

# Large Efficient Flexible and Trust Files Sharing

## Abstract:

The applications based on the file transfer system have been developed completely in the current time. This coursework is aimed at designing and implementing a Large Efficient Flexible and Trust Files Sharing (LEFT) system, which has the ability to transfer all format files between peers. Although, there are many difficulties in this task, such as multiprocessing, blocks transfer and file security in this task, most of them could be overcome by strong python language and useful modules

## Content:

Introduction-----	1
Methodology-----	2
Implementation-----	2
Testing and results-----	3
Conclusion-----	6

## Introduction:

In order to transfer a large file efficiently and trustily, the TCP protocol was applied in this application. As for the considering of flexible share, a simple peer-to-peer architecture was used on this system, and it also helps to crease the transfer efficiency. In the current program, many foundational and required functions have been implemented, but the file encryption test was failed in the testing for some unknow problems. In this report, the structure of the code and functions used will be discussed and the specific testing process will be given, which including the process of encryption test.

## Methodology:

In the code, there are four important function *scanner()*, *listener()*, *downloader()* and

*Say\_Hello()*, and the first three functions will running in three subprocess respectively. As for *Say\_Hello()*, it is used to connect with other peers when the program start. In the subprocess, *cannner()* is responsible to scan the new file in the 'share' folder. *Listener()* is used to listen to the new connection, when a new connection from other peers is requested, the listener put the connection to a new sub connection socket. In terms of *downloader()*, it download the files from other peers.

### **Implementation:**

To implement the above structure, three process sharing dictionary list are need. The first one is used to store the local files information which is named *local\_dict* and the second one is named *open\_dict* because it is used to store the information of the files from others, which is open to peers and they could compare this dictionary list with each other. The last one is *ticket*, and it stores the information of files needed to download.

When the program starts to run, it will check the current directory, if there are no files 'local\_dict.log', 'open\_dict.log' and folder 'share', it will set them automatedly. If they have existed, it will read them store the data to the *local\_dict* and *open\_dict*.

Then, the *Say\_Hello()* will run before the subprocess, in this function, the host will connect with other peers. If successfully with one peer, they will exchange their *open\_dict*, if not, code will run in the three subprocesses.

- **Scanner:**

In an infinite loop, it scans the current working directory, if any new files, it sends the new file information to other peers.

- **Listener:**

In an infinite loop, there is a socket listening the requested connection. If it accepts a request, there is a new socket connect with the requester. Their communication will be allocated in a new thread.

In the sub thread, the connection socket will receive three kinds of sign, 'hello', 'news' and 'get\_file'.

1. hello: Check the new files in the file list from the peer, if there are new files, make ticket and store it. Then, send the *open\_dict* back.
2. news: Check the new files in the file list from the peer, if there are new files, make ticket and store it.
3. get\_file: Check *open\_dict*, if there are the requested files, send the files the peer.

- Downloader:

In an infinite loop, it checks if there are tickets in the *ticket*, and sends the information about the first file in the *ticket* to the peer. Then, it waits for receiving the file.

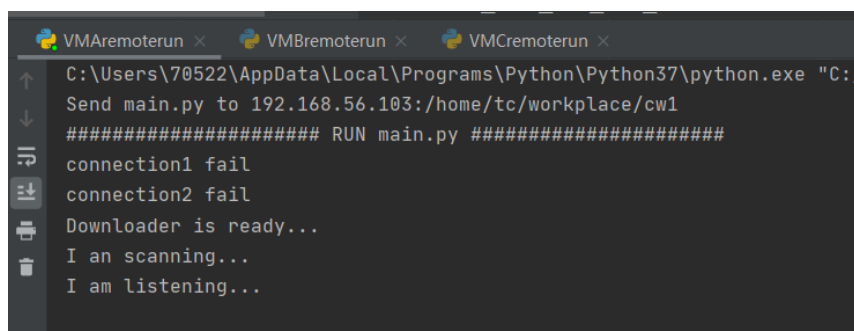
It saves the received data in the file (xx,xxx.lefting) firstly. When receiving work finished, it renames this file, update the *local\_dict* and *open\_dict* and remove this ticket.

