Xi'an Jiaotong-Liverpool University

西交利物浦大学

Large Efficient Flexible and Trusty (LEFT) Files Sharing

Author Zhibin Lin

Student ID 1928196

Module CAN201-Networking Project

Teacher Dr. Fei Cheng

Date 23^{th} / Nov / 2021

Contents

I Abstract

The main purpose of this coursework is to implement a Large Efficient Flexible and Trusty (LEFT) Files Sharing program, which can transfer files to each other while saving. File chunks and consolidation are used in the transmission in order to make it faster and smoother. The application layer protocol is based on TCP.

Key Words: synchronize, TCP, FEFT.

II Introduction

II.A Project Requirement

The coursework requirements are as follows:

1. All folders and files in the ‘./share’ folder can be synchronized to the peer, no matter when the peer is online and when the new file appears. The maximum hosted transfer is 500MB.

2. All modification to folders and files can be synchronized to the peer automatically.

3. If any peer program is interrupted during file transfer, the synchronization can be resumed after restarting at the original schedule.

4. No file errors can be tolerated.

II.B Literature Review

In the case of the information age, people's data is often used on multiple devices. Therefore, applications for data synchronization are very necessary in this situation. Products like iCloud, one drive, drop box, google cloud, etc. were made by big companies when they found this situation. This proves the importance of data synchronization in the information age. File synchronization tools have been developed since the 90s[1]. Later, the algorithms for data synchronization were improved to reduce the transmission delay and increase the reliability [2][3].

In addition to the previous complex and inefficient hardware synchronization, there are now soft syncs, such as the major web drives (Baidu Cloud, OneDrive). According to the official documentation of these web drives, today's automatic synchronization tools have achieved fast transfer, breakpoint transfer, multiple file transfer, and other technologies [1]. The current coursework is also a rough-line imitation and attempt of these already maturing applications. The existing functionality of these auto-sync tools has contributed to the inspiration of this coursework. I have designed the entire application layer transfer protocol, designed the entire application synchronization protocol, written the entire project code, and written this coursework report.

III Methodology

III.A Purposed protocol

The package of each file is not necessarily long, and it only contains (12 + file length) bytes to make the header transfer faster. **Only first receive a header, peer can decide the next operation followed by interactive information (such as send files or receive files).**

- Interactive information: this code is a double digit. And it loads the operation information for peer to act in the next step.

- File name: it loads the file name.

- File size: it loads the file size (this means how much space the file should take up).

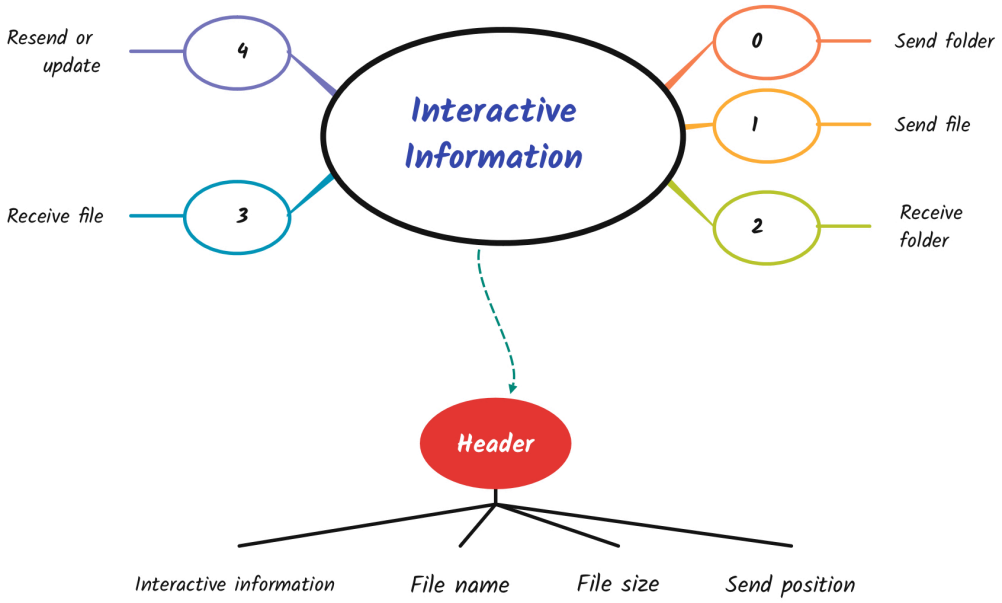


Fig. 1: Purposed protocol mind map

Table 1: Package

Interactive information	File name	File size	Send position
4 bytes	Flexible length	4 bytes	4 bytes

- Send position: it loads each file's current space taken up in local share (for resend, this makes the peer can write after the existed byte index).

