Assignment 2 --- a simple file share Client/Server

1. Introduction

This is a simple and quite not so robust file share ftp-like protocol, just name it as sftp(short for simple ftp). Sftp is implemented upon udp, and it add the mechanism of acknowledge and retransmitting to assure that the file transferred on the unreliable udp is properly transferred.

The sftp needs connection, because the server have to maintain some states (currently there is only one state --- current directory).

Of course, the sftp is just a simple protocol and it can not be as strong and robust as the ftp built on tcp, but fortunately it indeed can transfer files (with lots of limitations...).

2. Environment

The sftp is programmed using java, so it don’t depend on the specified OS environment, and the testing environment is two windows XP node linked within a LAN.

3. Functions and limitations

The functions includes ls(listing files on current), cd(change directory), get(download file), put(upload file), del(delete file), pwd(show the current directory).

The UI is based on command line, just like the classic shell or cmd, the UI is just user-entering-command --> user-wait-reply --> task-finished-print loop.

Now, here comes the quite many limitations:

Firstly, the file transferred must be 8MB, this is limited by the design of the sftp. Secondly, ls can only show the file names in the current dir. Thirdly, user can’t make directories or transfer any directories, only file can be transferred. Fourth, there’s nearly no safety features, and user do not need to provide user-name and passwords, sftp does not provide mechanisms of authorization.

4. System design

The sftp datagrams can be divided into 4 kinds, they are QUERY, ANSWER, ACK, DATA.

4.1 **format**

For the QUERY and ANSWER datagrams:

Type(8 bits)||Code(8 bits)||Session-ID(16 bits)||Session-data-size(16 bits)
Param-size(16 bits)||Parameter(variable,defined by Param-size)
Data(variable,512byte most)

For the ACK and DATA datagrams:

Type(8 bits)||Code(8 bits)||Session-ID(16 bits)
Session-data-id(16 bits)
Data(variable,512byte most)

4.2 **Common filed**

All the 4 types have the same first 4 byte format, Type indicate the datagram types(the 4 types), Code means the specific type of one type of datagram, it is the subtype, QUERY has subtypes of HELLO,BYE,CD,PWD,LS,GET,PUT,DEL, ANSWER has OK,ERR, ACK has ACK_YES,ACK_UPDATE,ACK_ERR, DATA has only one type, that is just common data.And Session-ID means the current session identifier which will be explained next.

4.3 **Session and state**

A session can be viewed as one task, the connection between client and server both maintained the current session number and add it to the datagrams send. When receiving a datagram , both have to check the session number and decide what to do next.

A session is opened by Client sending a QUERY, and the closing of a session need not be synchronized by client and server, once one side has send all the data, and received all the ACK for the data that have been sent, and received all the data that should be received, it can just close the session. And the ACK_UPDATE datagram is used to tell the other side the session has been finished.

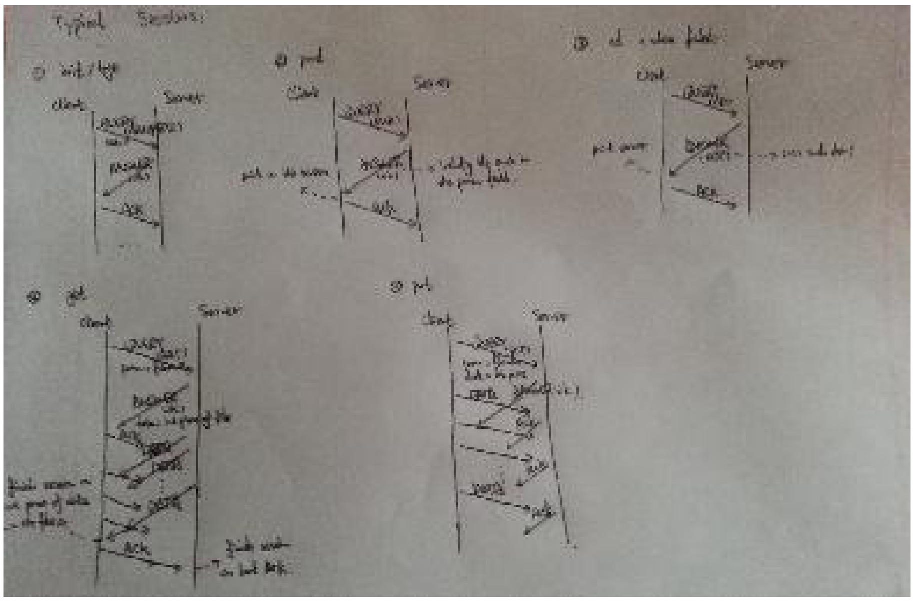
Both client and server have two states, that is IDLE and WAIT, for client it can receive user input from keyboard when IDLE and it can receive datagrams IDLE or WAIT, the server can receive datagrams IDLE or WAIT. In fact, IDLE no current session, and WAIT means the current session has not finished yet. Session number have the limit of 2^15, but maybe this is quite large and enough for usual use.

