Socket Programming—File Share Client/Server

You will implement a client and a server using TCP/UDP Socket APIs. The client could upload/download files to/from the server side. Functions include but not limited to:

1. List the objects in current directory
2. Change directory
3. Download objects from server side
4. Upload objects to server side
5. Delete objects on server side
6. (optional) If you select TCP Socket API, please try to implement according FTP protocol (http://tools.ietf.org/html/rfc959), and try to use your client to talk to SJTU Portal FTP server.
7. (optional) If you select UDP Socket API, please try to implement error control (packet loss and dis-order).
8. UI design is very flexible. Please concentrate on the Socket API programming instead of time-consuming UI.

Language recommended: Java or C/C++.

We will implement this project using java language.
First we will specify our goals.
We will implement a client and a server using TCP Socket APIS, following command are supported.

1. List the objects in current directory
2. Change directory
3. Download objects from server side
4. Upload objects to server side
5. Delete objects on server side
6. make directory
7. rename directory(only by client)
8. User name password authentication
9. Talk to Portal SJTU ftp server(follow RFC 959)

To achieve the above goal, we will first make clear of the protocol (RFC 959) . We will only implement part of the protocol to support our function listed above. Other unconcerned protocol will be neglected.

The client's commands we implement are listed below:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
CWD  <SP> <pathname> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
DELE <SP> <pathname> <CRLF>
MKD  <SP> <pathname> <CRLF>
LIST [<SP> <pathname>] <CRLF>
```

<SP> means space , <CRLF> denotes end-of-line which is similar as ”\n”.
Above commands are exact language that between clients and servers. Server will return status code followed by exact description as answers to the client. the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies indented and under them), then positive and negative completion, and finally intermediary replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

