Large Efficient Flexible and Trusty (LEFT) Files Sharing program

Qijing Zeng

1930702

Word count:1769

1. Abstract

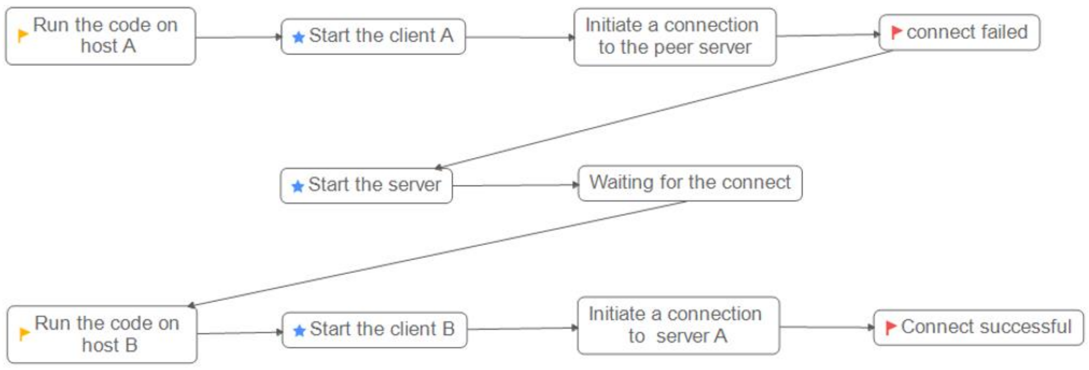
This paper proposes two customized end-to-end file sharing protocols. Through the implementation and testing of the two methods, it is found that both methods can ensure the integrity of file transfer. After comparing the test results of the two methods, a more efficient file sharing method is proposed in which the client requests all files from the server every 3s when a single file is less than 500MB.

2. Introduction

As end-to-end file sharing is used more and more, the credibility and transmission efficiency of file sharing are becoming more and more important to people. There are many ways to improve the efficiency of end-to-end file sharing. For example, a study (Guoxin, et al. 2015) proposed an algorithm for improving file search efficiency in file sharing in a cluster. However, it is easy to overlook the rate of file sharing for a single file that does not exceed 500MB. Aiming at this problem, this paper proposes two customized protocols to realize the Large Efficient Flexible and Trusty (LEFT) Files Sharing program when a single file is less than 500MB. The following is through the implementation and testing of the two protocols to compare the most efficient method.

3. Methodology

This part will first introduce how to establish a connection between two hosts, and then explain the general logic of the two methods to achieve file synchronization.
The logic diagram for establishing a connection between two hosts A and B are as follows.



Method 1: After the connection is established, the client of host B requests all files from the server of host A, the server of host A sends all the files, and the client of host B closes the connection after receiving all the files. By setting the timer every 3s, the client and server of the two hosts exchange roles and repeat the above operation. For instance, after 3s, the client of host A actively connects to the server of host B, repeats operations such as the client requesting a file from the server, and the client closing the connection after receiving it. After another 3s, the client of host B actively connects to the server of host A, and then repeats the above operation.
Method 2: After the connection is established, the client actively sends its own folder dictionary to the server. The key of the dictionary is the file name in the folder, and the value is the hash value corresponding to each file. The server also generates its own folder dictionary, then compares the differences between the two dictionaries, identifies missing or different files from the client, and sends them to the client. The

client closes the connection after receiving the file, and the host AB exchanges roles after every 3s and repeats the above operation.

4. Implementation

4.1 Implementation steps

a) divide the function to be implemented into partial functions.

b) integrate partial functions into the main code and call the corresponding functions according to the overall process.