III.B Proposed functions and ideas

1. Use file chunks function to divide a file into many small data chunks. For each file machine just sends one interactive package to peer, it will not cause the waste of header information. And it can also reduce the bandwidth of the transferred file.

2. For each received files, machine first mark a three-digit as a prefix. It can easily make the machine distinguish files. After sending all the file chunks, the prefix will be removed.

3. Machine has a 'temp' folder to temporarily store files that have not been transferred. Once the process finished, the files in the 'temp' folder will be moved into the 'share' folder.

4. For size and position, machine can decide whether the file/folder should be sent through comparing the size and the position. If the position equals to the size, then it means no need for sending.

5. Machine has a dictionary for storing the file's modify time. This is for updating by comparing the modify time of offline file with which in the dictionary.

IV Implementation

IV.A Steps of implementation

1. Define the requirements for the coursework.

2. Design an appropriate protocol to interact with peer.

3. Implement the major functions.

4. Divide the program into different modules and make codes clear.

5. Test and debug.

6. Improve the program according to the test results.

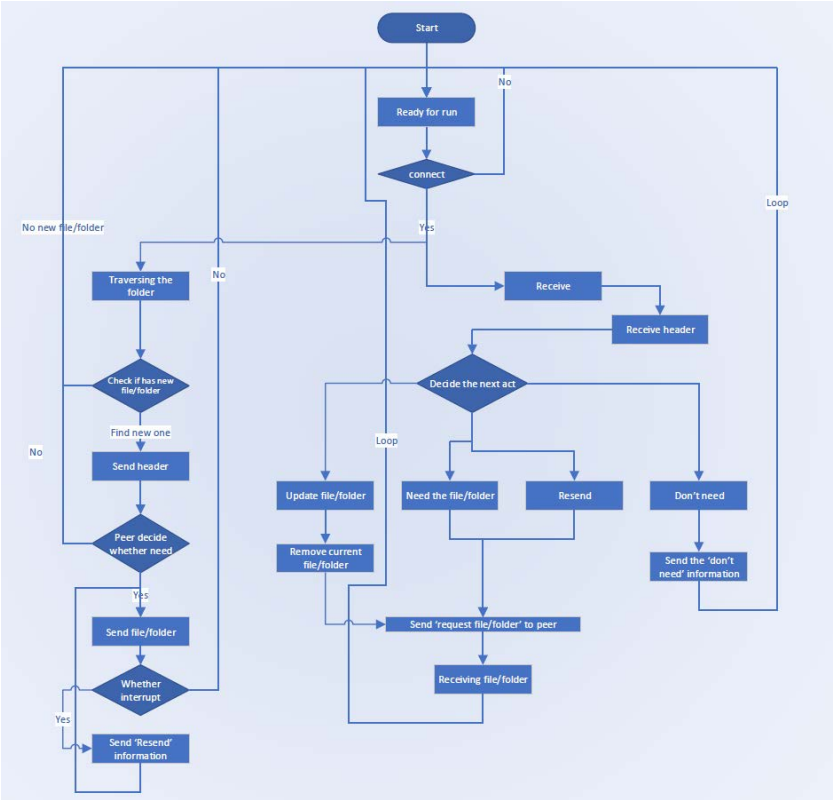


Fig. 2: LEFT flow chart

IV.B Programming skills

Object-oriented programming, modular and multithreading are used in this coursework. In the early stages of the project, unable to integrate server and client, use multithreading to solve this problem. After using multithreading, header-send functions sometimes output wrong information, such as empty list. Then a new thread which can check if peer is online is used.

