CS 421 – Computer Networks
Programming Assignment I
SeekAndDestroy

Due: April 4, 2019

I. Introduction

In this programming assignment, you are asked to implement a program in **Java**. The program must search for a file in the server, download it, and then delete it from the server. Your program must communicate with the server by using a custom application layer protocol named *CustomFTP*, which is inspired by the File Transfer Protocol (FTP). The server program written and tested in **Python 3.6** (to avoid decompiling to Java source code to use in the assignment) is provided for you to test your program.

The goal of the assignment is to make you familiar with the application layer and TCP sockets. You must implement your program using the **Java Socket API of the JDK**. If you have any doubt about what to use or not to use, please contact your teaching assistant.

When preparing your project please keep in mind that your projects will be auto-graded by a computer program. Any problems in the formatting can create problems in the grading; while you will not get a zero from your project, you may need to have an appointment with your teaching assistant for a manual grading. Errors caused by **incorrectly naming** the project files and folder structure will cause you to lose points.

II. Design Specifications

a. *CustomFTP* Specifications

The server program we provide, *CustomFTPServer*, uses an application layer protocol called *CustomFTP.* CustomFTP is a simplified file transfer protocol which is inspired by FTP.

i. Control Connection

All commands are sent from the client (your program) to the server through the *control connection.* After each command, the server sends a response to the client again using the same control connection. This control connection is persistent, i.e., it stays open throughout the session. The commands must be constructed as strings encoded in **US-ASCII** and have the following format:

`<CommandName><Space><Argument><CR><LF>`

- `<CommandName>` is the name of the command. For all possible commands and their explanations, see Table 1.
- `<Space>` is a single space character. Can be omitted if `<Argument>` is empty.
- `<Argument>` is the argument specific to the command. Can be empty if no argument is needed for the given command. See Table 1 for more information.
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., "\r\n".

Table 1: List of CustomFTP commands with explanations.

<CommandName>	**<Argument>**	**Explanation**	**Example Usage**
USER	<username>	Send username to the server.	USER bilkent\r\n
PASS	<password>	Send password to the server.	PASS cs421\r\n
PORT	<port>	Send port number of the data connection that the client is bound to.	PORT 60001\r\n
NLST	-	Obtain the list of all files and directories in the current working folder of the server.	NLST\r\n
CWD	<child>	Change the working directory to the one of the child directories of the current working directory of the server.	CWD images\r\n

CDUP	–	Go to the parent directory of the current working directory of the server.	CDUP\r\n
RETR	<filename>	Retrieve the file <filename> from the current working directory of the server.	RETR sample.txt\r\n
DELE	<filename>	Delete the file <filename> from the current working directory of the server.	DELE sample.txt\r\n
QUIT	–	Tell server to end the session and shutdown.	QUIT\r\n

The response messages sent by the server are also encoded in **US-ASCII**. Responses have the following format:

<Code><Message><CR><LF>

- <Code> is either 200 (success) or 400 (failure), for the sake of simplicity. You should check the <Message> for the reason of the failure if <Code> is 400.
- <Message> is the response message. It is always "OK" when <Code> is 200.
- <CR><LF> is a carriage return character followed by a line feed character, i.e., "\r\n".

ii. Data Connection

While commands and responses are exchanged through a persistent control connection, the data (e.g., files obtained using RETR command or list of files and directories obtained using NLST command) is sent through a non-persistent connection named *data connection*. That is, the connection is opened just before the data transmission and closed once the transmission is complete.

CustomFTP uses what is called *active mode* in FTP. In active mode, the client binds to an available port for the data connection and sends this port to the server using the PORT command, at the beginning of the session. Then, when a data transfer is about the begin, the server connects to the client at the specified port and the transmission begins.

For data transmission mode, CustomFTP uses *block mode*. In block mode, the data is sent in the form of a block with a 16-bit header in addition to the data. This header indicates the size of the data to be transmitted (see Figure 1). Byte order of the header is in big-endian format, i.e., the most significant byte is sent first, the least significant byte is sent last.