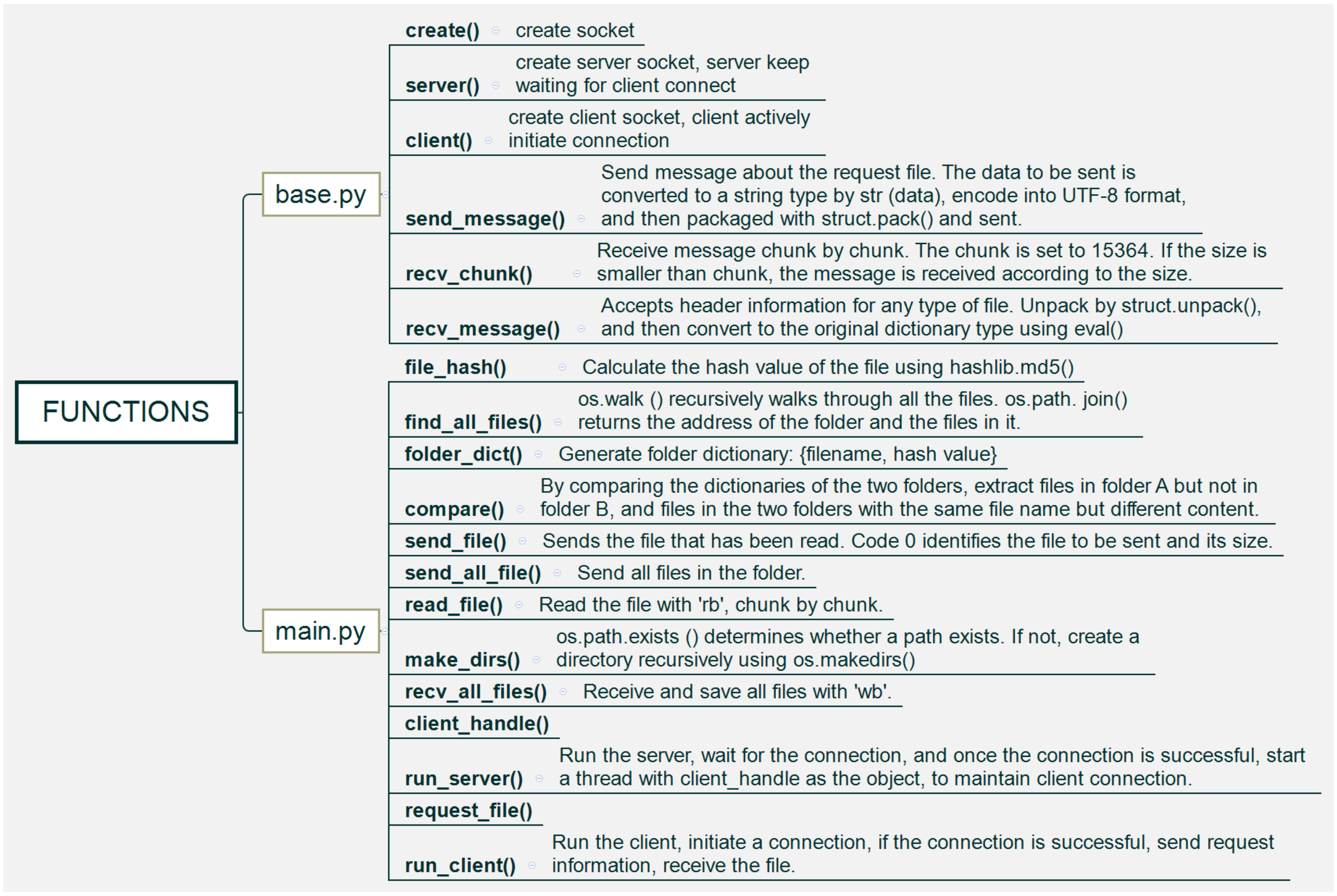
c) test.