4.4 **To provide slightly reliable transfer**

Each QUERY,ANSWER and DATA datagrams should be acknowledged, QUERY is acked by ANSWER, and ANSWER, each DATA should be acknowledged by ACK, ACK need not be acked. And those datagrams are all equipped with a timer, the retransfer will be triggerred after specified time --- that is 1s,2s,4s and so on till 1 min, the timing sequence reference the tcp’s idea. And of course the timer is canceled when get acked.

In fact, the retransfer is really needed, when testing the sftp without retransferring, data only transferred ok in one computer(using loopback interface), but when transfer a about 1M file between the two nodes in one LAN, packet-loss occurred.

4.5 **Typical sessions**



4.6 File transferring

The central function of the sftp is to transfer files, the sftp can only transfer a file less than 8M, this is constrained by the filed of the Session-data-size filed of QUERY/ANSWER, each datagram can transfer up to 512 bytes of data, and the Session-data-size is 16 bit, so the limit is 32M, but the real implementation just set a limitation of 8M to avoid the problem of signed number.

As said before, files are transferred in blocks of 512 bytes, and the data can be arrived out of order, but it is send in order and has the limiting window size of 10, that is, the number of datagrams not acked yet is at most 10, or the sending will not be continued until new ACK.

4.7 For the host OS

Although using java, the problems of filesystems is still related to the specified host OS, this implementation doesn't think much about it, so if there is errors when executing the java file-handle api, just think it as failure.

5. Implementation

In fact, the coding for the sftp is really poor-written, many of those codes can be simplified. And unfortunately and really sorry to say, there must be many bugs unfound in the program...

Briefly, the client side and the server side is quite similar, they all need send data, wait data's ACK, and wait the other side's data. So, those things are managed by the class Session, the session record the data for the current session, the unsending data, the sending data but not acked, the receiving data, and can tell whether the session should be finished.

And, here is another reason for not supporting large files, the data managed by the Session are all stored in the main memory, in fact in some HashMap. This is just for simplicity of programming, again really sorry for these. So, if to support larger files, the data should be managed and stored with the help of file system, maybe receiving data stored into some *.part files.

In fact, the Session should manage three things: to send all the data that should be send, to know whether all the data sent have been acked and which ones have not yet, and to store the data that should be received.

About receiving data, the data size is only recorded in the QUERY/ANSWER datagram, for QUERY there is no problem, because that is always the first one, but for ANSWER, it can be arrived later than DATA, again for simplicity, always wait for ANSWER, this is relied on the retransmitting of the sender.

Maybe the Session class is one of the core, and another core is the Connection. ConnectionClient and ConnectionServer are the two classes for client and server, they are different on the actions of the receiving of the datagrams. In fact, they can be viewed as finite state machines. It seems that they can be implemented as the same basic of the mechanism of a fsm and different policies for each side, but I just can not find a good way to do this, so in this implementation, they are almost separated which really leads to some kind of redundancy.

SftpDatagram is the representation of a sftp datagram, and it has the method of analyze and generate a datagram, the retransmit timer is also in SftpDatagram. In this class, a reference also direct to the Connection it belongs to let its sending as its class method.

And at last, the top-level driver is SftpClient and SftpServer, they manage the socket and execute the loop.For the server side, each connection is stored with its IpAddress and port number in a HashMap, for the client side, there is only one connection which get initialized when started, and there is another thread to get the user's input from keyboard.

6. Build and run

To run the applications, the running environment only need jre, and the running start from the command line (windows cmd or unix/linux all kinds of shell).