```
Connection Establishment
               120
                  220
               220
               421
            Login
               USER
                  230
                  530
                  500, 501, 421
                  331, 332
               PASS
                  230
                  202
                  530
                  500, 501, 503, 421
                  332
               CWD
                  250
                  500, 501, 502, 421, 530, 550
Transfer parameters
               PASV
                  227
                  500, 501, 502, 421, 530
               TYPE
                  200
                  500, 501, 504, 421, 530
File action commands
               STOR
                  125, 150
                     (110)
                     226, 250
                     425, 426, 451, 551, 552
                  532, 450, 452, 553
                  500, 501, 421, 530
               RETR
                  125, 150
                  (110)
                  226, 250
                  425, 426, 451
               450, 550
               500, 501, 421, 530
            LIST
               125, 150
                  226, 250
                  425, 426, 451
               450
               500, 501, 502, 421, 530
            DELE
               250
               450, 550
               500, 501, 502, 421, 530
            MKD
               257
               500, 501, 502, 421, 530, 550
```

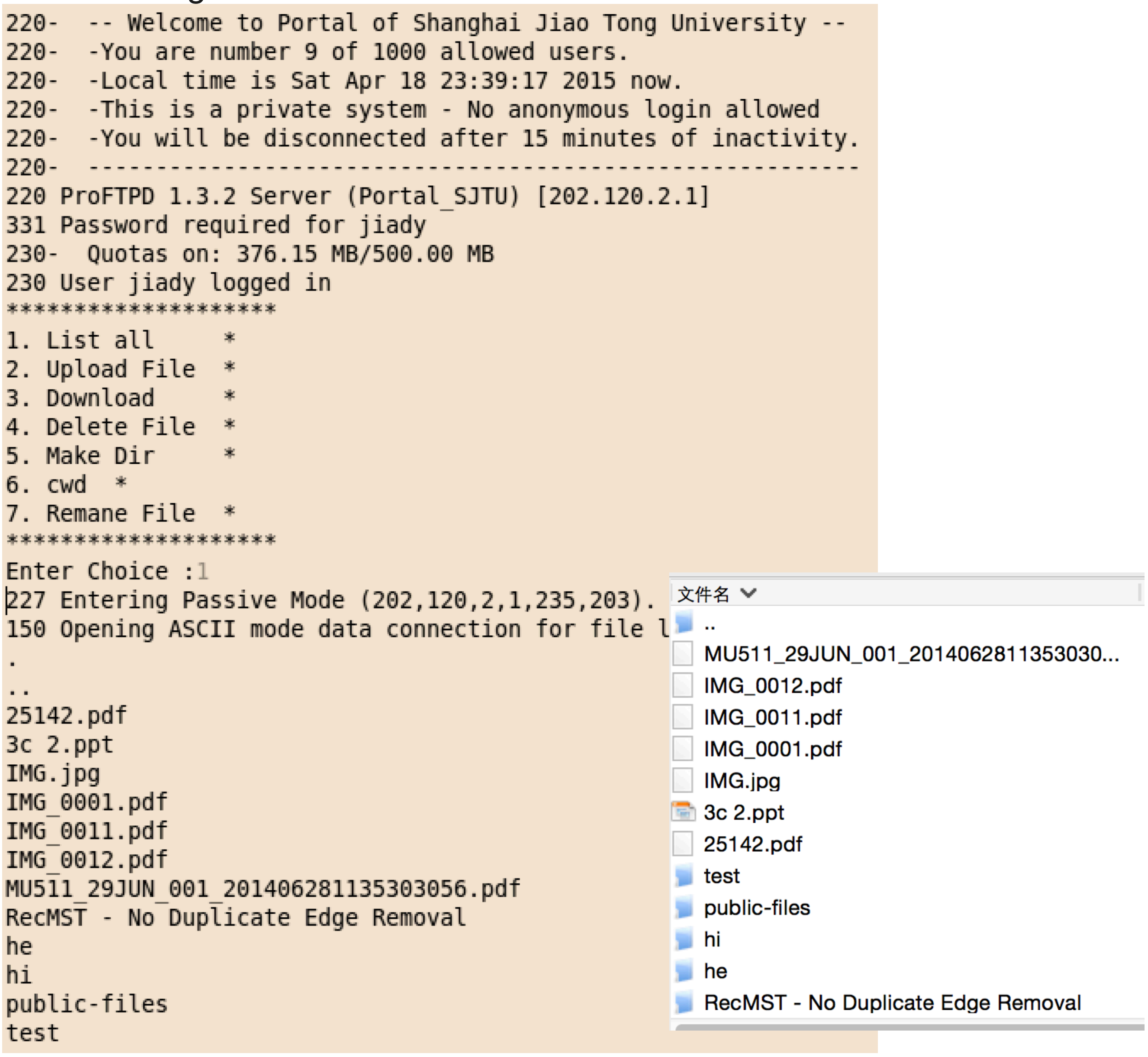
Numeric Order List of Reply Codes

```
         120 Service ready in nnn minutes.
         125 Data connection already open; transfer starting.
         150 File status okay; about to open data connection.
         200 Command okay.
         202 Command not implemented, superfluous at this
site.
         211 System status, or system help reply.
         212 Directory status.
         213 File status.
         214 Help message.
             On how to use the server or the meaning of a
particular
             non-standard command.  This reply is useful only
to the
             human user.
         215 NAME system type.
             Where NAME is an official system name from the
list in the
             Assigned Numbers document.
         220 Service ready for new user.
         221 Service closing control connection.
             Logged out if appropriate.
         225 Data connection open; no transfer in progress.
         226 Closing data connection.
             Requested file action successful (for example,
file
             transfer or file abort).
         227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
         230 User logged in, proceed.
         250 Requested file action okay, completed.
         257 "PATHNAME" created.

         331 User name okay, need password.
         332 Need account for login.
         350 Requested file action pending further
information.

         421 Service not available, closing control
connection.
             This may be a reply to any command if the
service knows it
             must shut down.
         425 Can't open data connection.
         426 Connection closed; transfer aborted.
         450 Requested file action not taken.
             File unavailable (e.g., file busy).
         451 Requested action aborted: local error in
processing.
         452 Requested action not taken.
             Insufficient storage space in system.


