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

A Development Report on the Design and Implementation of a Large Efficient Flexible and Trusty (LEFT) Files Sharing program

Author Yifei Luo

ID Number 1928155

Module Introduction to Networking

Teacher Dr. Fei CHENG

Date 28^{th} / Nov / 2021

Contents

Contents **A**

1 Introduction 1
- 1.1 Problem Statement 1
- 1.2 Background and Literature Review 1

2 Methodology 2
- 2.1 General Design Idea 2
- 2.2 Protocol Design 2
- 2.3 Module Specification 2

3 Implementation 3
- 3.1 Implementation Steps 3
- 3.2 Programming Skills 4
- 3.3 Difficulties and Solutions 5

4 Testing and Results 5
- 4.1 Testing environment 5
- 4.2 Testing Plans 5
- 4.3 Testing Results 6

5 Conclusion 7

Bibliography **8**

Abstract

With the utilization of network-related technology, a plenty of network-based file sharing applications, including Dropbox, Google Drive, Baidu NetDisk, iCloud, and XJTLU BOX, are capable of transferring files forthwith. This report aims to illustrate the development of a Large Efficient Flexible and Trusty (LEFT) Files Sharing program, especially on its design and implementation. This report will begin with the introduction of the background and literature review, and subsequently demonstrate the methodology such as general ideas and app structure as well as implementation which mainly focus on the used programming skills and faced dilemma, and then test the code, eventually culminated in a general but accurate conclusion, together with appropriate future plans.

Key Words: File sharing application, Python Socket Programming, Network

1 Introduction

1.1 Problem Statement

In this coursework, with the utilization of Python Socket network programming techniques we should develop a file sharing application with four key features, including largeness, efficiency, flexibility and trustiness [1]. The first feature is largeness, where the size of each file is over 500MB and the format of file can be any type, excluding hidden ones. For example, in terms of audio files, we can transfer .aac .au .mid .mp3 .ra .snd .wma .wav; regarding image files, we can transfer .bmp .eps .gif .jpg .pict .png .psd .tif; as for text files, we can transfer .asc .doc .docx .rtf .msg, .pdf .txt .wpd .wps; as to video files, we can transfer .avi .mp4 .mpg .mov .wmv. Furthermore, efficiency requires high speed and automatic synchronization of files. There is no standard for high speed, so as long as the synchronization is faster, the efficiency is better. Subsequently, flexibility refers to the changeability of IP addresses and process's resumption from interruption. IP addresses in the program can be modified as we set them as an argument. To illustrate the resumption, if the file is crushed and we restart the program again, the file can be synchronized. The last feature is trustiness, namely no errors during the sharing process. This is the most fundamental requirement, so we must give a top priority to this feature.

1.2 Background and Literature Review

During the Covid-19 pandemic, almost all schools around the world conduct the learning process by E-Learning courses. Under such condition, emergent file sharing applications, such as Dropbox and Google Drive, become increasingly prevalent due to multiple advantages [1]. The first merit is the convenience this cloud storage method offers for various learning materials to be uploaded and stored, compared to its counterpart, traditional hard storage devices [2]. Moreover, it could guarantee the files security of users even in transmission [3]. For instance, Dropbox, an excellent platform, requires two separate steps of verification before used. The first one is private password, and the second one is a verification code sent to customer's mobile phone [4]. With such two steps, many Internet attacks may be prevented. With these advantages, the cloud file sharing applications gain a great acceptance [5]. According to my research, the complete process still lies in the threading architecture, despite the extra complexities involved by advancements in recent years. Thus, this project will use this key concept to develop a similar app.

2 Methodology

2.1 General Design Idea

This file sharing application is designed as an active mode, where the program will detect the files by checking the md5, modified time of each file from both two sides, resulting in four conditions of each file, including no changes, break-point resuming, adding, and updating. When a file is manually added in the current directory or is partially updated, the program will send it (or just send the partially updated part) to the same directory of its peers after checking process.

2.2 Protocol Design

Since the protocol in this methodology is mainly applied in TCP communication, there is no need to design complex and cumbersome protocols and make packets. Moreover, in order to save bandwidth of the Internet, the format of protocol is just designed as some integers other than the normal human-readable type. Here is a mind map illustrating the comprehensive protocol.

