Name: Wang.Qi

ID: 1929518

Content

1. Abstract

File sharing plays an important role in the process of network data transmission. Efficiency, flexibility and trust are three important dimensions to examine the quality of the transmission. This report will illustrate a proposed protocol based on TCP, one implementation process of a large efficient flexible and trusty file-sharing program and thoughts of further improvement methods.

2. Introduction

2.1 Background

Large-scale data transformation has been applied to various living scenarios. For example, Dropbox, iCloud, Google Drive are some popular file transmission tools in recent years. They receive much attention on account of their huge effect in the fourth industrial revolution and the emerging Internet of Things(IoT)[1]. In this project, Using python socket network programming to build a similar small applying scenario is a direct way to learn this technology.

2.2 Project requirement

The main purpose of this project is to achieve large file synchronization between two devices. Establishing a connection based on TCP between two devices is the first step. Also, they should continuously connect without suddenly disruption. When files or folders with different sizes and types are added under the “share” folder, one device can send the files or folders to the other device efficiently and correctly.

There are two other situations if it occurs to one of the devices stop running, it can reconnect automatically and ensure the integrity of transmitting data. Moreover, if one file is modified, this device should find it timely and resend it to the other end to achieve file synchronization.

3. Methodology

3.1 Proposed functions

Following is a brief description of each function:

Function	**Description**
_argparse()	Parse parameters
Make_header()	Pack op_code, modified time, file size and file name to transmit
Parse_header(header)	Unpack related file information to assign values
Create_folder()	Check if it has a “share” folder or create one
Scan_folder()	Scan its own files in “share” folder
Compress_file(folder_name)	Compress the folder into a zip file
Decompress_file(folder_name)	Decompress the zip file into a folder
File_synchoronization()	Exchange files between two deivices
Client_send_msg(msg,port)	Send file information to the other end
Send_file(file,port)	Send specific file to the other end
Send_folder(folder_name,port)	Send the folder to the other end
Recv_file(name, rec_file_socket,mtime)	Receive specific file from the other end
Receive_folder(fd_name, r_file_socket,mtime)	Receive specific folder from the other end
Server_recv_msg(total_msg,rec_socket)	Receive related information message from the other end

3.2 Proposed protocols and ideas

The general idea of the proposed protocol is to prepare two dictionaries to save their size and the modified time of synchronized files respectively. Comparing latest scanning results with dictionary results to decide whether the file or folder should be sent to the other device or not. Based on TCP, the byte stream can be sent to the other device reliably without worrying about package loss or disorder. If one synchronization process finishes, record the synchronized file and continuously scan the “share” folder.

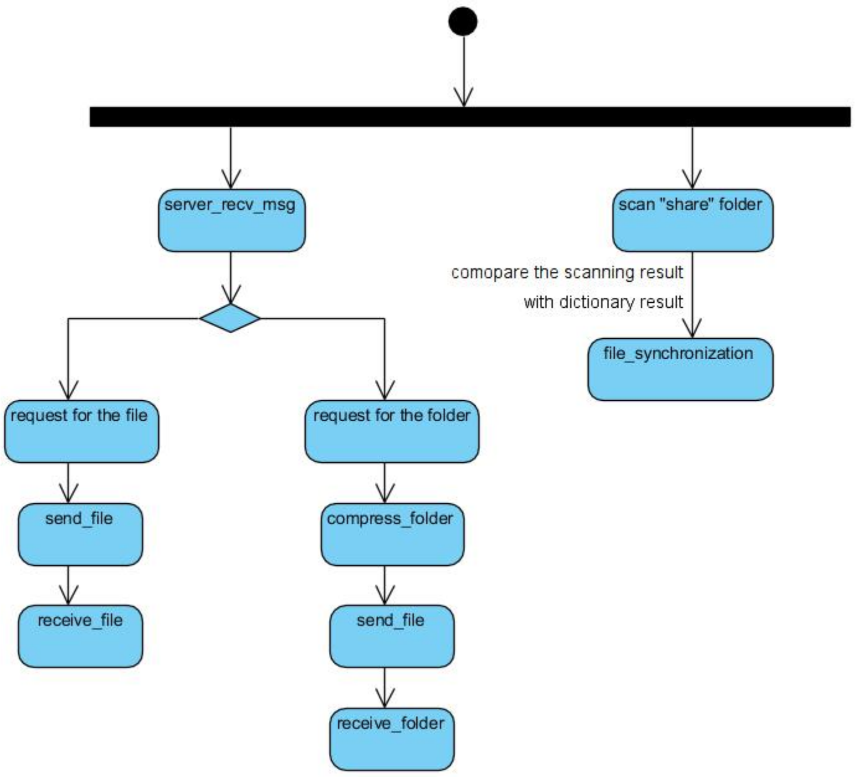


Figure 1: a brief FSM of protocol design

In this program, two threads are used to implement different functions. The main thread is responsible for scanning its own "share" folder and determining which file to send according to the comparison results of the known record and current file directory. Also, once an unrecorded or modified file is found, it will stop scanning. The receiving message thread acts as a sending and receiving station to read and write files and ensure the integrity of files. After confirmation receiving, it will write its current file information to a global dictionary to offer the convenience of further checking.