         500 Syntax error, command unrecognized.
             This may include errors such as command line too
long.
         501 Syntax error in parameters or arguments.
         502 Command not implemented.
         503 Bad sequence of commands.
         504 Command not implemented for that parameter.
         530 Not logged in.
         532 Need account for storing files.
         550 Requested action not taken.
             File unavailable (e.g., file not found, no
access).
         551 Requested action aborted: page type unknown.
         552 Requested file action aborted.
             Exceeded storage allocation (for current
directory or
             dataset).
         553 Requested action not taken.
             File name not allowed.
```

Passive Mode
In our implementation, there are two connections, control socket and data socket, control flow is for command and replies that mentioned above. Data flow is for data transfer, such as upload, download and list content transfer. When we ask for Passive Mode, The server will

allocate a new connection for data transfer, this new connection will be assigned with a random port in the available range. For example, if the server receive a request PASV, it will return “227 entering passive mode (h1,h2,h3,h4,p1,p2)” h1.h2.h3.h4 is the ip address, p1*256+p2 is the port number.

The following is the screenshot of the client.



```
220-   -- Welcome to Portal of Shanghai Jiao Tong University --
220-   -You are number 9 of 1000 allowed users.
220-   -Local time is Sat Apr 18 23:39:17 2015 now.
220-   -This is a private system - No anonymous login allowed
220-   -You will be disconnected after 15 minutes of inactivity.
220-   ---------------------------------------------------------------
220 ProFTPD 1.3.2 Server (Portal_SJTU) [202.120.2.1]
331 Password required for jiady
230-  Quotas on: 376.15 MB/500.00 MB
230 User jiady logged in
********************
1. List all      *
2. Upload File   *
3. Download      *
4. Delete File   *
5. Make Dir      *
6. cwd   *
7. Remane File   *
********************
Enter Choice :1
227 Entering Passive Mode (202,120,2,1,235,203).
150 Opening ASCII mode data connection for file l
.
..
25142.pdf
3c 2.ppt
IMG.jpg
IMG_0001.pdf
IMG_0011.pdf
IMG_0012.pdf
MU511_29JUN_001_201406281135303056.pdf
RecMST - No Duplicate Edge Removal
he
hi
public-files
test
```

```
文件名
..
MU511_29JUN_001_2014062811353030...
IMG_0012.pdf
IMG_0011.pdf
IMG_0001.pdf
IMG.jpg
3c 2.ppt
25142.pdf
test
public-files
hi
he
RecMST - No Duplicate Edge Removal
```

other command can be invoked by entering corresponding number.

A typical logic looks like this(take login as an example)

```
        public void login() throws Exception{
                out.println("USER " + username);
                String response ;
                response = readUntil("331 ");
                //System.out.println(response);
                if (!response.startsWith("331")) {
                        throw new IOException(
                                        "SimpleFTP received an unknown response after sending the user: "
                                                        + response);
                }

                out.println("PASS " + password);

                response = readUntil("230 ");
                //System.out.println(response);
                if (!response.startsWith("230")) {//230 login success
                        throw new IOException(
                                        "SimpleFTP was unable to log in with the supplied password: "
                                                        + response);
                }
        }
```

For passive request, the logic is to get the address of the data flow and assign a socket to it. Data transfer will be processed with data socket.

```
        public Socket passive() throws Exception{

                String response ;
                out.println("PASV");
                response = readUntil("227 ");
                if (!response.startsWith("227")) {
                        throw new IOException("FTPClient could not request passive mode: "
                                        + response);
                }
                String ip = null;
                int port1 = -1;
                int opening = response.indexOf('(');
                int closing = response.indexOf(')', opening + 1);
                if (closing > 0) {

                        String dataLink = response.substring(opening + 1, closing);
                        StringTokenizer tokenizer = new StringTokenizer(dataLink, ",");
                        try {
                                ip = tokenizer.nextToken() + "." + tokenizer.nextToken() + "."
                                                + tokenizer.nextToken() + "." + tokenizer.nextToken();
                                port1 = Integer.parseInt(tokenizer.nextToken()) * 256
                                                + Integer.parseInt(tokenizer.nextToken());
                        } catch (Exception e) {
                                throw new IOException(
                                                "FTPClient received bad data link information: "
                                                                + response);
                        }
                }
                return new Socket(ip, port1);
        }
```

For Data transfer, the logic is that we read the inputstream, at most 4096 bytes at one time, when the end of file shows in the inputstream, we finish the writing.

