

[1] W. Li et al. “Survey on Traffic Management in Data Center Network: From Link Layer to Application Layer,” IEEE Access, vol. 9, no. 1, 2021, pp. 38427 – 38456

[2] D. Trautwein, M. Schubotz and B. Gipp, “Introducing Peer Copy - A Fully Decentralized Peer-to-Peer File Transfer Tool,” presented at Espoo and Helsinki, Finland, Jun. 5-8 2021.

[3] A. Alhussen and E. Arslan, “Avoiding data loss and corruption for file transfers with Fast Integrity Verification,” Journal of Parallel and Distributed Computing, vol. 152, no. 6, 2021, pp. 33-44