CS 421 – Computer Networks
Programming Assignment I
TextEditor: Client for Collaborative Text Editing
Due: April 7, 2020 at 11:59PM

1) Introduction

In this programming assignment, you are asked to implement a program in **Java**. You should code a client that can collaboratively edit a text (.txt) file, which is located at a server, together with other clients. The server program written and tested in Python 3.7 (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 result in point deductions.

2) Specifications

"TextEditor" uses a custom application level protocol which is built on top of TCP. There is a .txt file provided to you with the server source code. Several clients (in your case, multiple clients can be run on different consoles at the same host) should be able to edit the text file together. Several operations and specifications are present for this case. In program lifecycle, first an authentication check will be done. A specific username and password will be looked after by the server. Authenticated clients can then edit the text by

using several commands that are described below. It is very important that a client (Client A) cannot change the text file without getting an update first, since the file may already been changed by another client (Client B) after the last update of Client A. If a new version of the file is present for Client A, that file should be seen as "version conflict" by Client A using the version information associated to the file. Each change by either of the clients results in an increment to the file version. If the client side version does NOT match with that of the server, the client can deduce that a change has been made in the file. Initially, server program sets the file version to 0 and increments it after each successful APND or WRTE operation. UPDT operation doesn't change the file version. Upon connecting to the server, a client should first download the text file by using the UPDT command. As already mentioned, in order to modify an already modified document, a preliminary UPDT operation must be made.

3) Connection Formats

i) Commands

In order to fulfill the program specifications, your client will need to use specific commands to communicate with the server. Commands should be strings encoded in **US-ASCII**. The format is below:

<MessageType><Space><Arguments><CR><LF>

- <MessageType> is the name of the command. For all possible commands and their explanations, see Figure 2.
- <Space> is a single space character. Can be omitted if <Argument> is empty.
- <Argument> is the argument specific to the command. All arguments include version of client side. See Figure 1 for more information.
- <CR><LF> is a carriage return character followed by a line feed character, i.e., "\r\n".

MessageType	Arguments	Definition	Example
USER	<username>	Send the username for authentication	USER bilkentstu\r\n
PASS	<password>	Send the password for authentication	PASS cs421s2020\r\n
WRTE	<version><Space><linenumber><Space><text>	Write arbitrary text on the specific line <linenumber> in the file. <version> is the last value for the version of the file known by the client .	WRTE 12 5 "The Dark Side of the Moon"\r\n
APND	<version><Space><text>	Append a new line to the file. <version> is the last value for the version of the file known by the client .	APND 12 "Train of Thought"\r\n
UPDT	<version>	Get up-to-date file. <version> is the last value for the version of the file known by the client .	UPDT 12\r\n
EXIT	-	Terminate the program	EXIT\r\n

Figure 1 List of commands with examples

ii) Responses

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

<Code><Space><version><Space><Response><CR><LF>

- <Code> is either OK (success) or INVALID (failure), for the sake of simplicity. You should check the <Response> part if <Code> is INVALID or the response belongs to UPDT which contains the current state of the file.

- <version> is the number specifying the current version of the document. <version> is incremented by the server after each change in the document. If a discrepancy between the server and the local version is observed by the client, then an update is required at the client side.
- <Response> is the response message. Detailed responses to erroneous operations as well as the up-to-date file at the server are returned within this part.
- <CR><LF> is a carriage return character followed by a line feed character, i.e., "\r\n".

For example, for "WRTE" command, the response might report the "Version Conflict" status when the version reported by the client does NOT match with that of the server.

4) Running the server program

The server program we provide is written and tested in **Python 3.7**. You should start the server program **before** running your client program using the following command:

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

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

- <Addr> 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> 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:

```
C:\Users\user\Desktop\bilkentedu\cs421\CS421_2020SPRING_PA1>python Server.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.

5) Running the TextEditor

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

```
java TextEditor <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:

```
C:\Users\user\Desktop\bilkentedu\cs421\CS421_2020SPRING_PA1>java TextEditor 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.

Do not forget to run more than one client (two is enough) from multiple command prompts. Same source code can be used as clients.

6) 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.

- 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 commands must be constructed as strings and encoded with US-ASCII encoding.
- Use big-endian format if it is necessary to use.
- **Please put the folder containing the images under the same directory with the server and client codes.**

7) 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 **mehmet.sakaoglu[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_2020SPRING_PA1], and include your name and student ID. For example, the subject line must be [CS421_2020SPRING_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_2020SPRING_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_2020SPRING_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 archive should not contain these:
    - Any class files or other executables,
    - Any third-party library archives (i.e., jar files),
    - Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans, etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.
- **The standard rules for plagiarism and academic honesty apply. Your submitted codes will be carefully checked and disciplinary actions will be taken in case of plagiarism.**