Figure 1: Illustration of data transmission in block mode.

b. *SeekAndDestroy* Design Specifications

The server program provided for you, *CustomFTPServer,* simulates a server with some randomly generated directories and files. When a client connects to the server, the current working directory is the "root" folder. The client can view the contents of the current working directory by using the `NLST` command. The `NLST` command retrieves the contents of the current working directory as a US-ASCII encoded string using the data connection in the following form:

`<name1>:<type1><CR><LF><name2>:<type2><CR><LF>...<nameN>:<typeN>`

- `<nameX>` is the name of the `X`th directory/file in the list.
- `<typeX>` is the type of the `X`th directory/file in the list. It is `d` if `X` is a directory, `f` if it is a file.
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., "\r\n".

Note that there is no extra `<CR><LF>` at the end. Thus, if the current directory is empty, `NLST` does not send anything but a 16-bit header of zeros. Some examples strings that can be retrieved after an `NLST` command are as follows:

```
cheesecake:d\r\nfuton.txt:f\r\ntarget.jpg:f\r\npocket:d
research:d
bear.png:f
```

Note that `cheesecake`, `pocket` and `research` are directories, `futon.txt`, `target.jpg` and `bear.png` are files in the examples above.

After listing the contents, the client can descend into any subdirectory from the received list by using the `CWD` command with the subdirectory name as the argument. It can go back to

the parent directory with `CDUP` command. Basically, `NLST`, `CWD` and `CDUP` is the way to traverse the directories in the server.

The program you are asked to implement, *SeekAndDestroy*, is expected to find a file named "**target.jpg**" in the server. The exact location of this file is determined **randomly** at each run of the server. Once the file is found, SeekAndDestroy must download it, then delete it from the server. The following list of steps explains the job expected from your program in full detail:

1. Start the control connection by connecting to the server from the control port.
2. Send a `USER` command with the username **bilkent**.
3. Send a `PASS` command with the password **cs421**.
4. Bind to an available port on your machine for the data connection. Send this port number to the server with a `PORT` command.
5. Use a combination of `NLST`, `CWD` and `CDUP` commands to search for the file **target.jpg**. There is no restriction or requirement on how you search for the target file; you can implement any search technique as long as it finds the file.
6. Download the file from the server using a `RETR` command. Note that once you send a `RETR` command successfully, the server will try to connect to your program from the data connection port that you are bound to. After you download the file, save it to the same directory as your program with the name **received.jpg**.
7. Delete the file from the server using a `DELE` command.
8. Send a `QUIT` command to end the session.

A few important tips and notes that you should keep in mind:

- Check the response message after sending each command and debug your program according to the message. The server shuts itself down if it sends a failure response.
- Compare received.jpg with target.jpg to see if you received the file correctly.
- `The RETR` command deletes the file from the server's "artificial" disk, but you will still see target.jpg in the same directory as the server code (it is required for the server code to work).

c. Running the Server Program

The server program we provide is written and tested in **Python 3.6**. You are also provided an image file named target.jpg, which is required for the server program to work. You must put target.jpg in the **same directory** as CustomFTPServer.py and start the server program **before** running your own program using the following command:

```
python CustomFTPServer.py <Addr> <ControlPort>
```

where "< >" denotes command-line arguments. These command-line arguments are:

- `<Addr>` [Required] The IP address of the server. Since you will be running both your program and the server program on your own machine you should use `127.0.0.1` or `localhost` for this argument.
- `<ControlPort>` [Required] The control port to which the server will bind. Your program should connect to the server from this port to send the control commands.

Example:

```
python CustomFTPServer.py 127.0.0.1 60000
```

The command above starts the server with IP `127.0.0.1`, i.e., localhost, which uses port `60000` for the control commands.

d. Running *SeekAndDestroy*

Your program must be **a console application** (no graphical user interface, GUI, is allowed) and should be named as `SeekAndDestroy.java` (i.e., the name of the class that includes the `main` method should be `SeekAndDestroy`). Your program should run with the command

```
java SeekAndDestroy <Addr> <ControlPort>
```

where "< >" denotes command-line arguments. These arguments must be the same as the arguments for the server program, which are explained above.

