2. Group Practice Option 2: FTP System

2.1. Introduction

FTP is an application layer protocol created for the transference of files. This protocol is based on the client-server paradigm. In this practice you will implement a reduced version of an FTP client and an FTP server.

2.2. FTP Operation

The FTP protocol is implemented using TCP as transport protocol. For this, servers use two ports: 20 for to send data and 21 to send control commands and responses. This protocol uses out-of-band signaling.

In this protocol, the server and the client maintain a dialog similar to other text-based protocols. The FTP client sends commands to the server and the server generates response codes and performs actions. Caused by the commands, the server and the client will transfer files. This process can be done in two ways: active mode or passive mode. This nomenclature is given by the server behavior during the connection.

Active mode

In the active mode, when the client request any kind of information to the server (files, lists, actual directory...) or wants to upload a file, the client indicates a listening port to the server. The server plays an active role: The server connects to this port of the client by stablishing a TCP connection from the port number 20. Then, the transference of information starts.

Sometimes, the client is not visible from the internet (for example, when it is connected inside a privet network or behind a firewall). To solve this situation, the passive mode was created.

Passive mode

In the passive mode, the client doesn't indicate a port to the server. The client only indicates its desire to use passive mode. In this operation mode, the server indicates to the client the port where it is waiting for a data connection and the client stablish the TCP connection to this port.

2.3. FTP Protocol

FTP has several phases. In this practice you will only implement some of them. Each phase is composed by a request from the client using a command an one or more responses from the server.

The client sends commands in text mode ending each one with a "Carrier Return" and a "Line Feed" (<CRLF>).

Server responses starts with a 3-digit code and can be classified based on the two first ones.

2.3.1. Server responses

In FTP, clients send text-based-commands to the server and the server responds with 3-digit codes. These codes indicate the state of the server and give a text with an explanation of it.

Codes are classified depending on the digits they have in the first and the second positions:

1yz	**Positive Preliminary reply**	The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult. The server-FTP process may send at most, one 1yz reply per command.
2yz	**Positive**	The requested action has been successfully completed. A new request

	Completion reply	may be initiated.
3yz	**Positive Intermediate reply**	The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.
4yz	**Transient Negative Completion reply**	The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server- and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)
5yz	**Permanent Negative Completion reply**	The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

x0z	**Syntax**	These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.
x1z	**Information**	These are replies to requests for information, such as status or help.
x2z	**Connections**	Replies referring to the control and data connections.
x3z	**Authentication and accounting**	Replies for the login process and accounting procedures.
x4z	**Unspecified**	
x5z	**File system**	These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

2.3.2. Basic Sequence

The basic sequence must be implemented in all the works. To send a command, the client will use the control connection using text. Example: "LIST\r\n" o "RETR file.txt\r\n".

To express some symbols or ASCII characters in text is complex in some cases. To solve it, in this document we will use the following ones (they are the same as in the RFC):

<SP>: Space. It's used to separate commands from arguments.

<CRLF>: End of line and carrier return. It means "\r\n".

Connection

At the beginning, the client establishes a connection in the port 21 of the server. They will use this connection to exchange **text-based commands**. Once the connection is stablished, the server will send:

- 220 Service ready for new user.

Now, several actions can be performed by the client.

Data connection establishment

To initiate any data transference in active mode, the client must send:

```
PORT <SP> <host-port> <CRLF>
```

The port is indicated using the following structure:

```
h1,h2,h3,h4,p1,p2
```

where `hx` and `px` are 8-bit integers. The sub-index indicates the order of the bit starting from the left.

The server will respond with:

- 200 Command okay.

You have to close the connection once the transfer has end. To make a new connection is necessary to send a new PORT command. In other case the server will response:

- 503 Bad sequence of commands.

Commands that require data connection

List all files in a directory

First of all, the client must be listening to the port indicated to the server in character mode. Then it can send the command:

```
LIST [<SP> <pathname>] <CRLF>
```

If no directory is indicated, the server assumes actual folder.

The server will respond with one of the following messages:

- 150 File status okay; about to open data connection.
    - 226 Closing data connection. Requested file action successful (for example, file transfer or file abort).
    - 425 Can't open data connection.
    - 451 Requested action aborted: local error in processing.
- 450 Requested file action not taken. File unavailable (e.g., file busy).
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).

