

CS 421 – Computer Networks

Programming Assignment II

CustomFTP Server

Due: May 9, 2019

I. Introduction

In this programming assignment, you are asked to implement the CustomFTP server which will be able to serve multiple clients at once.

The goal of the assignment is to make you familiar with concurrent networking using threads and stateful servers. 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

You will implement the server program which will handle CustomFTP request from a CustomFTP client we will provide. This CustomFTP protocol is very similar to the one we provided in the first programming assignment.

i . Control Connection

All commands are sent from the client (i.e., the program we provide) to the server (i.e., the program you will provide) 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><Space><Data><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.
- `<Data>` is only used in `PUT` command. The data format is explained in Fig. 1
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., “`\r\n`”.

<CommandName>	<Argument>	Explanation	Example Usage
PORT	<code><port></code>	Send port number of the data connection that the client is bound to.	<code>PORT 60001\r\n</code>
GPRT	–	Get the port of the server through a data connection.	<code>GPRT\r\n</code>
NLST	–	Obtain the list of all files and directories in the current working folder of the server.	<code>NLST\r\n</code>
CWD	<code><child></code>	Change the working directory to the one of the child directories of the current working directory of the server.	<code>CWD images\r\n</code>

CDUP	-	Go to the parent directory of the current working directory of the server.	CDUP\r\n
PUT	<filename>	Put the file <filename> to the current working directory of the server	PUT Image.jpg\r\n
MKDR	<foldername>	Create the new folder <foldername> in the current working directory of the server.	MKDR imagefolder\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
DDIR	<foldername>	Delete the folder <foldername> from the current working directory of the server.	DDIR imagefolder\r\n
QUIT	-	Tell server to end the session and shutdown.	QUIT\r\n

Your program must send response messages which are encoded in US-ASCII. Responses have the following format:

<Code><CR><LF>

- `<Code>` is either 200 (success) or 400 (failure), for the sake of simplicity. Your server should only send the success or failure codes and nothing else.
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., `"\r\n"`.

The specifications which show when your program should send failure is described in Table 2.

<code><CommandName></code>	Fails if:
PUT	File with same name already exists.
MKDR	Folder with same name already exists.
DELE	File does not exist in the current working directory.
DDIR	Folder does not exist in the current working directory.
CWD	Folder does not exist in the current working directory.
CDUP	Current working directory is the root folder.

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 to 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.

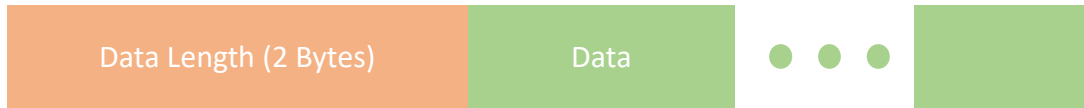


Figure 1: Illustration of data transmission in block mode. Note that the maximum data length is 65535 Bytes

b. CustomFTPServer Design Specifications

The client program provided for you, `CustomFTPClient`, is a simple client which attempts to connect and create a pre-determined folder structure in the folder which the `CustomFTPServer` is running. In the test program, multiple clients will connect to your `CustomFTP` server simultaneously and each will create their folder structures. The clients will utilize the commands described here so your server must implement them correctly to pass the test routines. Below there are detailed explanations of the important commands and expected behavior from your server program.

i. GPRT

The `GPRT` command retrieves the data socket of your server program for the client since the client has to use your data-port for `PUT` command. The format of the expected data response is the same as all data connections (2-byte header and data). You should however send the port number encoded as US-ASCII string, e.g., port 60000 is 5 bytes of ASCII.

The general format of the message is in the following form: `<portNumber>`

- `<portNumber>` is the randomly selected port of the server.

ii. NLST

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"`.

- Your program should connect to the client using its port number previously received when the connection was first established.
- Your program should terminate this connection after sending the data.

Note that there is no extra `<CR><LF>` at the end. Thus, if the current directory is empty, `NLST` should not send anything but a 16-bit header of zeros. Some example strings that will be sent by your program after receiving an `NLST` command are as follows:

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

iii. RETR

When your program receives the `RETR` command, it should try to connect to the client with its address and the port number it received with the `PORT` command. For example, consider the client is operating on `localhost` and has sent a `PORT` command with `60002` as its argument when the connection was first established. When your program receives the command `RETR note.txt`, it will first connect to the client at `localhost:60002` and it will send the file using the *block mode*, i.e., with 2-byte data-length header and the data following the header.

- Your program should terminate the connection after sending the data.

iv. PUT

The `put` command is used for transferring a file from the client to the server. An example of a `PUT` command is below.

```
PUT image.jpg<CR><LF>
```

When a `PUT` command is received, your server should start to listen to its data port (similar to `RETR` and `NLST` for client). Do not forget to send responses for the commands prior to the data connection.

c. Running your Server

We provide a testing program called `ServerTester.py` which tests your CustomFTP server for conformity with the specifications. You must execute this program while your CustomFTP server is running. For testing, we provide a sample directory structure, which your CustomFTP server will “serve”. You must run your server at the root folder of this folder structure in the folder called “Testing”.

When started, the clients we provide will automatically try to access and modify the folder contents; for tests we will use multiple clients.

Your program should run with the command

```
java CustomFTPServer <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.
- You must run your server before the test program.
- You must run your server in the provided `ServerFolder`.

d. Running the Tests

The test program called `CS421PA2Tester.py` is provided to you. These programs will try to perform certain operations using your `CustomFTPServer`.

You should run the testing program with the following command:


```
python CS421PA2Tester.py
```

This test program will spawn 2 CustomFTPClient's which would try to use your server simultaneously.

- You must run this test program in the provided `TestFolder`.

- For your convenience we provide another program called “CustomFTPConsole” which you can use to debug your program. This is a small “REPL” interface for the CustomFTP. Note that it’s a dummy interface, the commands PUT and RETR do not alter anything on the machine.
- Test programs are expecting your servers control port to bind to `localhost:60000`.

Final Remarks

- Please contact your teaching assistant if you have any doubt about the assignment.
- Note that we have omitted the `USER` and `PASS` commands from the previous assignment.
- The main goal of this project is for you to provide a CustomFTP server with multithreading. Therefore, you can assume that the main part of the grade will come from this aspect. As a guideline, prioritize according to these: (from most important to least important) multithreading, correct behavior for correct usage, error handling for incorrect usage, and boundary cases.
- **You can and should refer to PA1 for more detailed explanations of the previous behavior.**
- You can modify the source code of the clients 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 `yarkin.cetin[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_PA2]`, and include your name and student ID. For example, the subject line must be
`[CS421_2019SPRING_PA2]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_PA2]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_PA2]` 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.