Example:

```
java SeekAndDestroy 127.0.0.1 60000
```

In this example, the program connects to the server with IP `127.0.0.1`, i.e., localhost, on port `60000`. Please note that you must run your program **after** you start the server program.

III. Example

In this part, we provide you a simple example showing a full session between SeekAndDestroy and CustomFTPServer for clarification. Figure 2 shows the organization of the files and the directories in the server in a tree structure. Table 2 shows all the interactions between SeekAndDestroy and CustomFTPServer throughout the session. Note that the server responses received from the control connection are not shown in this example. Also note that the search technique used here is just an example; you can search for the target using a combination of `NLST`, `CWD` and `CDUP` commands in whichever fashion you like.

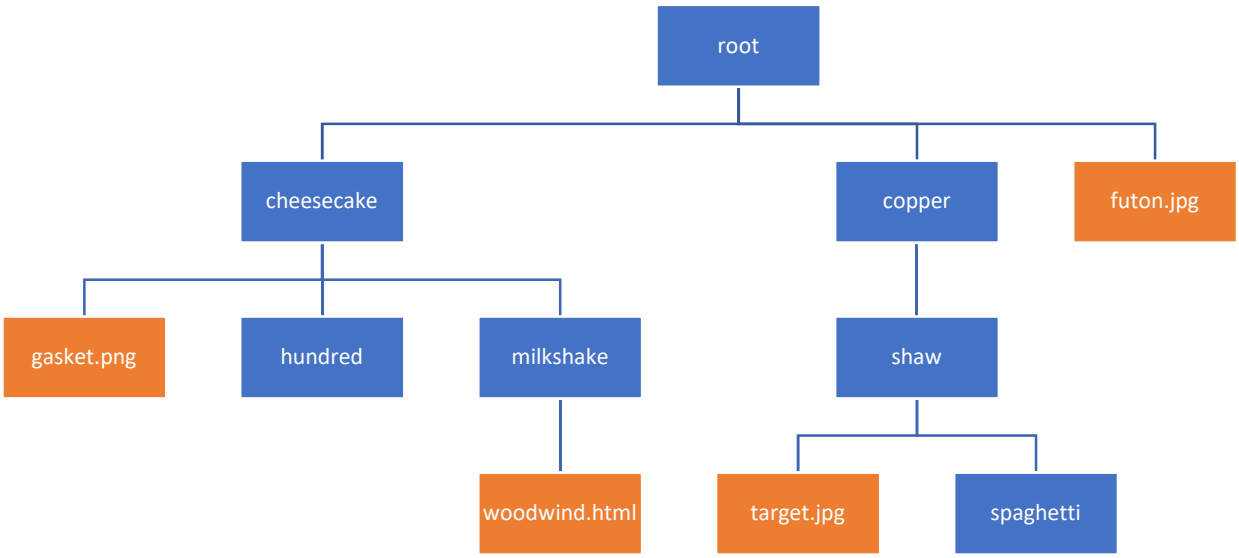

Figure 2: Organization of the files and the directories in the server for the given example. Directories are shown in blue, files are shown in orange.

Table 2: An example session.