## Testing and results:

In the testing process, there are three virtual machines (VMA, VMB, VMC). There are the specific testing steps:

1. Action: Run VMA.

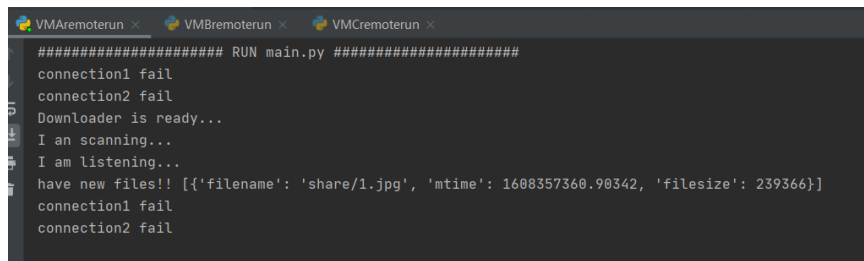
Result: After attempting to connect with other peers (failed), it successfully ran in three processes.



```
VMAreoterun x VMBreoterun x VMCreoterun x
C:\Users\70522\AppData\Local\Programs\Python\Python37\python.exe "C:/
Send main.py to 192.168.56.103:/home/tc/workplace/cw1
##### RUN main.py #####
connection1 fail
connection2 fail
Downloader is ready...
I an scanning...
I am listening...
```

2. Action: Add a file (233kb) in the 'share' folder of VMA.

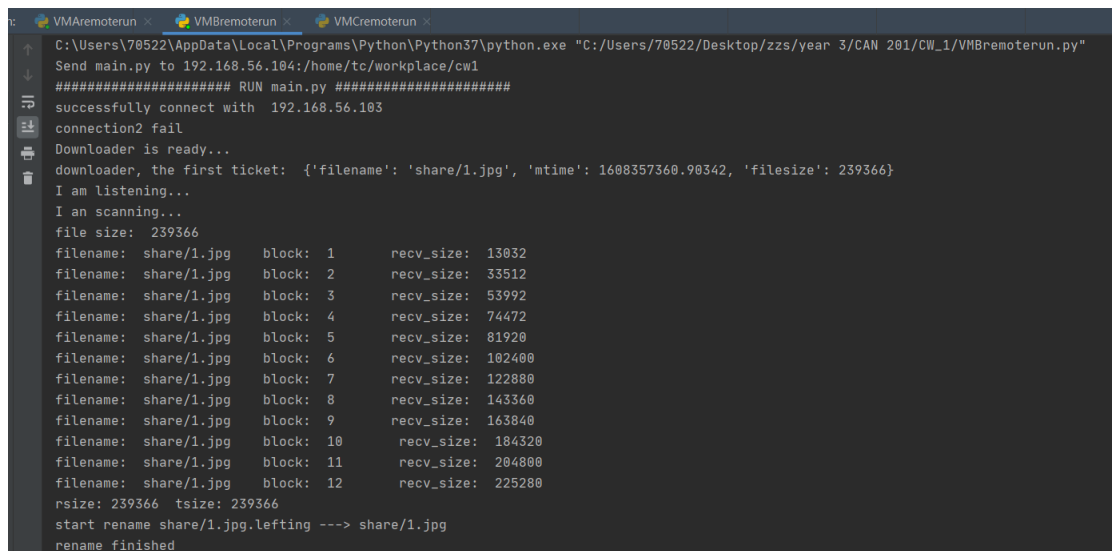
Result: The scanner found the new file and attempted to tell other peers (failed). Then, it updated the *open\_dict* and *local\_dict*.



```
##### RUN main.py #####
connection1 fail
connection2 fail
Downloader is ready...
I am scanning...
I am listening...
have new files!! [{'filename': 'share/1.jpg', 'mtime': 1608357360.90342, 'filesize': 239366}]
connection1 fail
connection2 fail
```

### 3. Action: Run VMB and VMC.

Result: VMB and VMC connected with each other and VMA, they exchanged their *open\_dict* and file new files. Their downloader downloaded the files successfully from VMA



```
C:\Users\70522\AppData\Local\Programs\Python\Python37\python.exe "C:/Users/70522/Desktop/zs/year 3/CAN 201/CW_1/VMBremoterun.py"
Send main.py to 192.168.56.104:/home/tc/workplace/cw1
##### RUN main.py #####
successfully connect with 192.168.56.103
connection2 fail
Downloader is ready...
downloader, the first ticket: {'filename': 'share/1.jpg', 'mtime': 1608357360.90342, 'filesize': 239366}
I am listening...
I am scanning...
file size: 239366
filename: share/1.jpg block: 1 recv_size: 13032
filename: share/1.jpg block: 2 recv_size: 33512
filename: share/1.jpg block: 3 recv_size: 53992
filename: share/1.jpg block: 4 recv_size: 74472
filename: share/1.jpg block: 5 recv_size: 81920
filename: share/1.jpg block: 6 recv_size: 102400
filename: share/1.jpg block: 7 recv_size: 122880
filename: share/1.jpg block: 8 recv_size: 143360
filename: share/1.jpg block: 9 recv_size: 163840
filename: share/1.jpg block: 10 recv_size: 184320
filename: share/1.jpg block: 11 recv_size: 204800
filename: share/1.jpg block: 12 recv_size: 225280
rsize: 239366 tsize: 239366
start rename share/1.jpg.lefting ----> share/1.jpg
rename finished
```

```
Run: VMAremonrun x VMBremonrun x VMCremonrun x
C:\Users\78522\AppData\Local\Programs\Python\Python37\python.exe "C:/Users/78522/Desktop/zzs/year 3/CAN 201/CW_1/VMCremonrun.py"
Send main.py to 192.168.56.105:/home/tc/workplace/cw1
##### RUN main.py #####
successfully connect with 192.168.56.103
successfully connect with 192.168.56.104
Downloader is ready...
download, the first ticket: {'filename': 'share/1.jpg', 'mtime': 1608357360.98342, 'filesize': 239366}
I am listening...
I am scanning...
file size: 239366
filename: share/1.jpg block: 1 recv_size: 20480
filename: share/1.jpg block: 2 recv_size: 40960
filename: share/1.jpg block: 3 recv_size: 61440
filename: share/1.jpg block: 4 recv_size: 81920
filename: share/1.jpg block: 5 recv_size: 102400
filename: share/1.jpg block: 6 recv_size: 122880
filename: share/1.jpg block: 7 recv_size: 143360
filename: share/1.jpg block: 8 recv_size: 163840
filename: share/1.jpg block: 9 recv_size: 184320
filename: share/1.jpg block: 10 recv_size: 204800
filename: share/1.jpg block: 11 recv_size: 225280
rsize: 239366 tsize: 239366
start rename share/1.jpg.lefting ---> share/1.jpg
rename finished
download, the first ticket: {'filename': 'share/1.jpg', 'mtime': 1608357360.98342, 'filesize': 239366}
```