In the first case, the client must wait one of the other commands. If all works properly, the server will send 226. In other case, the server will explain the error. The list of files is received in the data connection.

Download a file from the server

To download a file, the client must be listening to the port indicated to the server in stream of bytes mode. Then, the client must send the following command:

```
RETR <SP> <pathname> <CRLF>
```

The server will connect to the port and immediately will start sending the file. The end of the file is indicated by closing the connection.

The different responses that the server could send are:

- 150 File status okay; about to open data connection.
    - 226 Closing data connection. Requested file action successful (for example, file transfer or file abort).
    - 451 Requested action aborted: local error in processing.
    - 425 Can't open data connection.
    - 426 Connection closed; transfer aborted.
- 450 Requested file action not taken. File unavailable (e.g., file busy).
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).

Upload a file to the server

To upload a file, the client must be listening for connections in the port indicated to the server in stream of bytes mode. Then, the client must send the following command:

```
STOR <SP> <pathname> <CRLF>
```

The server will connect to the port and immediately will start receiving the file. The end of the file is indicated by closing the connection.

The different responses that the server could send are:

- 150 File status okay; about to open data connection.
    - 226 Closing data connection. Requested file action successful (for example, file transfer or file abort).
    - 451 Requested action aborted: local error in processing.
    - 425 Can't open data connection.
- 450 Requested file action not taken. File unavailable (e.g., file busy).
- 452 Requested action not taken. Insufficient storage space in system.
- 553 Requested action not taken. File name not allowed.

Close the connection

To close the connection, the client has to send the command:

```
QUIT <CRLF>
```

The server will respond with the following code and will close the control connection.

- 221 Service closing control connection.

2.3.3. Optional modules

Authentication

Authentication needs two commands. The first one is:

```
USER <SP> <username> <CRLF>
```

For security reasons, the server should always send:

- 331 User name okay, need password.

There are other possibilities for the response code but won't be used in this practice.

The second command is:

```
PASS <SP> <password> <CRLF>
```

The response code depends on the validity of the user's password

- 230 User logged in, proceed.
- 530 Not logged in.

Get the path to the working directory

The client must send:

```
PWD <CRLF>
```

The server will reply with the following code:

- 257 "*path_of_the_directory*".

Change the working directory

To change the working directory you have to send:

```
CWD <SP> <pathname> <CRLF>
```

Pathname indicates the new directory.

The response codes are:

- 250 Requested file action okay, completed.
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).

Create a directory

To create a directory you have to send:

```
MKD <SP> <pathname> <CRLF>
```

The response code will be:

- 257 "*directory-name*" directory created.
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).

Remove a directory

To remove a directory the client must send:

```
RMD <SP> <pathname> <CRLF>
```

The server will reply with one of the following codes:

- 250 Requested file action okay, completed.
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).

Delete a file

To remove a file from the server, the client must send:

```
DELE <SP> <pathname> <CRLF>
```

The server will reply with one of the following codes:

- 250 Requested file action okay, completed.
- 450 Requested file action not taken. File unavailable (e.g., file busy).
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).

Rename a file or a directory

To rename, the client must send two commands. The first one indicates the object to rename:

```
RNFR <SP> <pathname> <CRLF>
```

The server will reply:

- 450 Requested file action not taken. File unavailable (e.g., file busy).
- 550 Requested action not taken. File unavailable (e.g., file not found, no access).
- 350 Requested file action pending further information.

If the server replies the code 350, the client must indicate the new name or the new path to the file:

```
RNTO <SP> <pathname> <CRLF>
```

Finally, the server will reply with one of the following codes:

- 250 Requested file action okay, completed.

- 553 Requested action not taken. File name not allowed.

Passive connection

To use passive mode when the information is sent, the client must send the command:

```
PASV <CRLF>
```

The server will create a new `ServerSocket` in a port and will reply with:

- 227 Entering Passive Mode (h1,h2,h3,h4,p1,p2)

The format of the address is the same as in the active mode. For the next information transference, the server will wait for the client to stablish the connection.

2.3.4. Other codes sent by the server

Other codes that should be used in this practice are:

- 500 Syntax error, command unrecognized.
- 501 Syntax error in parameters or arguments.
- 202 Command not implemented, superfluous at this site.
- 502 Command not implemented.
- 503 Bad sequence of commands.
- 421 Service not available, closing control connection.