Time	Action	Received Data	Comments
1	Connect to the server from the control port.		
2	Send command: `USER bilkent\r\n`		
3	Send command: `PASS cs421\r\n`		
4	Start listening for incoming data connection requests at an available port. Let that port number be 53462 in this example.		
5	Send command: `PORT 53462\r\n`		

6	Send command: `NLST\r\n`	`cheesecake:d\r\ncopper:d\r\nfuton.jpg:f`	Directories: `cheesecake`, `copper` File: `futon.jpg` Since target is not found, we should descend into the other directories and search there.
7	Send command: `CWD cheesecake\r\n`		Current working directory is switched from `root` to `cheesecake`.
8	Send command: `NLST\r\n`	`gasket.png:f\r\nhundred:d\r\nmilkshake:d`	Still no target. We should look somewhere else.
9	Send command: `CWD hundred\r\n`		Current working directory: `hundred`.
10	Send command: `NLST\r\n`	<Nothing but a 16-bit header of 0's received>	Directory is empty.
11	Send command: `CDUP\r\n`		We should go back to the parent directory. Current working directory: `cheesecake`.
12	Send command: `CWD milkshake\r\n`		Current working directory: `milkshake`.
13	Send command: `NLST\r\n`	`woodwind.html:f`	Target not found. No subdirectories exist either.
14	Send command: `CDUP\r\n`		Go up. Current working directory: `cheesecake`.
15	Send command: `CDUP\r\n`		We searched for all the subdirectories but did not find the target. We should go up. Current working directory: `root`.
16	Send command: `CWD copper\r\n`		Current working directory: `copper`.
17	Send command: `NLST\r\n`	`shaw:d`	
18	Send command: `CWD shaw\r\n`		Current working directory: `shaw`.
19	Send command: `NLST\r\n`	`target.jpg:f\r\nspaghetti:d`	Target located.
20	Send command: `RETR target.jpg\r\n`	<contents of the file target.jpg>	Target file downloaded.

21	Save the received file to the disk with name “received.jpg”.		
22	Send command: `DELE target.jpg\r\n`		Delete the file from the server.
23	Send command: `QUIT \r\n`		End the session.

Final Remarks

- Please contact your teaching assistant if you have any doubt about the assignment.
- **Do not forget to check the response message after sending each command** to see if your code is working and debug it if it is not. Note that the server cannot detect all the errors that you make; therefore, you might have to experiment a bit to correct all your errors.
- You can modify the source code of the server for experimental purposes. However, do not forget that your projects will be evaluated based on the version we provide.
- We have tested that these programs work with the discussed Java-Python combination.
- You might receive some socket exceptions if your program fails to close sockets from its previous instance. In that case, you can manually shut down those ports by waiting for them to timeout, restarting the machine, etc.
- Remember that all the CustomFTP commands must be constructed as **strings** and encoded with **US-ASCII** encoding.
- Remember that all data (i.e., directory listings and files) are sent in block mode with a **16-bit (2 bytes)** header in **big-endian** format.
- Remember that the control connection and data connection are separate. All commands and responses are exchanged using the control connection, which stays active throughout the session, while all data is sent through the data connection which is closed immediately after the data transfer is completed.

Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above.**

- The assignment should be submitted as an e-mail attachment sent to `bulut.aygunes[at]bilkent.edu.tr`. Any other methods (Disk/CD/DVD) of submission will not be accepted.
- The subject of the e-mail should start with `[CS421_2019SPRING_PA1]`, and include your name and student ID. For example, the subject line must be

  `[CS421_2019SPRING_PA1]AliVelioglu20141222`

  if your name and ID are `Ali Velioglu` and `20141222`, respectively. If you are submitting an assignment done by two students, the subject line should include the names and IDs of both group members. The subject of the e-mail should be

  `[CS421_2019SPRING_PA1]AliVelioglu20141222AyseFatmaoglu20255666`

  if group members are `Ali Velioglu` and `Ayse Fatmaoglu` with IDs `20141222` and `20255666`, respectively.
- All the files must be submitted in a zip file whose name is the same as the subject line **except the `[CS421_2019SPRING_PA1]` part**. The file must be a `.zip` file, not a `.rar` file, or any other compressed file.
- All the files must be in **the root of the zip file**; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain **any file other than the source code(s) with .java extension.**

The standard rules for plagiarism and academic honesty apply; if in doubt refer to the 'Student Disciplinary Rules and Regulations', items 7.j, 8.l and 8.m.