```
                    BufferedOutputStream output = new BufferedOutputStream(
                                    new FileOutputStream(new File(filenameLocal, filename)));
                    BufferedInputStream input = new BufferedInputStream(
                                    dataSocket.getInputStream());
                    byte[] buffer = new byte[4096];
                    int bytesRead = 0;
                    while ((bytesRead = input.read(buffer)) != -1) {
                            output.write(buffer, 0, bytesRead);
                    }
                    output.flush();
                    output.close();
                    input.close();
```

For the Server, it should support multiple clients, in other words, it should maintain the status of each clients without interference with each other. We will use a ServerSocket to wait for client visit, if one client comes, we will run a thread and assign a socket to this client to run the inner logic. The Server will then run back to the waiting state for the next client.

```
  public void run()
  {

     while (true)
        {
          System.out.println("Listenning...");
          try
          {
//             每个请求交给一个线程去处理
             sk = server.accept();
             if(sk!=null){
                      ServerThread th = new ServerThread(sk);
                      th.start();
             }
          }
          catch (Exception e)
          {
             e.printStackTrace();
          }

        }
  }
```

The Server will answer to each command with only one statement(for SJTU portal, it may reply multiple statement, but our server will reply only one. But our client can handle these different with the same structure.)

The main part is:

```
switch (arg[0]){
            case "USER":
                  user(arg);
                                          break;
            case "PASS":
                  pass(arg);
                                          break;
            case "PASV":
                  pasv();
                                          break;
            case "TYPE":
                  type(arg);
                                          break;
            case "STOR"://upload
                  upload(arg);
                                          break;
            case "LIST":
                  list(arg);
                                          break;
            case "RETR"://download;
                  retr(arg);
                                          break;
            case "DELE":
                  dele(arg);
                                          break;
            case "CWD":
                  cd(arg);
                                          break;
            case "MKD":
                  mkdir(arg);
                                          break;
            default:
                  out.println("202 Command not implemented, superfluous at this site.");
                  break;

      }
```

We use a stack to maintain the current path of the server. For simplicity, one command can only move up or down one level of the folder.

```
public void cd(String [] args){
        if(args.length>1){
                String dir=args[1];
                if (!valid(dir))
                {
                        out.println("451 move only one level at one time.'/' is not allowed");
                }
                else
                {
                        if ("..".equals(dir))
                        {
                                if (path.size() > 0)
                                        path.pop();
                                else{
                                        out.println("451 Requested action aborted: already in root dir.");
                                        return;
                                }
                        }
                        else if (".".equals(dir))
                        {
                                ;
                        }
                        else
                        {
                                File f = new File(getpath());
                                if (!f.exists()){
                                        out.println("451 Requested action aborted: Directory does not exist: " + dir);
                                        return;
                                }
                                else if (!f.isDirectory()){
                                        out.println("451 Requested action aborted:Not a directory: " + dir);
                                        return;
                                }
                                else{
                                        path.push(dir);
                                }
                        }
                        out.println("250 dir switched.");
                }