Different actions depend on different operation codes. Before sending and receiving the file, both devices will tell with each other one corresponding operation. Following is a table of meanings of each op_code:

op_code	**meaning**
0	Receive header message
1	Check and receive the file

2	Request for sending the file
3	Received the file / folder successfully
6	Check and receive the folder
7	Request for sending the folder
8	Retransmit the modified file
10	Start scanning

4. Implementation

4.1 Steps of implementation

1. Two devices scan their own “share” folder respectively.

2. Compare the latest scanning result with the known dictionaries to determine if each file is synchronized or not.

3. Send header information of the unrecorded file to the other device.

4. Stop scanning with changing the value of op_code

These steps above are done in the main thread. Here is an FSM diagram to show the process in detail:

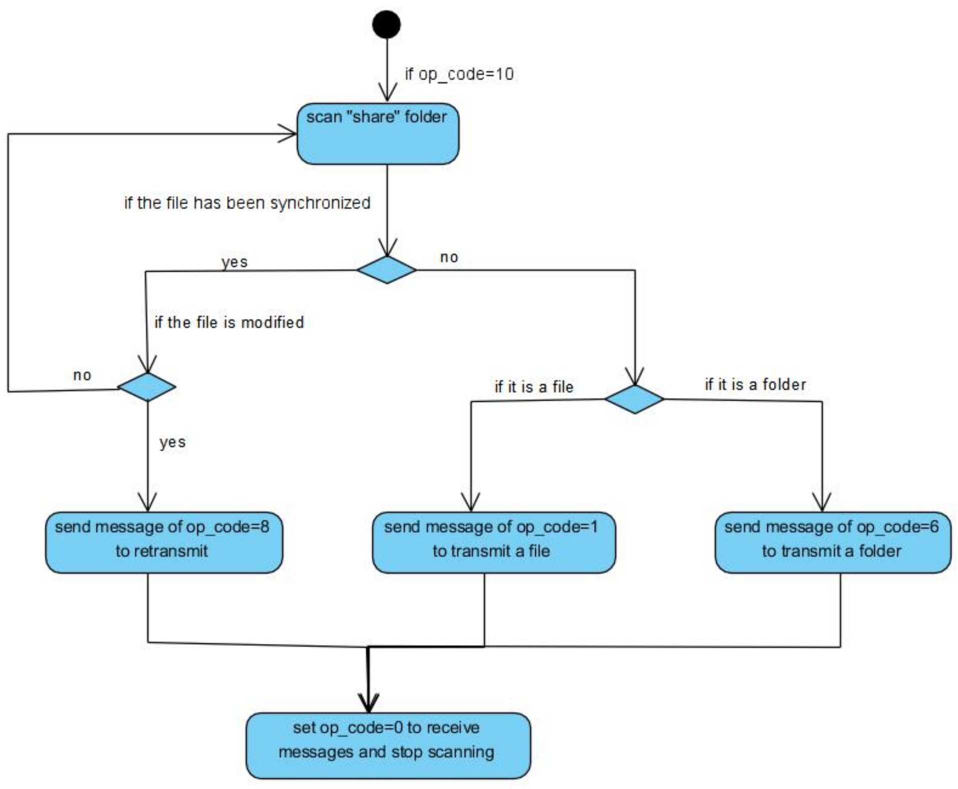

if op_code=10
scan "share" folder
if the file has been synchronized
yes
no
if the file is modified
no
yes
if it is a file
if it is a folder
send message of op_code=8 to retransmit
send message of op_code=1 to transmit a file
send message of op_code=6 to transmit a folder
set op_code=0 to receive messages and stop scanning

