Report

CMPE156 Name: Shuli He, ID:she77@ucsc.edu
Concurrent-File-Transfer

Usage：

Use command 'make' (Makefile) to compile the source code. (Using g++)

Client:

./myclient <server-info.txt> <num-chunk> <filename>

example: ./myclient serverinfo.txt 3 set

Server:

./myserver <port>

example: ./myserver 12345

Protocol Design：

1. Designed a simple UDP function getFilesize() to get the file size from server by using UDP socket. This function also used for testing if the server is available and has the file.
2. Client will use the getActiveSockList() to find the active server in server file which can be use in following file download.
3. For chunks, Client will create same numbers of thread to recive the different

part of the file.

4. Here maintain 3 lists:

vector<int> serverState; //use lock save the server state thread count and is able to connect

vector<int> socklist; // the public socket for the first connect with server

vector<SimpleAddress> addList; //only read the list of servers store the address.

The UDP packet as:

```
struct SimpleUDPmsg{
    int code; //code 1 for query, 2 for read request, 3 for ack
    int filesize;
    int offset;
    int chunksize;
    int numchunk;
    int serverPort;
    char filename[MAX_FILENAME_LEN];
    char buffer[BUFFER_SIZE];
};
```

For downloading:

Design like TFTP

1. The client uses public socket (start the public socket lock) to connect (UDP

actually no connection) to the server and send the code 2 to request file read(download).

2. The server received the code 2 to start new thread. The thread first creates the a new private socket and return the new port number by public socket.
3. As client received the new port it will send back ACK to server and create new private socket (release the lock) then use the new socket to receive the file.
4. When server got the new port ACK, it starts to transmit the file chunk depends on the offset and chunk size.
5. With Each UDP packet, the client will send back ACK and the server only send the next packet when it gets the right ACK.
6. All thread finished the job and clear the memory. The fileAssemble() function will take care of the file chunk and check if there is a missing chunk and assemble to one file by these chunks.

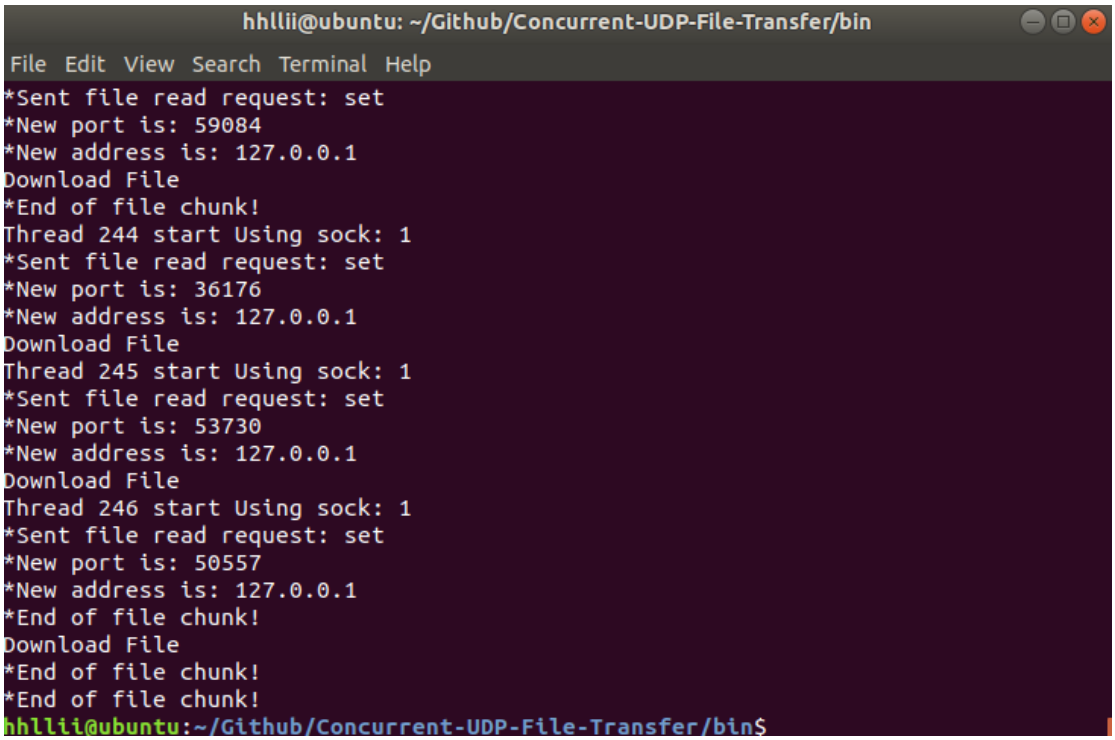
Potential problem:

1. If the file transmission is fast, only the first several of servers will be used.
2. There is a #define MAX_SERVER 1024 set to 1024 which limit the thread number (chunk number).
3. Didn't consider the retransmission with server failure. Server failure will cause file part lost and need to download the file again.
4. If there are too much address in the server info file might load too much and

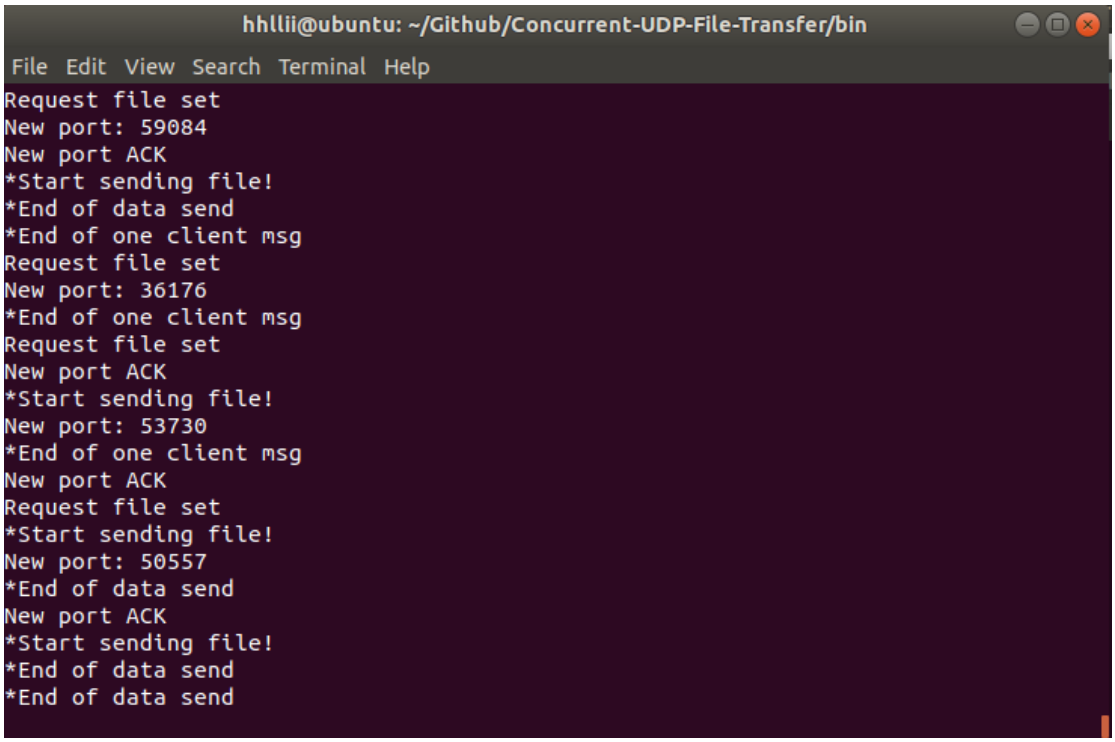
cause the memory excess.

5. Some struct use the program stack might can put in heap.

Test of client:



Test of server:



Diff the test files

```
hhllii@ubuntu: ~/Github/Concurrent-UDP-File-Transfer/bin
File  Edit  View  Search  Terminal  Help
> BASH_VERSINFO=([0]="4" [1]="4" [2]="19" [3]="1" [4]="release" [5]="x86_64-pc-l
inux-gnu")
> BASH_VERSION='4.EM_HOME=/home/hhllii/.rvm/gems/ruby-2.3.0
hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$ diff files/set dest/set

hhllii@ubuntu:~/Github/Concurrent-UDP-File-Transfer/bin$
```