        }else{
                out.println("500 Syntax error, command unrecognized.");
        }
    }
```

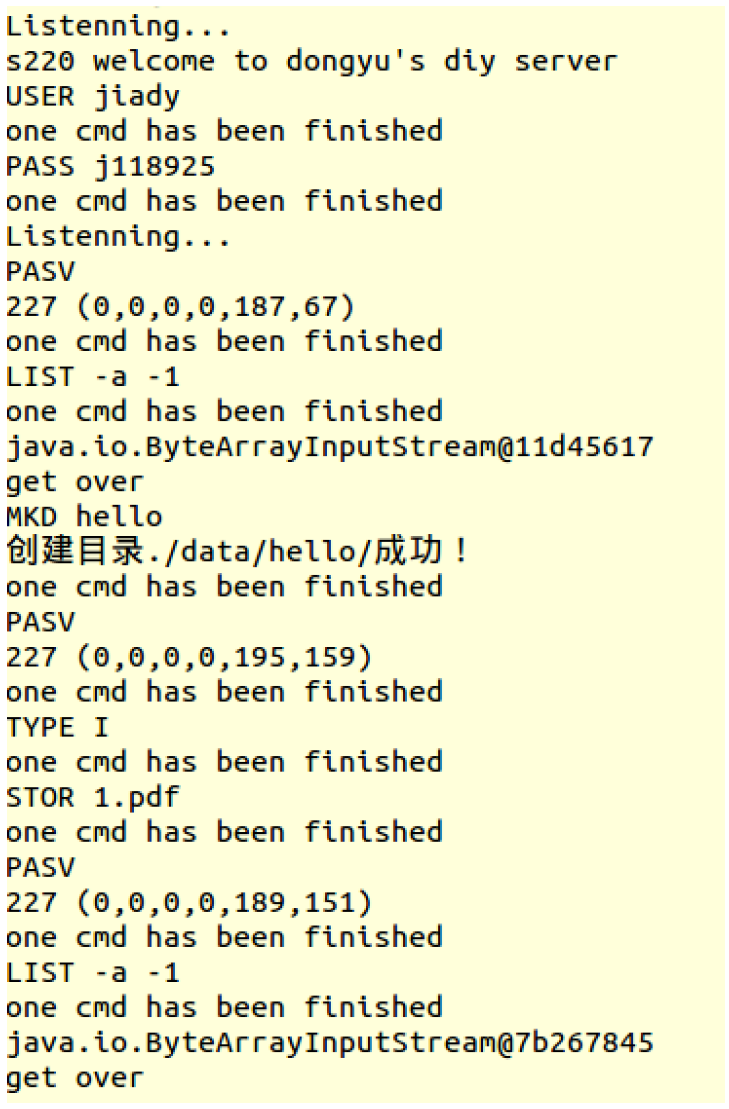
For passive mode , The logic looks like this:

```
    public void pasv() throws IOException{
        serverSocket = new ServerSocket(0, backlog, host);
        InetSocketAddress h=(InetSocketAddress) (serverSocket.getLocalSocketAddress());
        int p=h.getPort();
        int p1=p%256;
        int p0=p/256;
        byte b[]=h.getAddress().getAddress();
        System.out.printf("227 (%d,%d,%d,%d,%d,%d)\n",b[0],b[1],b[2],b[3],p0,p1);
        out.printf("227 (%d,%d,%d,%d,%d,%d)\n",b[0],b[1],b[2],b[3],p0,p1);
    }
```

For data transfer, We will open another thread for data transfer. Following code is the download thread which will be invoke in list and download file procedure.

```
    private static class GetThread implements Runnable//download
        {
                private ServerSocket dataChan = null;
                private InputStream file = null;
                PrintWriter pt;

                public GetThread(ServerSocket s, InputStream f,PrintWriter out)
                {
                        dataChan = s;
                        file =  f;
                        pt=out;
                }
                public void run()
                {
                        try
                        {
                                //System.out.println(" wait for the client's initial socket");
                                Socket xfer = dataChan.accept();
                                //System.out.println(" Prepare the output to the socket");
                                BufferedOutputStream out = new BufferedOutputStream(
                                                xfer.getOutputStream());
                                System.out.println(file.toString());
                                // read the file from disk and write it to the socket
                                byte[] sendBytes = new byte[4096];
                                int iLen = 0;
                                while ((iLen = file.read(sendBytes)) != -1)
                                {
                                        out.write(sendBytes, 0, iLen);
                                }
                                out.flush();
                                out.close();
                                xfer.close();
                                file.close();
                                System.out.println("get over");
                        }
                        catch (Exception e)
                        {
                                e.printStackTrace();
                        }
                        pt.println("200 trasfer complete");
                }
        }
```

Part of Server Running Log



```
Listenning...
s220 welcome to dongyu's diy server
USER jiady
one cmd has been finished
PASS j118925
one cmd has been finished
Listenning...
PASV
227 (0,0,0,0,187,67)
one cmd has been finished
LIST -a -1
one cmd has been finished
java.io.ByteArrayInputStream@11d45617
get over
MKD hello
创建目录./data/hello/成功！
one cmd has been finished
PASV
227 (0,0,0,0,195,159)
one cmd has been finished
TYPE I
one cmd has been finished
STOR 1.pdf
one cmd has been finished
PASV
227 (0,0,0,0,189,151)
one cmd has been finished
LIST -a -1
one cmd has been finished
java.io.ByteArrayInputStream@7b267845
get over
```