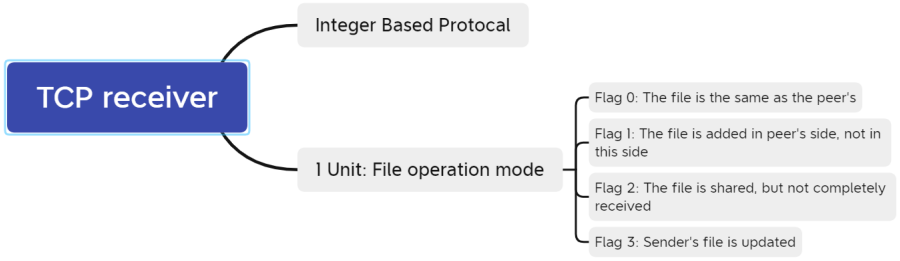

Fig. 1: Protocol Design

2.3 Module Specification

According to the design idea and methodology mentioned above, the application is developed incrementally in each step. In each round, there will be some new features and functions added to the current one. The table as followed will demonstrate all the functions defined in this program. A function marked with [Function] means it only completes one essential task such as getting file block and file size. One marked with [Module] which may perform a composite function is used in thread and the one denoted with [Thread] refers to actual thread running in the program.

Table 1: Important Functions in Program

[Function] argparse()	Run program with arguments
[Thread] send_file(receiver_ip, receiver_port)	Send file by TCP
[Function] scan_file(file_dir)	Traverse the file
[Function] create_file_info(file_name)	Get file information
[Function] create_file_md5(file_name)	Get file md5
[Module] sendFile(file_name, file_size, sender_socket)	Send file by TCP
[Function] get_file_block(file_name, file_size, block_index)	Get each file block
[Thread] receive_file(local_ip, port)	Receive the peer's files
[Function] unpack_file_info(file_info)	Unpack file information
[Module] create_file_flag(file_name, file_mtime, file_md5)	Create the flag to indicate the file
[Thread] write_file(file_name, file_size, file_flag, connection_socket)	Download the files

3 Implementation

3.1 Implementation Steps

According to the function of each module and the description mentioned above, the app is designed via applying multi-threading programming techniques. One thread should be responsible for message receiving and another for file contents sending. To give readers a clear understanding of the structure, an activity diagram is showed as followed.