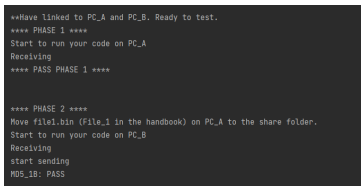
V Testing and results

V.A Testing

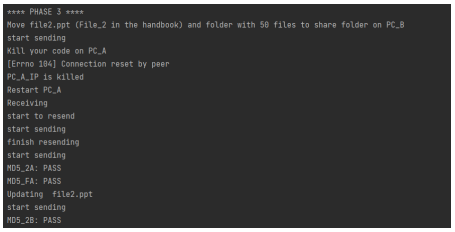
VirtualBox-6.1.14-140239-Win.exe
Core CW.ova —— provided by Dr. Fei Cheng
Testing plan is also provided:

1. A is executed with no error.
2. Move file1(10MB) to A, and execute B, synchronize both two machines. (Checked by md5)

3. Move file2(500MB) to B, then kill B, restart A after 0.5 second, synchronize the rest of the file2. (Checked by md5)
4. Move 50 folders (each one has a 1KB file) to B, synchronize all the folders and files. (Checked by md5)
5. Update the file2, then synchronize the file2. (Checked by md5)
6. Get the run time.



(a) Fig1



(b) Fig2

Fig. 3: Running

V.B Results

Result: the relationship between run time and buffer size can be illustrated by the following chart:

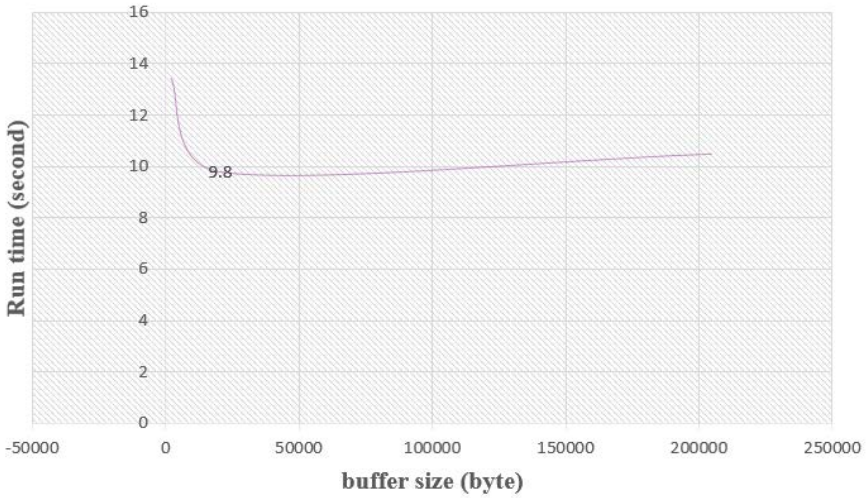


Fig. 4: Relationship

The run time firstly is reduced, and then is increased by the expansion of buffer. And then the relationship between run time and traverse frequency has been also discovered. The final result is shown in the following chart:

Fig. 5: result

VI Conclusion

In this project assignment, I designed an application layer transport protocol and implemented the code. In the process of optimizing the code, I found that there were some points that needed attention, such as the possibility of sending and receiving too fast between the mutual transmission of header protocols leading to program crashes, so I used time.sleep() several times in the local tests to avoid this problem. Also, the length of sleep time needs to be optimized for different types of computers. This is an issue that needs to be optimized in the future.

Another feasible way to be able to reduce the time is to use multi-threaded file transfer. However, the current code has not yet solved the problem of receiving too many files in a short time in the multi-threaded case and the multi-threaded breakpoint transfer, which will be the focus of future time optimization.

Reference

[1] J. Christoffel, “Bal-a tool to synchronize document collections between computers.” in *LISA*, 1997, pp. 85–88.

[2] S. Agarwal, D. Starobinski, and A. Trachtenberg, “On the scalability of data synchronization protocols for pdas and mobile devices,” *IEEE network*, vol. 16, no. 4, pp. 22–28, 2002.

[3] J. Hughes, B. C. Pierce, T. Arts, and U. Norell, “Mysteries of dropbox: property-based testing of a distributed synchronization service,” in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2016, pp. 135–145.