4. Action: Add a file (about 2 GB) in the 'share' folder of VMB. After 2 second, kill the program in VMA. When VMC finished downloading, restart VMA
- Result: Scanner in VMB found this file and told the news to peers. Then, peers downloaded this file from VMB. (about 90s) After killing and restarting VMA, VMA and VMC downloaded this file successfully.

```
filename: share/1.mp4 block: 125038 recv_size: 2457291832
filename: share/1.mp4 block: 125039 recv_size: 2457272112
filename: share/1.mp4 block: 125040 recv_size: 2457292592
filename: share/1.mp4 block: 125041 recv_size: 2457313072
filename: share/1.mp4 block: 125042 recv_size: 2457333552
filename: share/1.mp4 block: 125043 recv_size: 2457354032
filename: share/1.mp4 block: 125044 recv_size: 2457374512
filename: share/1.mp4 block: 125045 recv_size: 2457394992
filename: share/1.mp4 block: 125046 recv_size: 2457415472
filename: share/1.mp4 block: 125047 recv_size: 2457435952
rsize: 2457442066 tsize: 2457442066
start rename share/1.mp4.lefting ---> share/1.mp4
rename finished
```

5. Action: Add a folder including one small file to the 'share' folder of VMB.
- Result: Scanner in VMB found this folder and told the news to peers. Then, peers downloaded this folder and the content successfully.

```

filename: share/1.mp4    block: 125601    recv_size: 2457395200
filename: share/1.mp4    block: 125602    recv_size: 2457415680
filename: share/1.mp4    block: 125603    recv_size: 2457436160
rsize: 2457442066  tsize: 2457442066
start rename share/1.mp4.lefting ---> share/1.mp4
rename finished
Connect with ('192.168.56.104', 34610)
received news !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! [{'filename': 'share/ty/group.docx', '
downloader, the first ticket: {'filename': 'share/ty/group.docx', 'mtime': 1608357687.2401311, 'filesize':
file size: 18327
filename: share/ty/group.docx    block: 1    recv_size: 13032
rsize: 18327  tsize: 18327
start rename share/ty/group.docx.lefting ---> share/ty/group.docx
rename finished
|

```

6. Action: Add a small file in the 'share' folder of VMC. After VMA and VMB received this file, change this file in VMC

Result: VMA and VMB downloaded and updated this file successfully.

7. Action: Restart these three virtual machines with the argument '--encryption yes'

Result: The program stopped in the function *encryptfile()* with unknown problems.

Solution: Print functions was used in *encryptfile()*, and found the code *pad(data, AES.block\_size)* has errors. However, the reason of this error was not found and this problem could not be solved.

```

def encryptfile(filename):
    nf = open(filename, 'rb')
    data = nf.read()
    key = b'feimax_zuishai!!'
    iv = b'qwertyuiopasdfgh'
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    encrypted_data = cipher.encrypt(pad(data, AES.block_size))
    ef = open(filename + '.lefting', 'wb')
    ef.write(encrypted_data)
    nf.close()
    ef.close()

def decryptfile(filename):
    key = b'feimax_zuishai!!'

```

## Conclusion:

In this coursework, a LEFT files sharing system was designed and implemented. It based on TCP protocol and peer-to-peer structure, allowing many hosts share their

files at a same time. Moreover, the bandwidth in this application is up to at least 60 Mbps. Although this app could transfer big files efficiently and flexibly, the one defect is that it can not implement the file encryption.