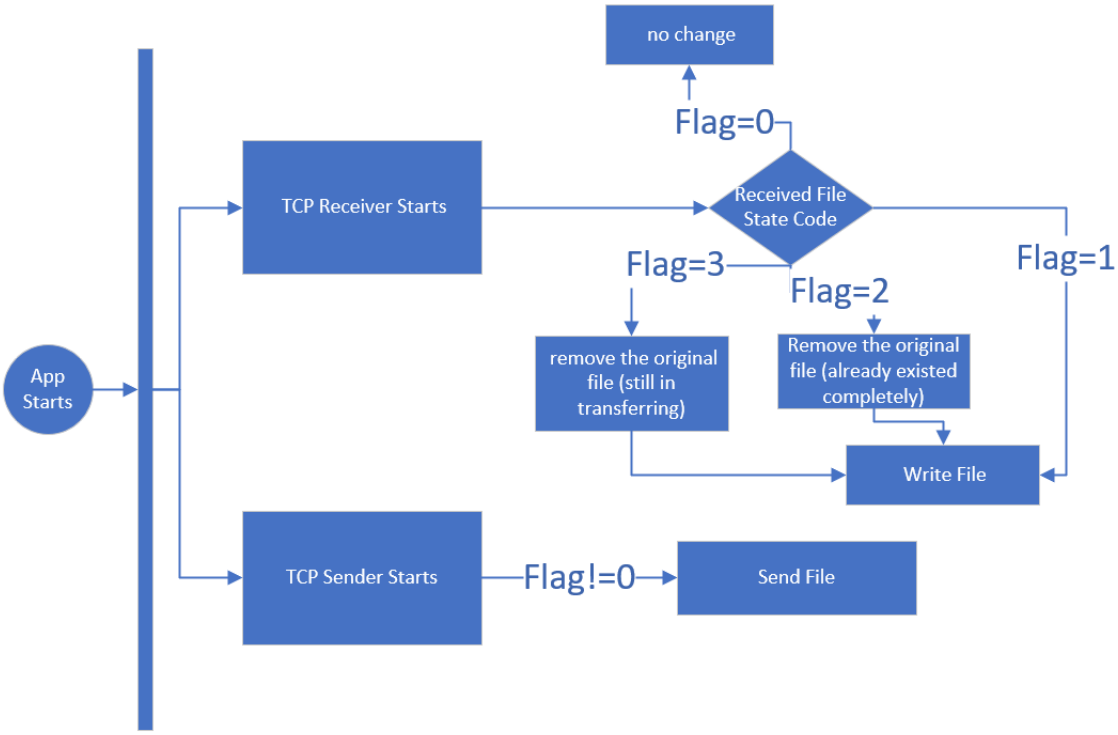

Fig. 2: File Sharing Activity Diagram

In the first place, the thread send_file will run and be ready to listen to another receive_file until they connect successfully. To emphasize, if they lost connection or cannot build connection as the first time, the 'while True' loop will help them to connect again. Secondly, the function scan_file(file_dir) will scan all the files and folders in 'share' directory and and get the list of files' paths. We traverse all the files and get its information by create_file_info(file_name) and send it to receive thread. In the receive thread, it provides a file state flag for further operation. As I demonstrate in 2.2 Protocol Design, different flags indicate different state of file. And the further operation is presented in this activity diagram.

If all the aforementioned steps of transmitting one file are done, the system begins to loop among the list and deal with the next file. After all the files in the list have been looped already, the system will begin to loop and scan the 'share' file again to check if there are some new files or folders added in.

3.2 Programming Skills

In this coursework, one specific programming skill is used to implement certain functions. Since the application should be both client and server and should be workable during its life-cycle, multi-threading technique is needed. When app starts, the main thread will fork

another four separate threads which are responsible for diverse functions. For instance, one thread is used for detecting new files and modifications while another is accountable for communicating with other peers.

3.3 Difficulties and Solutions

Recover from interruption

One specific requirement claims that the application should have the ability to recover from interruption which means all the files should also be synchronized when shared during its downtime. One possible solution is illustrated as followed. When interruption occurs, Thus, flag is expected to solve the problem. Once the program restarted, if the transferred files are cut, the flag will turn into 2, resulting in the deletion of this file. Afterwards, the program will partially send this file again to another side, achieving the synchronization.

Thread synchronization

When a global variable is visited or changed by different threads at the same time, it will result in a great catastrophe. In this program, when a folder and a big file are put into sharing directory simultaneously, it is possible that the big file is transmitted into the folder or folder files are transmitted outside it.

4 Testing and Results

4.1 Testing environment

Two Linux Cores on Oracle VM VirtualBox

4.2 Testing Plans

The testing plan follows marking criteria mentioned in coursework specification strictly. Before testing, we should download two virtual machines, CAN201-CW-TestScript from XJTLU box. Following the steps provided in the your_code (.py), I can run this script for testing the code.

The two peers can be called PC_A and PC_B. The first step is adding File_1(10MB) in PC_A, the PC_B will be executed. Then the 'share' file in PC_B will receive the File_1, and this transmission time is TC_1B. In the next step, File_2 around 500MB and a folder

with 50 small files(1KB) will be added to the 'share' folder in the current working directory of PC_B. Then these files will be synchronized to PC_A, and the transmission time is TC_2A+TC_FA. But in this step, the app on PC_A will be killed after 0.8s finishing the last step, we still need to guarantee the accuracy of these files. The last step is to update the content in File_2 on PC_A randomly. We need to synchronize the content to PC_B and the time consumed in this part is TC_2B.

4.3 Testing Results

There are three lines to indicate the consumed time, the blue one for TC_1B, the red one for TC_2C + TC_FA, the blue one for the TC_2B.

