COE 4DN4 Lab 2 -

A Multi-threaded Song/Photo Sharing Application "McNapster"

Name:
ChengXuan Yang 0865949
Fengyi Song 1068106

Date: Feb 18th, 2014

Objective and Introduction

This is the second lab of COMP ENG 4DN4. We aim to develop an FTP client-server system for sharing music and photos. By referring to *TCP/IP sockets in Java*, we are able to start building our projects based upon the defunct Napster file-sharing system using TCP/IP sockets.

Structure:

The file-sharing system consists of server and clients. A server is the host where files are stored. So when a client connects to a server, it would be able to freely communicate with the server.

Our system is a multiple to single connection rather than a single-to-single connection. That is, multiple clients could be connected to a server and request service such as reading and writing file.

Commands:

Basically, our file sharing system contains different commands, namely LIST-ALL, READ, WRITE, BYE AND QUIT, initiated either by Client or Server. The detailed specification is listed as below. QUIT command can only be approved by the last client who sends it. In other words, if the client number is greater than or equal to two, the system cannot be quitted. Noted that, in our file-sharing system, the QUIT command can be received by Client and finally initiated by the Server. Technically speaking, it is an operation of mutual efforts.

Command	Initiated by	Description
LIST-ALL	Client	List the files and their sizes from the server's current music directory.
READ <*filename*>	Client	The client requests the file specified by *filename* and stores it in it's current working directory.
WRITE <*filename*>	Client	The client writes the file specified by *filename* into the server's appropriate directory.
BYE	Client	Exit from the client after closing the connection server, if one exists.
QUIT	Server	Exit from the server after closing the connection(s) with the clients, if there are any.

Experimental Results

1.Start multi-thread server by entering the port number that you want the server to have

```
multithreadserver [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Mar 23, 2014, 11:42:27 AM)
Enter Server Port Number :

12321
Server Socket Established at port number: 12321
```

2. Start client by entering server address and port number

```
TCPEchoClient (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Mar 23, 2014, 11:45:55 AM)
Enter Server Name :
localhost
Enter Server Port Number :
12321
Connected to server
Enter command to send :
```

3. Enter the command on client side console. The read and write command will follow the format as: command <file path>, list-all command will follow the format as: command <directory path>. Please note that under both client/server projects' root folder, there will be the default directory of pub/music and pub/photo.

Read command:

```
TCPEchoClient (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Mar 23, 2014, 11:45:55 AM)
read pub/music/halo.mp3
Client received : read
Server successfuly received: read
Client received : yes
Server transmitting requested file
Stored the file at the directorypub/music/halo.mp3
```

Write command:

```
TCPEchoClient (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Mar 23, 2014, 11:52:19 AM)
Enter command to send :
write pub/photo/1.jpg
Client received : write
Server successfuly received: write
The file to write exists on client
Client received : File successfuly stored at pub/photo/1.jpg on server
File successfuly stored at pub/photo/1.jpg on server
```

List-all command:

```
Enter command to send :
list-all pub/photo
Client received : list-all
Server successfuly received: list-all
Client received : test.txt size= 0bytes
test.png size= 217829bytes
einstein.tif size= 65786bytes
cameraman.tif size= 65786bytes
birds.tif size= 98502bytes
1.jpg size= 148547bytes
```

Bye command:

```
Enter command to send :
bye
Client received : bye
Server successfuly received: bye
Client received : Server thread for client will be closed after client closes the socket conneciton
Closing socket,will exit the connection with server
```

Quit Command:

1.Initiate quit command when more than one clients are connected to the server, the quit request will be automatically rejected by the server

```
TCPEchoClient (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Mar 23, 2014, 11:58:03 AM)
Connected to server
Enter command to send :
quit
Client received : quit
Server successfuly received: quit
Waiting for server to enter quit command
Client received : There are existing clients, can not quit
There are existing clients, can not quit
```

2. Initiate quit when only one client is connected to the server (which is the current client that is waiting for command to be entered), the client will send the quit request to server and wait for server's approval on quitting.

```
TCPEchoClient (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Mar 23, 2014, 12:02:01 PM)
localhost
Enter Server Port Number :
12321
Connected to server
Enter command to send :
quit
Client received : quit
Server successfuly received: quit
Waiting for server to enter quit command
```

3.The server could approve the request by simply entering "quit" on server side console or reject the request by entering nothing.
Entered "quit" on server side console:

```
Handling client at localhost/127.0.0.1:56192
Server received : quit
Please enter quit command in server console
quit
closing the socket connection
Closing server, Server will be shut down
```

Entered nothing on server side console:

```
Handling client at localhost/127.0.0.1:56186
Server received : quit
Please enter quit command in server console

Server side did not enter quit.Socket connections will remain
```

Server and Client Flow Chart

Please refer to the “ServerFlowChart.pdf” and “ClientFlowChart.pdf” in the submission.

Documented Source Code

Please refer to the “McNapsterClientCode” folder and “McNapsterServerCode” folder in the submission.

Issues/Problems Encountered

As mentioned in the introduction, the QUIT command in our system can only be received from Client. Although the server finally approves it, the server cannot quit independently.

In addition, because of limited time of preparation, we did not handle all exceptions or errors that would occur during the process. For example, if the Client has a typo in command when sending instructions, the server will be shut down immediately because of the unreadable commands. This could be handled and solved by catching exceptions if enough time is given.

Conclusion

Napster is a name given to the music-focused online service. It was originally founded as a pioneering peer-to-peer file sharing Internet service that emphasized sharing audio files, typically music. This lab is basically a small project simulating Napster. The Server is receiving command and sends the instructions back to Client while the multiple Clients would share the files and music in the same Server. All the commands are successfully implemented and the knowledge of socket programming on client/server file sharing is gained.