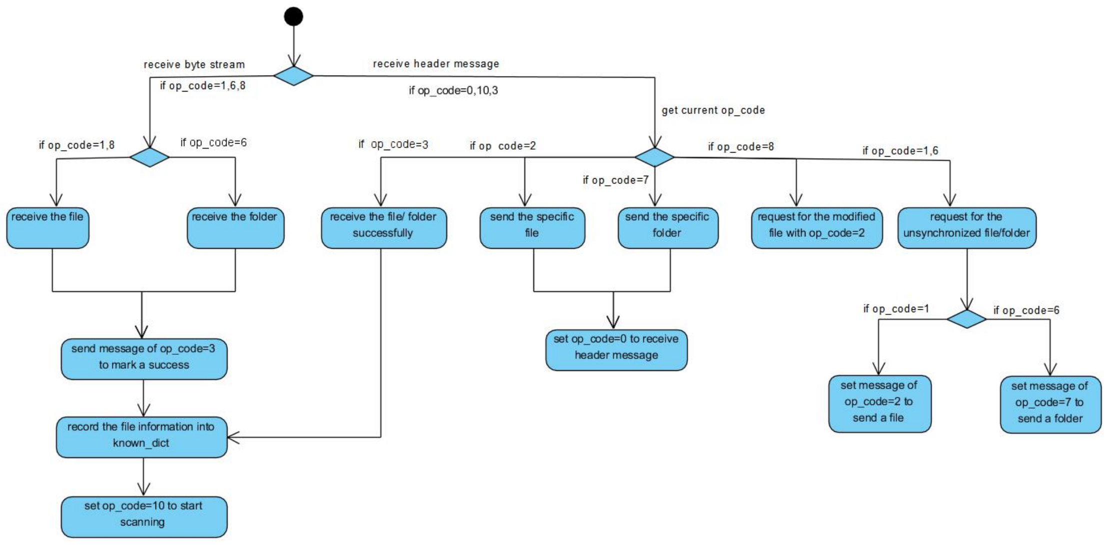
Figure 2: FSM of main thread

5. When one device receives the header message, it prepares to receive and send a "request" message to the other side according to the value of op_code,

6. When the other device successfully write it and send the corresponding message back, both devices record the size and modified time of this file

7. If one file is changed by comparing the latest modified time with the recorded modified time, retransmit this file to the other end.

In the thread of receiving and sending, receiving the header message and taking the corresponding operation depending on different op_code are two general processes. The following FSM diagram of this thread will show the process in detail:



receive byte stream
if op_code=1,6,8
receive header message
if op_code=0,10,3
get current op_code
if op_code=1,8
if op_code=6
if op_code=3
if op code=2
if op_code=8
if op_code=1,6
if op_code=7
receive the file
receive the folder
receive the file/ folder successfully
send the specific file
send the specific folder
request for the modified file with op_code=2
request for the unsynchronized file/folder
send message of op_code=3 to mark a success
set op_code=0 to receive header message
if op_code=1
if op_code=6
set message of op_code=2 to send a file
set message of op_code=7 to send a folder
record the file information into known_dict
set op_code=10 to start scanning

Figure 3: FSM of receiving and sending

4.2 Programming skills

Multithreading

Two threads used here take charge of different functions. When the program starts running, they will both start with a tiny time interval. To achieve communication between two threads, setting several global variables is necessary to avoid ambiguity.

4.3 Difficulies encountered and solutions

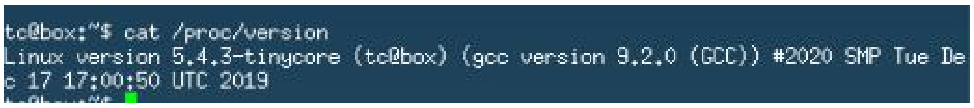
One difficulty is designing an efficient mode to detect which file should be shared. The first idea is that both two devices send their scanning results by JSON with each other. Then, compare two results and decide which to send. However, modified time won't be equal in two devices because of transmission time. Hence, it may result in infinite retransmission.

The new model, "scanning itself and comparing this result with the last version", can solve most of these questions with dictionary data structures to store the synchronized file information. However, it is still not a perfect method because when one device is out of connection, reconnection means retransmitting all known files due to the empty known dictionary. Under this assignment requirement, it does not affect much. If there is a situation with multiple files, this will cause a waste of time.

5. Testing and results

5.1 Testing environement

Two testing devices are replaced with two virtual machines with Linux operating system in one computer. The version of these two virtual machines is:



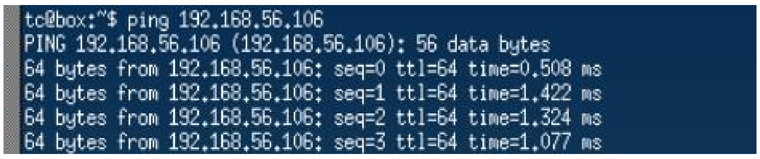
```
tc@box:~$ cat /proc/version
Linux version 5.4.3-tinycore (tc@box) (gcc version 9.2.0 (GCC)) #2020 SMP Tue De
c 17 17:00:50 UTC 2019
```

Figure 4: Linux version in the virtual machine

5.2 Testing plan of the best interval time

5.2.1 Pre-test

1. Ensuring the network connection between two virtual machine by Ping command:



```
tc@box:~$ ping 192.168.56.106
PING 192.168.56.106 (192.168.56.106): 56 data bytes
64 bytes from 192.168.56.106: seq=0 ttl=64 time=0.508 ms
64 bytes from 192.168.56.106: seq=1 ttl=64 time=1.422 ms
64 bytes from 192.168.56.106: seq=2 ttl=64 time=1.324 ms
64 bytes from 192.168.56.106: seq=3 ttl=64 time=1.077 ms
```