2.4. Organization of the practice

This practice is designed to be developed in a modular way. You should start with a basic and mandatory configuration. Later, you can add more modules to get a better grade.

All the code for the practice has to be written in Java using the libraries installed with the JSDK.

2.4.1. Basic configuration (Mandatory)

The basic server for the practice must have the following characteristics:

- Single thread server.
- Active connection mode.
- Single directory.
- List the files in the directory.
- Receive files
- Download files
- Binary transference mode for files and ASCII for the list of files.

The basic client for the practice must have the following characteristics:

- Configurable IP and port to connect to the server.
- FTP commands must be hidden to the user.
- Binary transference mode for files and ASCII for the list of files.
- Simple to use.

2.4.2. Optional modules

- **Authentication system**: You will implement the commands for authentication, but every user with any password will be able to access to the system.
- **User Control**: You have to implement the Authentication system and to control users and passwords.
- **Directory System**: The system will use directories. The following actions must be permitted: Change the directory, get the actual path, make directories, and remove directories.
- **Multithread Server**: The server will be able to accept several connections at the same time. You have to control and to limit the number of concurrent connections.
- **Graphic User Interface for the client**.
- **Delete and rename files**: The system will permit to remove or rename files.
- **Server log**: The server will save in a file all the actions and received requests. If you have implemented the user contro, the log must indicate the user that has made the request.
- **Passive mode**: Client and server will be able to open the data connection in passive mode.

2.5. Marks

The final mark depends on the modules you have implemented. Each module, depending on the number of students, has a maximum value. In the following table you can read all the combinations:

	2 students	**3 students**	**4 students**	**5 students**
Base	8 points	7 points	6 points	5 points
Authentication	1 point	0.7 points	0.5 points	0.3 points
User control	1.5 points	1.2 points	1 point	0.8 points
Directory System	3 points	2.5 points	2 points	1.7 points
Multithread Server	3 points	2.5 points	2 points	1.7 points
Graphic User Interface for the client	1.5 points	1.25 points	1 point	0.75 points
Delete and rename files	1 point	0.7 points	0.5 points	0.3 points
Server Log without users [with users]	0.5 points [1 point]	0.4 points [0.8 points]	0.3 point [0.6 points]	0.2 points [0.4 point]
Passive mode	2.5 points	2 points	1.5 points	1.3 points

Every group can propose other modules to the professor. The lecturer will assign marks to the new modules.

The maximum grade you can get is 10. The practice will be evaluated over the total of the modules and later the maximum will be applied. You also have to test the program with the lecturer. During this test, you can restart the client as many times as necessary **but never the server. The server has to *serve* all the time. If the server breaks, the correction of the practice will end.**

The mark in each module is not only based on the technical results. According to the teaching guide, this practice will be evaluated based on the following learning outcomes:

R01 Assimilate, comprehend and manage protocols.

R02 Comprehend and use complex architecture and systems.

R03 Master the programming linked to this discipline.

R04 Work methodically.

R06 Work productively in a team.

R07 Comprehend and produce technical documents in English.

The work is in group, but the students will be grade individually according to these learning outcomes.

2.6. Documentation

You must present a document with the main information about the practice.

In the first page you must write:

- Names of the members of the group.
- Implemented modules.

In the following pages you must describe your solution: classes, organization, diagrams…

At the end of the document you must indicate who was involved in each module of the project and in the different tasks (organization, testing, documentation, etc.). This part could be used to differentiate the grades of the different members of the team. This description will be verified by the teacher during the evaluation of the practice.

2.7. Other information

You can download the SDK in:

- http://www.oracle.com/technetwork/java/javase/downloads/index.html

You can read the Java documentation in:

- http://download.oracle.com/javase/8/docs/

In these practices the main classes you have to use are:

- http://download.oracle.com/javase/8/docs/api/java/io/package-summary.html
- http://download.oracle.com/javase/8/docs/api/java/net/package-summary.html

In the PDU you can find examples of these classes to understand the TCP Socket class in Java.

You also can find the RFC of both standards.

It's very recommended to use Wireshark to sniff real servers. Using this, you will be able to understand all the instructions of these protocols.