Introduction to computer networking

Second part of the assignment

Academic year 2019-2020

Abstract

In this assignment, students will have to implement a server application using Java Sockets.
The server uses the FTP protocol to provide access to a remote file system. This assignment will use the following additional concepts : control and data connection, user authentication, active and passive mode, binary and ascii transfer mode, directory navigation and thread pools.
Students will work in teams of 2 students.
The deadline for this project is December, 15th.

Section 1 is an introduction to the project. Sections 2 and 3 are an executive summary of the technologies we use. Sections 4 and further provide the assignment's objective as well as practical guidelines.

1 Introduction

As stated in the abstract, you will implement a FTP Server using Java Sockets. The server waits for TCP connections on a given port, and can handle FTP requests through that connection, called "control connection" when established. Upon file transfer (whether upload or download), a new connection, called "data connection" will be established between the client and server. This connection can be established either in active or passive mode, and the data transfer can be made either in text mode or in binary mode. Every file handled by your server is virtual, in the sense that it does not correspond to a file on disk, but exists only in the memory of your server (as a tree structure for example).

You don't have to write a corresponding client code but will use a FTP client instead. For your convenience, Filezilla has been installed on ms8** machines.

Beyond the implementation of the server, one objective of this assignment is for you to learn, on your own, the details of the FTP protocol. This assignment will only provide some high-level basic knowledge, but is certainly not sufficient to be able to achieve the implementation. A good entry point for documentation would probably be the RFCs related to FTP, such as RFC 959, for starters.

2 The FTP Protocol

FTP is a client/server application protocol that enables files to be transferred between machines.

2.1 FTP architecture

As depicted in Figure 1, the FTP architecture uses separate control and data connections.

The control connection is used to send requests (e.g. `PASV`) from the client to the server, and to send responses from server to the client. This connection is *text-oriented*, meaning that all exchanges are intended to be human-readable.

The data connection is used for transferring files (whether upload or download) between the client and the server. This connection may be either text-oriented or binary-oriented[1]. While the control connection stays enabled during the whole session, multiple data connections can be created, but only one is active at a time.

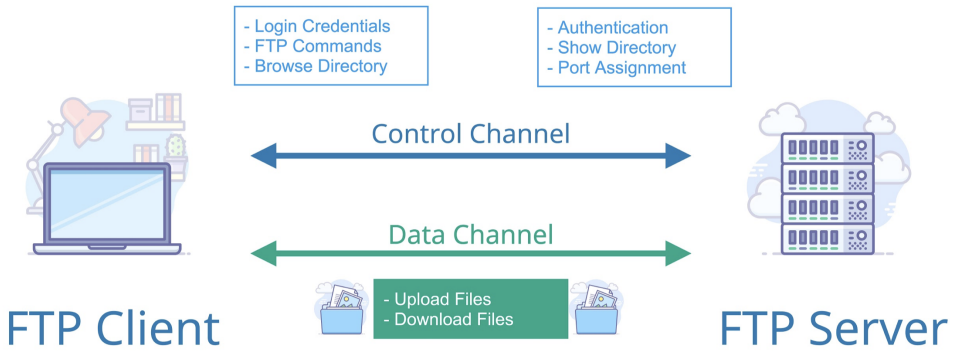

Figure 1: *Graphical overview of TCP*
(img src : https://www.exavault.com/blog/what-is-ftp-tutorial-video-blog/)

[1] features custom information packing

2.2 Overall FTP command syntax

A FTP request is a line of text starting with a method in upper-case and ending with a CRLF. Some commands only contain the method while others take extra parameters. For instance, `RETR File.zip` and `PWD` are valid FTP requests. A list of FTP requests can be found at this address : `https://en.wikipedia.org/wiki/List_of_FTP_commands`

A FTP response starts with a 3-digit number followed by some human-readable information about the response. The first digit is used to indicate one of three possible outcomes — success, failure, or to indicate an error or incomplete reply. The second digit defines the kind of error: (0) syntax, (1) information, (2) connections, (3) authentication and accounting, (4) not defined, (5) file system. The third digit of the reply code is used to provide additional detail for each of the categories defined by the second digit. For instance code 226 indicates that the data connection is about to be closed after a successful file transfer. A list of FTP return codes can be found at this address : `https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes`.