Figure 5: Testing result of ping

2. Using python python command to parse parameter of Ip address and start running: Python3 main.py --ip 192.168.56.106

5.2.2 Testing process

Different interval times may influence the performance of the program. Hence, we calculate the time consumption of each time by setting different values of “time. sleep(n)”. In the testing process:

1, No other programs will be running simultaneously

2, The value of time-consuming will be an average number of multiple tests(>5)

3, All tests are completed in the same period, which means the performance of the laptop can be the same

5.3 Testing results and analysis

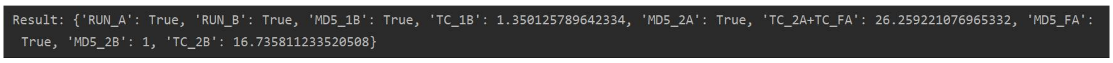
The testing result is that the best sleeping time should set around 0 to 1.

Here is a table with the results of the time spent：

Time	**TC_1B**	**TC_2A+TC_FA**	**TC_2B**
0	1-2s	Around 20s	Around 20s
0.5	Around 0.6s	Around 10s	Around 10s
1	Around 1s	Around 15s	Around 15s
2	Around 2s	Around 15s	Around 15s

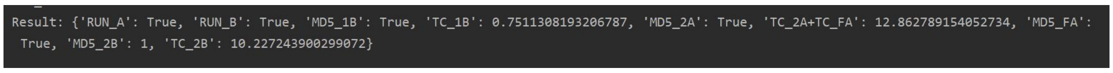
If the sleeping time is much more than the killed interval, there exists some time wasting espacilly in TC_2A becuase it would detect the newly added file after the other device restarted. However, if setting the sleeping time to zero, the efficiency will decreased instead. That is because the file should be detected before completely added. Also, the uncompleted file in phase 2 will be abandoned in this protocol. It has to retransmit all uncompleted file. Therefore, it is not a best value of sleeping time.

The screenshot of one of results of setting interval time to 0 seconds:



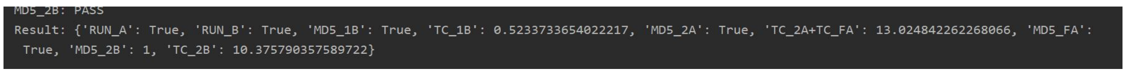
```
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 1.350125789642334, 'MD5_2A': True, 'TC_2A+TC_FA': 26.25922107696532, 'MD5_FA':
 True, 'MD5_2B': 1, 'TC_2B': 16.735811233520508}
```

0.5 seconds:



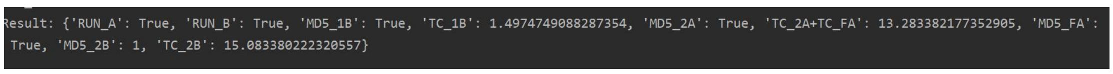
```
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.7511308193206787, 'MD5_2A': True, 'TC_2A+TC_FA': 12.862789154052734, 'MD5_FA':
 True, 'MD5_2B': 1, 'TC_2B': 10.227243900299072}
```

1 seconds:



```
MD5_2B: PASS
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.5233733654022217, 'MD5_2A': True, 'TC_2A+TC_FA': 13.024842262268066, 'MD5_FA':
 True, 'MD5_2B': 1, 'TC_2B': 10.375790357589722}
```

2 seconds:



```
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 1.4974749088287354, 'MD5_2A': True, 'TC_2A+TC_FA': 13.283382177352905, 'MD5_FA':
 True, 'MD5_2B': 1, 'TC_2B': 15.083380222320557}
```

6. Conclusion

This assignment can be regarded as a function model of file sharing applications to achieve relatively large, efficient, flexible and trusty file transmission between two devices. In this process, having a deeper understanding of TCP, designing an application protocol, solving problems of breakpoint and retransmission are all beneficial ways of learning computer networks. Furthermore, the interval time should be set in a reasonable range to achieve best performance after testing. Also, What a student should learn is persistent learning and thinking and the capacity of handling different problems.

6.1 Future plan

1. Multithreading used here doesn't do parallel execution, so to further improve the efficiency, a better designing protocol should fully utilise threads.
2. The security of transmitting files and messages is not guaranteed. Therefore, adding some measures of file encryption transmission maybe help.

7. Reference

[1] K. G. Tsiknas, P. I. Aidinidis, and K. E. Zoiros, "Performance evaluation of transport protocols in cloud data center networks,", Photonic Network Communications, vol. 42, no. 2, pp. 105-116, Oct 2021.