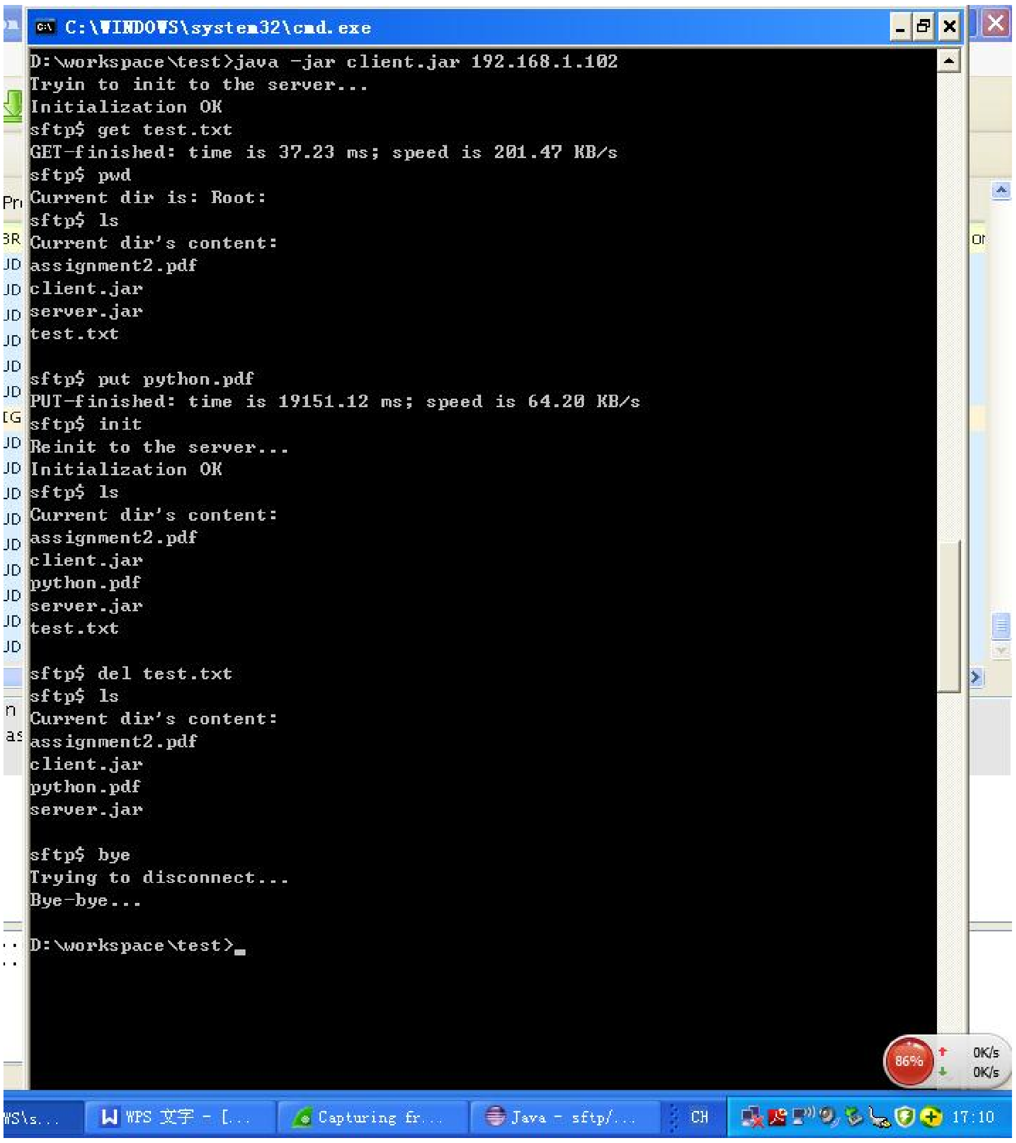
For server, just type **java -jar server.jar**,and Ctrl-C to exit.

For client, type **java -jar client.jar <ServerIpAddr>**, and then can type commands.

7. Testing

The testing environment is two Windows XP (192.168.1.100 and 192.168.1.102) connected inside a LAN.

Here's some examples:



```
D:\workspace\test>java -jar client.jar 192.168.1.102
Tryin to init to the server...
Initialization OK
sftp$ get test.txt
GET-finished: time is 37.23 ms; speed is 201.47 KB/s
sftp$ pwd
Current dir is: Root:
sftp$ ls
Current dir's content:
assignment2.pdf
client.jar
server.jar
test.txt

sftp$ put python.pdf
PUT-finished: time is 19151.12 ms; speed is 64.20 KB/s
sftp$ init
Reinit to the server...
Initialization OK
sftp$ ls
Current dir's content:
assignment2.pdf
client.jar
python.pdf
server.jar
test.txt

sftp$ del test.txt
sftp$ ls
Current dir's content:
assignment2.pdf
client.jar
python.pdf
server.jar

sftp$ bye
Trying to disconnect...
Bye-bye...

D:\workspace\test>_
```

So, in fact the transferring of the file is not so efficient, maybe adjust the sending window limit and adjust the data size per datagram can help.

8. Summary

To be honest, the design and the implementation both have lots of problems, about the design, I’m not quite sure whether he way to transfer and acknowledge the data is all right, maybe to ack QUERY also using ACK can be better, for this can let server and client follow more similar behaviour. Or maybe build another layer upon udp to assure reliable transfer can be better, but this has already realized by tcp, and maybe for file transfer, reliable stream-based transfer can be better which is the way of tcp.

All in all, this sftp is just a simple and testing protocol for file transfer with all those kinds of limitations, better design and implementation needs more time and energy......