FTP operations may span over several requests. For example, the authentication is achieved with a first `USER` request followed by a `PASS` request. Operations may also be achieved via both control and data connection. For instance, a `RETR` or `STOR` request in the control connection will indicate that a transfer is required and the data connection will be created.

2.3 User authentication

User authentication is done in two steps. First, the client will send a `USER` request (containing the username, or "anonymous"), to which the server responds with a 331 code requesting the password. Then, the client sends a `PASS` request with the password. The server then responds either with 230 (success) or 430 (Login/pass incorrect).

When the user authentication is required, and the client tries to access protected files, the server will respond with a code in the 530 series.

As a remark, note that the credentials are sent in the clear, problem that we will not address in this course, but justifies the use of SFTP instead of FTP nowadays.

2.4 Active and passive mode

The data connection can be established either in active or in passive mode.

In active mode, the client selects a port number, then the server establishes the connection on that port. The client uses the `PORT` command with 6 numbers as parameters, separated by a comma. The first 4 numbers are the 4 parts of the IP address of the client (e.g. 139.165.8.211) and the last 2 numbers indicate the port number. Given that

the two numbers are X and Y, the corresponding port is X*256+Y. For instance, 154 and 208 correspond to 154*256+208 = 39632. Figure 2 shows a possible exchange in active mode.

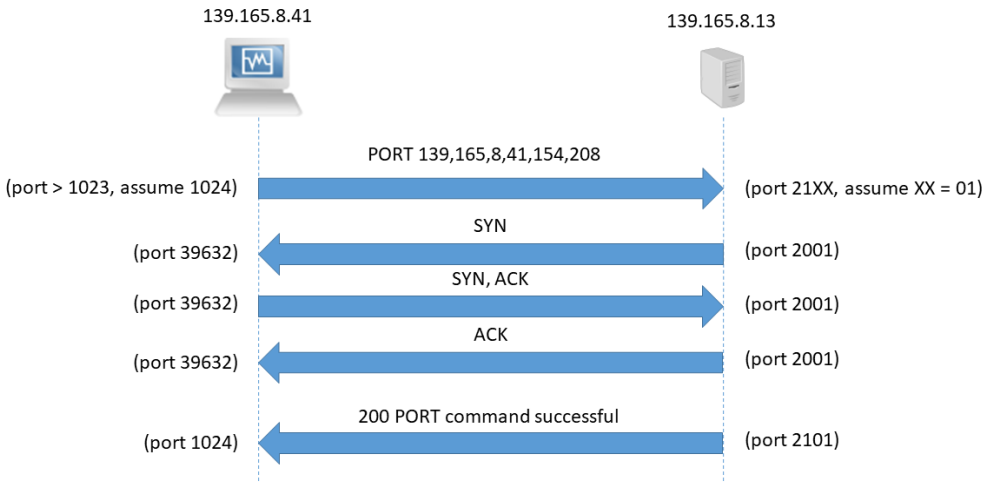

Figure 2: *Possible exchange in active mode*

The problem with active mode is that the server needs to establish a connection to the client, which may cause some problems if the client is behind a firewall. Passive mode is the solution to this problem and is the default for FTP data connection establishment.

In passive mode, the client requests the data connection, but the server indicates on which port the client should initiate it. The client sends a `PASV` request to which the server respond with a 227 code, followed by some text ("Entering Passive Mode") and 6 numbers indicating the IP address of the server and the port number (as in active mode). Figure 3 shows a possible exchange in passive mode.

