CS 421 – Computer Networks
Programming Assignment I
TaxonomySearcher

Due: 29.10.2020

1) Introduction

In this programming assignment, you are asked to implement a program in **Java**. You should code a client traversing a folder structure existing on the server and download only the required images found under the folders after an authentication step. Client program must communicate with the server by using the protocol defined in this document. 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 project will be auto-graded by a computer program. Problems in formatting may also create problems in the grading; although you will not get a zero from your project, you may need to have an appointment with your teaching assistant for manual grading. Errors caused by incorrect naming of the project files and folder structure will cause you to lose points.

2) Specifications

"TaxonomySearcher" uses a custom application level protocol which is built on top of TCP in order to communicate with the server. At the server side, there is a folder structure randomly created upon launching the program. Folder names reflect a contextual hierarchy starting from a rather general category to more and more specific ones as you go deeper into the folder hierarchy (For example, the "Giraffes" folder will be under the "Animals" folder). This

structure is randomized but it is tree-like nonetheless. You can traverse in this folder structure by the commands specified by the TaxonomySearcher protocol. Under each leaf (bottommost) folder, there may be a file belonging to the category of that folder. Your task is first to acquire the list of files to be downloaded and then download them by traversing the folder structure. **Note that you must download only the files on the list obtained from the server.** Both the folder structure and the files are randomized at each run. You can run the server code first to see a sample folder structure generated before writing the client side. The program will always start in the root directory under which the server code resides. You can terminate the program after all required images are downloaded successfully. The flowchart below can help you understand the overall process.

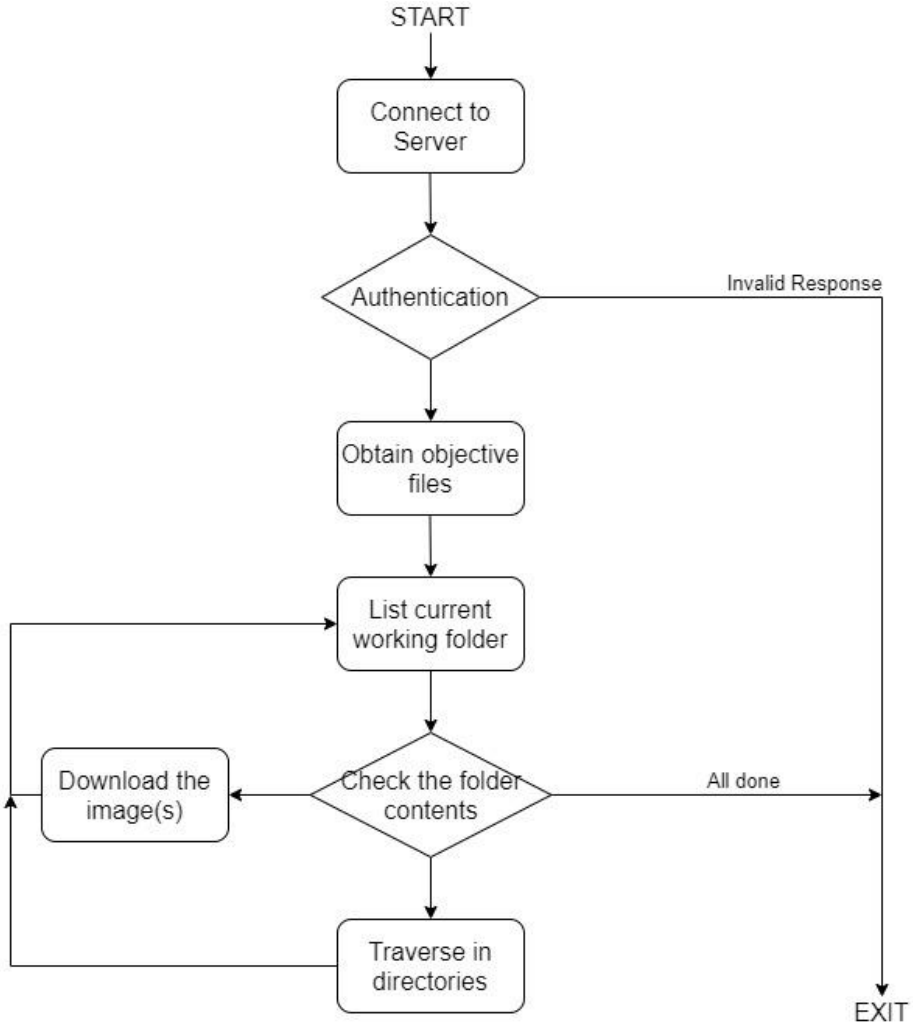

Figure 1 Client process flowchart

3) Connection Formats

i) Commands

You will need to use specific commands to communicate with the server. Commands should be strings encoded in **US-ASCII**. The format is as shown 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 `<Arguments>` is empty.
- `<Arguments>` is the argument specific to the command. Can be empty if no argument is needed for the given command. See Figure 2 for more information.
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., "\r\n".