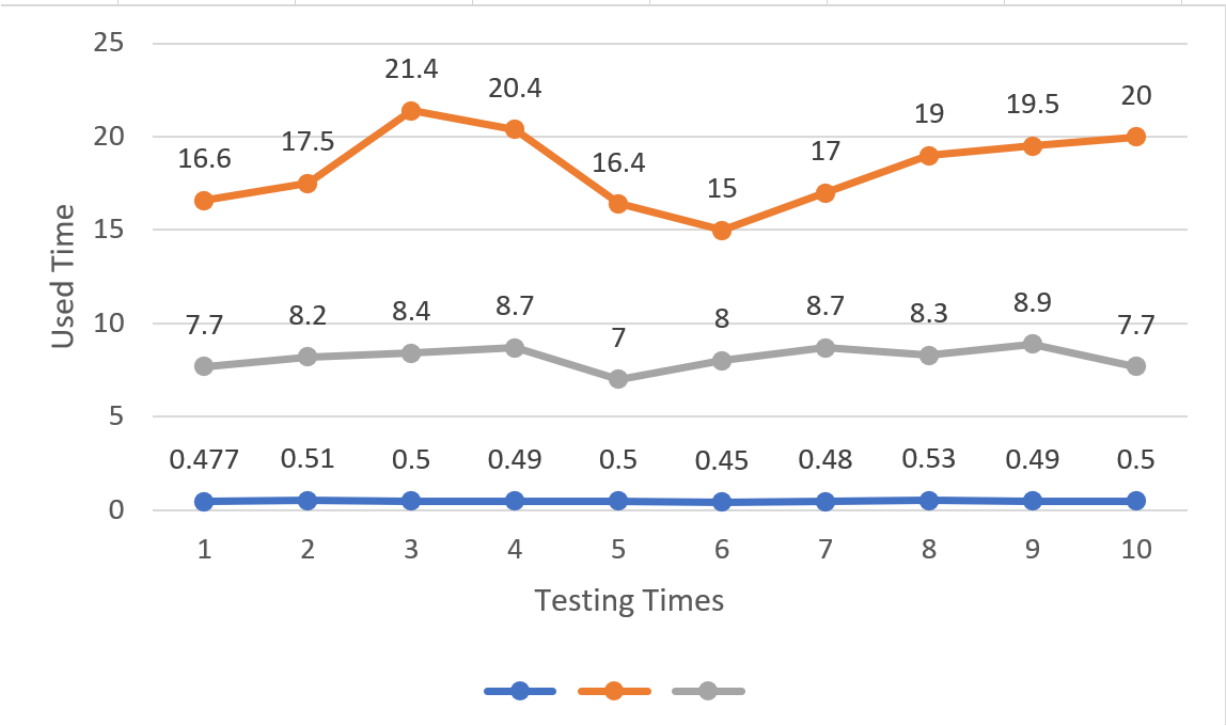

Fig. 3: Testing Diagram

Table 2: Testing Summary Table

	Shortest time	Longest time	Average time
TC_1B	0.477	0.53	0.4927
TC_2C + TC_FA	15	21.4	18.28
TC_2B	7	8.9	8.16

5 Conclusion

To sum up, this report is designed to depict a file sharing application through the design methodology and specific implementation. In implementation, it mentioned a significant programming skill, multi-threading programming. Moreover, several dilemmas in the development process and appropriate solutions were demonstrated.

One underlying limitation in the research is the simplicity of programming, only achieving the basic functionalities. Unlike the complex programs, we cannot prevent the attacks from bad buys. For instance, as we learned in the lecture, the bad guys can put malware into host via the internet, attack servers and network infrastructure, sniff packets, masquerade as someone you trust. Thus, we should seek defenses against sniffing, end-point masquerading, man-in-the-middle attacks, DDos attacks, malware in the future research.

Reference

[1] f. Cheng, "Can201 introduction to networking coursework 1 specification," 2021.

[2] S. Lila, E. Femmy, and S. Annisa Anggraini, "Using technology acceptance model 3 (tam 3) at selected private technical high school: Google drive storage in e-learning." *Utamax*, vol. 3, no. 2, pp. 80 – 89, 2021. [Online]. Available: http://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsdoj&AN=edsdoj.78fd93af09034cab9af1e6faec17398d&site=eds-live&scope=site

[3] A. Sadik, "Students' acceptance of file sharing systems as a tool for sharing course materials: The case of google drive." *Education and Information Technologies: The Official Journal of the IFIP Technical Committee on Education*, vol. 22, no. 5, p. 2455, 2017. [Online]. Available: http://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edssjs&AN=edssjs.63B27F7&site=eds-live&scope=site

[4] S. Li, W. Sun, and J. Liu, "A mechanism of bandwidth allocation for peer-to-peer file-sharing networks via particle swarm optimization." *Journal of Intelligent Fuzzy Systems*, vol. 35, no. 2, pp. 2269 – 2280, 2018. [Online]. Available: http://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=131542950&site=eds-live&scope=site

[5] S. Gandeva Bayu, "Digital forensics study of a cloud storage client: A dropbox artifact analysis." *CommIT Journal*, vol. 13, no. 2, 2019. [Online]. Available: http://login.ez.xjtlu.edu.cn/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsdoj&AN=edsdoj.2fd619d4a0c749c58a61174bbbf85055&site=eds-live&scope=site