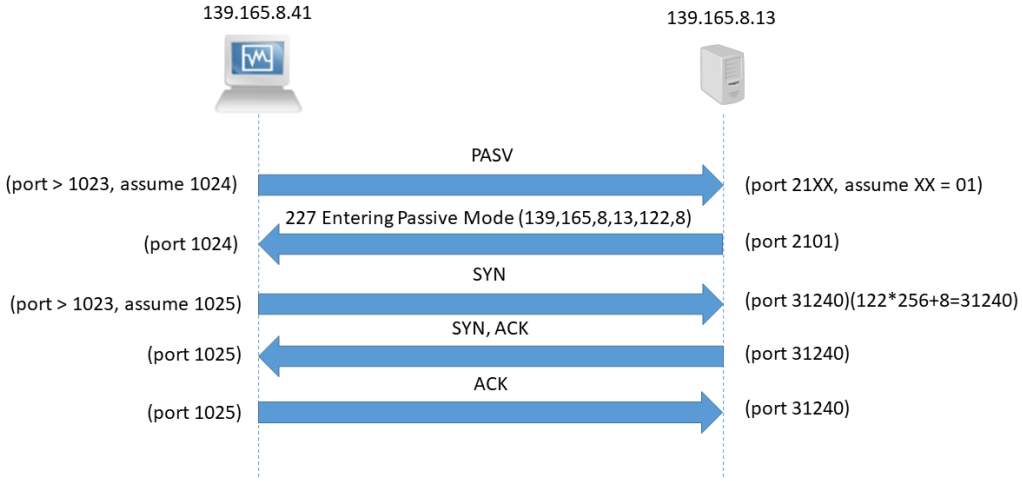

Figure 3: *Possible exchange in passive mode*

2.5 Binary and ASCII mode

File transfer can be achieved either in binary or ASCII (text) mode.

Selecting the type of transfer is done with the `TYPE` request, with a character as parameter which is either `I` for binary or `A` for ASCII.

In text mode, the content of the file is transferred as a string of characters, while in binary mode it is transferred as an array of bytes.

Since you will only develop the server, you should not be concerned about what type to use in which situation, that's the role of the client, but for your information, binary mode is used with most files (images, videos, raw data file, ...) while text mode is used with text files encoded in UTF-8. Using text mode for a binary file (such as an image) might lead to a corruption of the file, while using binary mode for a text file could also be potentially problematic since all operating systems are not necessarily consistent with each other with respect to the characters used to delimit an end-of-line.

2.6 Directory navigation

When uploading or downloading a file, the provided path is a *relative* one (e.g. "File.txt") instead of an *absolute* one (e.g. "/private/File.txt"). It is thus the responsibility of the server to keep the location of the current directory in memory, as well as allow the client

to navigate through that directory.

The following methods will thus be very helpful:

- **CDUP**: change to parent directory;
- **CWD**: change working directory;
- **LIST**: list the content of a given directory, or the current one if no parameter;
- **PWD**: gives the path of the current directory;

3 Thread Pool

When a server accepts a connection, it usually invokes a new *thread* that will handle that connection, so that the server can go back to listening to the port. This is very convenient to guarantee a certain level of accessibility but it also has a flaw.

The *(Distributed) Denial of Service* (or(D)Dos) is an attack that targets servers with this kind of behaviour. In this attack, one (for DoS) or several (for DDoS) machines initiate many bogus connections. If the server launches a new thread for each of these connections, it will soon encounter performance problems or even crash.

To circumvent this problem, one can use a *Thread Pool* that limits the number of threads that can be executed concurrently, while keeping the other jobs on hold until new threads become available. This thread pool can be implemented through the use of `java.util.concurrent.Executors` by calling the `newFixedThreadPool(int maxThreads)` method to create a fixed-size pool of *maxThreads* threads and calling the `execute(Runnable worker)` method to assign the work represented by *worker* to one of a thread in the pool, when available.

You can (and are encouraged to) use a thread pool in your assignment.

4 Project statement

4.1 Mandatory features

The FTP protocol is quite complex, and this assignment doesn't aim at implementing the whole protocol, but the following operations should at least be possible :

- establish proper connections with Filezilla and handle requests such as SYST, FEAT and MDTM;
- download a file;

- upload a file;
- rename a file;
- delete a file;
- navigate in the directory;
- list the content of the current directory.

You don't have to handle the creation/renaming/deletion of folders.

The establishment of the data connection must be possible in active mode and in passive mode, and the data transfer must be possible in ASCII mode and in binary mode.