4.2 Python modules used

os, sys, socket, struct, hashlib, threading, time

4.3 Functions

Functions are defined to realize partial functions. The function name and description are shown in the figure below.



create() create socket

server() create server socket, server keep waiting for client connect

client() create client socket, client actively initiate connection

send_message() Send message about the request file. The data to be sent is converted to a string type by str (data), encode into UTF-8 format, and then packaged with struct.pack() and sent.

recv_chunk() Receive message chunk by chunk. The chunk is set to 15364. If the size is smaller than chunk, the message is received according to the size.

recv_message() Accepts header information for any type of file. Unpack by struct.unpack(), and then convert to the original dictionary type using eval()

file_hash() Calculate the hash value of the file using hashlib.md5()

find_all_files() os.walk () recursively walks through all the files. os.path. join() returns the address of the folder and the files in it.

folder_dict() Generate folder dictionary: {filename, hash value}

compare() By comparing the dictionaries of the two folders, extract files in folder A but not in folder B, and files in the two folders with the same file name but different content.

send_file() Sends the file that has been read. Code 0 identifies the file to be sent and its size.

send_all_file() Send all files in the folder.

read_file() Read the file with 'rb', chunk by chunk.

make_dirs() os.path.exists () determines whether a path exists. If not, create a directory recursively using os.makedirs()

recv_all_files() Receive and save all files with 'wb'.

client_handle()

run_server() Run the server, wait for the connection, and once the connection is successful, start a thread with client_handle as the object, to maintain client connection.

request_file()

run_client() Run the client, initiate a connection, if the connection is successful, send request information, receive the file.

4.4 Different functions of the two methods

The difference between method 1 and method 2 is the client_handle()function. The first method:

```python
def client_handle(client, address):
    print("client", address)
    server_folder_dict = None
    try:
        while True:
            # accept the request message
            message = base.recv_message(client)
            if message['code'] == 1:
      # After the connection is established, the server sends all files to the client
                send_all_file(client, find_all_files("share/"))
            elif message['code'] == 2:
                client_folder_dict = message['content']
                server_folder_dict = folder_dict(find_all_files("share/"))
                new_info = compare(server_folder_dict, client_folder_dict)
                send_all_file(client, new_info.keys())
            base.send_message(client, {'code': -1})
            break
    except OSError as e:
        print(e)
    else:
        if server_folder_dict is not None:
            info = {"code": 2, "content": server_folder_dict}
        else:
            info = {"code": 1}
        print("A request file", info)
        threading.Timer(3, run_client, args=(args.ip, info)).start()
    finally:
        client.close()
```

The second method:

```
def client_handle(client, address):
    print("client", address)
    server_folder_dict = None
    try:
        while True:
            message = base.recv_message(client)  # accept the request message
            if message['code'] == 1:
                send_all_file(client, find_all_files("share/"))  # send all files
            if message['code'] == 2:
                client_folder_dict = message['content']
# a dictionary of received client folder information
                server_folder_dict = folder_dict(find_all_files("share/"))  # generate a server folder dictionary
                new_info = compare(server_folder_dict, client_folder_dict)  # comparing folder differences
```

```python
                send_all_file(client, new_info.keys())  # server sends the extracted file to the client after comparison
            base.send_message(client, {'code':-1})  # ending file Transfer
            break
    except OSError as e:
        print(e)
    else:
        if server_folder_dict is None:
            server_folder_dict = folder_dict(find_all_files("share/"))
            info = {"code": 2, "content": server_folder_dict}
        else:
            info = {"code": 2, "content": folder_dict(find_all_files("share/"))}
        print("A request file", info)
        threading.Timer(3, run_client, args=(args.ip, info)).start()
        # The timer is set to 3s
    finally:
        client.close()  # the client closes the connection
```

4.5 Thread used

Threads are more lightweight than processes and are used in this study to handle concurrency.

a) After the server and the client are successfully connected, the server runs a thread to maintain the connection. This thread target is client_handle.

```python
def run_server():
    sock = base.server()
    print("start my server")
    while True:
        print("start wait client connect...")
        client, address = sock.accept()
        print('connect success')
        threading.Thread(target=client_handle, args=(client, address)).start()
```

b) The thread cyclic task timer is used in client_handle().

```python
threading.Timer(3, run_client, args=(args.ip, info)).start()
# The timer is set to 3s
```