MessageType	Arguments	Message Definition	Message Example
USER	<username>	Sends the username for authentication	USER bilkentstu\r\n
PASS	<password>	Sends the password for authentication	PASS cs421f2020\r\n
OBJ	-	Obtains the list of files required to be downloaded by the client.	OBJ\r\n
NLST	-	Obtains the list of all files and folders under the current working directory of the server.	NLST\r\n
CWDR	<child>	Changes the working directory to one of the child directories of the current working directory of the server.	CWDR animal\r\n

CDUP	-	Goes to the parent directory of the current working directory of the server.	CDUP\r\n
GET	<filename>	Gets the image from the server.	GET cat.jpg\r\n
EXIT	-	Terminates the program	EXIT\r\n

Figure 2 List of commands with examples

ii) Responses

The response messages sent by the server are also encoded in **US-ASCII**. Responses (except for "GET" command) have the following format:

<StatusCode><Space><StatusText><CR><LF>

- <StatusCode> is either OK (success) or INVALID (failure), for the sake of simplicity.
- <StatusText> is the response message given by the server.
    - You should check the <StatusText> for the reason of the failure if <StatusCode> is INVALID.
    - In response to NLST and OBJ commands, <StatusText> is the list of file and folder names if the <StatusCode> is OK.
- <CR><LF> is a carriage return character followed by a line feed character, i.e., "\r\n".

For "GET" command, response is a message with the following format:

< StatusCode><ImageSize><ImageData>

- <StatusCode> is "ISND" for this response different from the usual responses.
- <ImageSize> is the size of the image **in bytes**. By using <ImageSize> you can spot the end point of <ImageData> which is of variable size. Size of this field is 3 bytes. Maximum size of an image can be 131.072 kilobytes.
- <ImageData> is the image file itself.

4) Search within Folders

Images provided to you with the assignment have categorical names (bear.jpg, pine.jpg, etc.). **We expect your main function to conduct an automated search on the server-side folders, by traversing these folders and getting the objective files without requiring any manually entered commands. If your client code uses hard-coded folder and file names, it will be penalized accordingly.**

5) Running the server program

The provided server program is written and tested in **Python 3.7**. You are also provided with the images in a folder, which are required for the server program to work properly. You must put the image folder in the **same directory** as Server.py and start the server program **before** running your own client program using the following command:

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

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

- <Addr> The IP address of the server. Since you will be running both the client and the server program on your own machine, you should use 127.0.0.1 or localhost for this argument.
- <Port> The port number to which the server will bind. Your program should connect to the server from this port to send commands and download files.

Example:

```
C:\Users\user\Desktop\CS421_2020FALL_PA1>python Server.py 127.0.0.1 60000
```

The command above starts the server with IP address 127.0.0.1, i.e., localhost and port number 60000.

6) Running the TaxonomySearcher

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

```
java TaxonomySearcher <Addr> <Port>
```

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\CS421_2020FALL_PA1>java TaxonomySearcher 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**.

An example client code run snippet can be seen below.

```
Sending:OBJ
OK eagle.jpg orange.jpg harvester.jpg
Sending:NLST
OK animal plant
Sending:CWDR animal
OK
Sending:NLST
OK arthropoda chordata
Sending:CWDR arthropoda
OK
Sending:NLST
OK insect shellfish spider
Sending:CDUP
OK
Sending:CWDR chordata
OK
Sending:NLST
OK bird mammal
Sending:CWDR bird
OK
Sending:NLST
OK eagle pigeon read.py
Sending:CWDR eagle
OK
Sending:GET eagle.jpg
ISND
```

Figure 3 Example client code run

You should clearly demonstrate the flow of the client-side by printing the sent and received messages to the Command Prompt as shown in Figure 3. No or partial demonstration of commands and responses will be penalized.

7) 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 to debug otherwise. 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 provide to you as part of this assignment.
- 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 whenever necessary.
- **Do NOT forget to put the folder containing the images under the same directory with the server and client codes.**

8) Submission rules

You are required to comply with the following rules in your submission. You will lose points otherwise or if your program does not run as described in the assignment above.

- All the files must be submitted in a zip file (not a .rar file, or any other compressed file) whose name must include your name and student ID. For example, the file name

must be “AliVelioglu20141222” if your name and ID are “Ali Velioglu” and “20141222”, respectively.

- Grouping with another student from the same section is allowed.
    - If you are submitting as a group of two students, the file name must include the names and IDs of both group members. For example, the file name must be “AliVelioglu20141222AyseFatmaoglu20255666” if group members are “Ali Velioglu” and “Ayse Fatmaoglu” with IDs “20141222” and “20255666”, respectively.
    - For group submissions ONLY one member must make the submission. The other member must NOT make a submission.
- 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; if in doubt refer to the ‘Student Disciplinary Rules and Regulations’, items 7.j, 8.l and 8.m.