4.2 Directory content

The directory is refreshed every time your server is restarted (since all files are virtual and only stored in the RAM) and its content is described in figure 4.

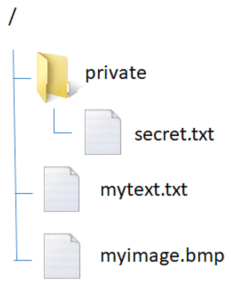

Figure 4: *Content of the virtual directory*

myimage.bmp is a thumbs up image that will be provided, secret.txt contains "UPUPDOWNDOWNLEFTRIGHTLEFTRIGHTBASTART" and mytext.txt contains "Irasshaimase".

4.3 User authentication

Anonymous login must be enabled on your server. The user "Sam" is also allowed access with password "123456".

Sam and the anonymous users have access to the same files/folders, except for the folder "private" which only Sam can see and use.

4.4 Launch

The server software is invoked on the command line with one additional argument:
`java FTPServer maxThreads`
where *maxThreads* is the maximum number of java Threads that can be run in a concurrent way to handle the requests. One can also see this argument as the maximum number of sessions that can be treated "simultaneously" (see section 3).

The server listens on port $21xx$ for the control connection, and uses port $20xx$ to initiate data connections in active mode– where xx is your group ID (assigned by the Montefiore Submission Platform).

4.5 Incorrect or incomplete commands

When dealing with network connections, you can never assume that the other side will behave as you expect.

It is thus your responsibility to check the validity of the client's request (and respond with the correct status code) and to ensure that a malevolent person won't make your server freeze by initiating a TCP connection and keep it open for an indefinite period of time (see Section 3 for a counter-measure).

5 Guidelines

- You will implement the programs using Java 1.8, with packages `java.lang`, `java.io`, `java.net`, `java.awt`, `javax.imageIO` and `java.util`[2],
- You will ensure that your program can be terminated at any time simply using CTRL+C, and avoid the use of ShutdownHooks
- You will not manipulate any file on the local file system.
- You will ensure your main class is named FTPServer, located in FTPServer.java at the root of the archive, and does not contain any `package` instruction. You are allowed to create more classes and java files than the imposed one.
- You will not cluster your program in packages or directories. All java files should be found in the same directory.
- You will ensure that your program is fully operational (i.e. compilation and execution) on the student's machines in the lab (ms8xx.montefiore.ulg.ac.be).

Submissions that do not observe these guidelines could be ignored during evaluation.

[2]Don't be shy and ask for more packages if you feel like there's one you would really like to use

Your commented source code will be delivered no later than December, 15th to the Montefiore Submission platform (`http://submit.run.montefiore.ulg.ac.be/`) as a .zip package.

Your program will be completed with a .pdf report (in the zip package) addressing the following points:

Software architecture: How have you broken down the problem to come to the solution? Name the major classes responsible for requests processing. A diagram of classes is highly recommended.

Program Logic: How does your server work? Sequentially describe which operations it performs and mention the location in your source code where a given operation is implemented.

FTP protocol: How far did you go into the implementation of FTP? Provide a comprehensive list of requests that your server can handle and the possible reply codes that your server might send.

TCP stream : How did you read the requests from the TCP stream? You are allowed to provide the source code that handles this operation.

Multi-thread organisation: How did you achieve multi-threading in your program? You may also include source code to justify. Did you synchronize the activity of the different threads? If so, what content did you protect?

Robustness: How did you make your server robust? How did you manage the different types of exceptions that your server might receive?

6 How to start

Considering that this assignment is a bit more complex than the previous one, some of you might encounter difficulties to begin the work.

A good way of understanding how something works is to see it in action. In a few minutes, one can quite easily find a free-to-use FTP server (e.g. `ftp://speedtest.tele2.net/`). I suggest you to connect to this FTP server, and monitor, using Wireshark or tcpdump, the network exchanges between the client and the server.

For your program, do not try to implement everything at once, but rather develop in an incremental manner. Start with a server with minimal functionality; for instance it would only accept the `USER` request; and work your way up by adding more and more functionalities (and test your program between each added feature).

Good programming...