4.6 Problems encountered and solutions

Question1: It is easy to confuse the content when receiving different information and notifications

Solution1: Define different codes to represent different types of message headers.'-1'means the acceptance is complete and the acceptance is stopped. '0'means sending a file.'1' means executing the first method.'2'represents the execution of the second method. The specific example in code as shown below.

```python
def recv_all_files(sock):
    while True:
        message = base.recv_message(sock)
        print("message:", message)
        if message['code'] == -1:
            print("stop")
            break
```

Question 2: The timer time in client_handle() is set too short， so that when a large file is added to host B during the test, it will be sent to another host before the addition is completed, and the wrong file is sent, so an error is reported.

Solution 2: In the case that other variables remain unchanged, set a different time, test the results repeatedly, and finally choose to set the time parameter to 3 to achieve faster file sharing on a stable basis.

5. Testing

5.1 Test environment:

Create two VMS of the same specifications on Oracle VM VirtualBox. Hosts A and B are referred to below. Each virtual machine has 512MB of memory and 8GB of virtual disks. Tiny Core Linux is used.

5.2 Test Plan

For the P2P file-sharing protocol implemented in this paper, the main test purpose is to detect when two hosts run this protocol, add and modify files in the folder of one host, these updated files can be completely and quickly shared to another host. The md5 value is used to determine whether the file is completely synchronized to another host. The faster method can be obtained by comparing the transfer time. Run the virtual machine first, find the IP of the two virtual machines, and write them into the test code. Then run a test code in Pycharm. The test code detection steps are as follows:

a) Run the code on host A

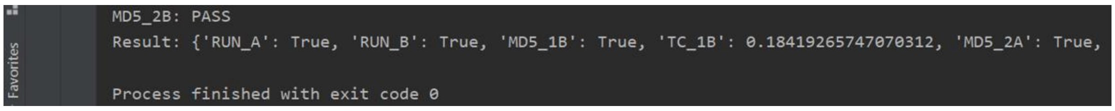
b) Add a 10MB file to the share folder in the current path of PC_A. Run the code on host B. The md5 value of the calculation file is recorded as MD5_1B. Record file transfer time is TC_1B

c) Add a 500MB File2 and a folder containing 50 files (1KB) to PC_B. Record the md5 value of File2 as MD5_2A. Kill PC_A and restart, record the time when PC_B transfers File2 to PC_A as TC_2A, and record the folder transfer time as TC_FA.

d) Change at least 10% of the bytes of File2 in PC_A, and record the transmission time for PC_A to transfer the changed File2 to PC_B and record as TC_2B.

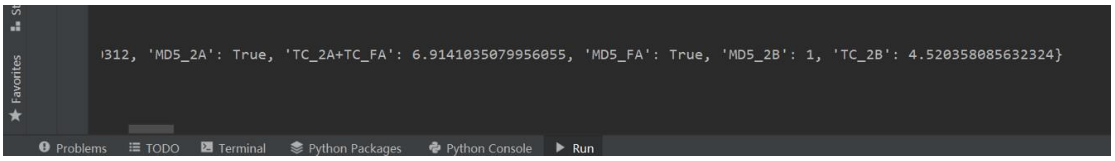
5.3 Test outcome

The test result of the first method is as follows.



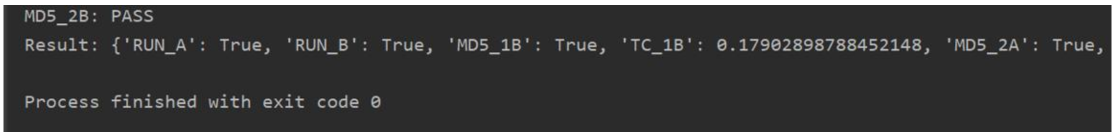
```
MD5_2B: PASS
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.18419265747070312, 'MD5_2A': True,

Process finished with exit code 0
```



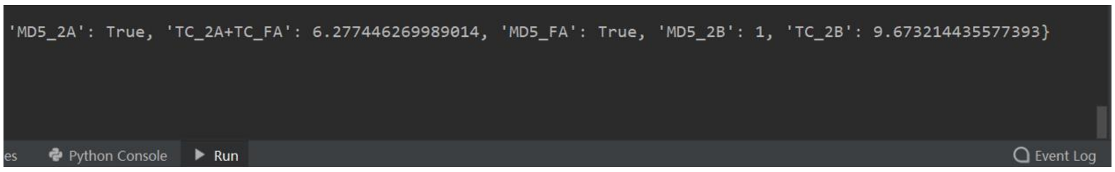
```
312, 'MD5_2A': True, 'TC_2A+TC_FA': 6.9141035079956055, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 4.520358085632324}
```

The test result of the second method is as follows.



```
MD5_2B: PASS
Result: {'RUN_A': True, 'RUN_B': True, 'MD5_1B': True, 'TC_1B': 0.17902898788452148, 'MD5_2A': True,

Process finished with exit code 0
```



```
'MD5_2A': True, 'TC_2A+TC_FA': 6.277446269989014, 'MD5_FA': True, 'MD5_2B': 1, 'TC_2B': 9.673214435577393}
```

In order to avoid the instability of the test environment caused by hardware configuration, and to enhance the universality of the test results, the two methods were tested four times respectively.
The result was shown in the table.

	Method	MD5_1B	MD5_2A	MD5_FA	MD5_2B	TC_1B	TC_2A+TC_FA	TC_2B	Total_time
1	**1**	TRUE	TRUE	TRUE	1	0.1842	6.9141	4.5204	11.6187
2	**2**	TRUE	TRUE	TRUE	1	0.179	6.2774	9.6732	16.1296
3	**1**	TRUE	TRUE	TRUE	1	0.1784	6.3408	4.4398	10.959
4	**2**	TRUE	TRUE	TRUE	1	0.1794	6.634	8.9293	15.7427
5	**1**	TRUE	TRUE	TRUE	1	0.1772	5.6636	4.8445	10.6853
6	**2**	TRUE	TRUE	TRUE	1	0.1858	7.2755	11.6156	19.0769
7	**1**	TRUE	TRUE	TRUE	1	0.1792	6.0471	4.4565	10.6828
8	**2**	TRUE	TRUE	TRUE	1	0.1831	7.4074	9.6599	17.2504

Add up the three transmission times of each test record and record it as Total_time. Test data shows that implementing the same test plan on the same hardware facility for both methods can achieve complete file sharing. But the total transmission time of method 2 is 1.56 times the sum of the time of method 1 on average.

6. Conclusion

End-to-end file sharing is increasingly used in daily life. In the case of ensuring the integrity of file transfers, the requirements for file sharing transfer speeds of 500MB and below for a single file are getting higher and higher. This article introduces two methods for implementing the Large Efficient Flexible and Trusty Files Sharing program. By testing the protocols of the two methods and analyzing the results, it is found that the update rate of method one is faster on file sharing. In the end-to-end file sharing requirement that a single file does not exceed 500MB, method 1 of sending all files to the other client every 3s can achieve a higher file synchronization rate. However, this result has limitations, because in internal network tests, the network transfers files faster, and it takes more time to calculate the file hash value. If the network delay is high and the file transfer is slow, the second method may have the higher efficiency, which is used to compare the file differences by calculating the hash value, and then transfer partial files.

7. Reference

Guoxin Liu, Haiying Shen and Ward, L. (2015) ‘An Efficient and Trustworthy P2P and Social Network Integrated File Sharing System’ . Available from:
https://ieeexplore-ieee-org.ez.xjtlu.edu.cn/stamp/stamp.jsp?arnumber=6624110
[Accessed: 15 December 2021]