CS 436 - Project: Network programming (HTTP, TCP protocols) 100 points

This assignment is to be done in two-person groups. Do not show your code to other groups and do not look at other students' code. Prevent other students from accessing your code (do not put it in a public directory). However, you are free to use any tools you like (library, search engines), to discuss problems with others, and to seek the help of the instructor.

1. Project

The following is the project I assign to the class. **You can either do this project or define another project**. If you define a project, you need to get the professor's approval. Some topics include:

1. Chatting between multiple clients through a server.
2. Communication between an IoT device, like a Raspberry Pi, and the client program to exchange data through a server on the cloud.
3. A blockchain application where the communicates with the blockchain server.
4. Secure communications between two IoT or mobile devices.
5. A program that automatically communicates with public servers like GitHub or YouTube. You should define an application for such communication as well.

2. Python3

You will implement this project in Python3. How to learn Python3? There are many great tutorials on the Internet. The following is a link to a very easy and helpful tutorial for beginners with hands-on examples in the popular PyCharm IDE. Study the first 2 hours of the video.
https://www.youtube.com/watch?v=_uQrJ0TkZlc&t=16696s

The playlist tutorial by Corey Schafer is also great to learn Python with examples.
https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU

3. Socket programming in Python

Refer to the socket programming lecture and video. Also, a simple socket programming example is given in the project section of the course. Download the attached folder. I have also provided an instruction to run the files in the project section. Implement this project with **UDP**.

4. Project Objective

In this project, you will develop a client-server application that implements a simplified forwarding protocol using socket programming in Python language.

5. Implement HTTP and TCP protocols

In this project, we implement HTTP and TCP protocols.

1. We implement two programs: a client and a server. We name them “client.py” and “server.py”. Use port number 18000 for the server. While you are testing your program, if you got any issue with this port number, feel free to change it to 18001 or above.
2. First run the server program. Then we run the client program. The client program displays a menu to the user and the user enters one of the options.
    1. Get a file from the server.
    2. Quit the program.
3. Assume the user selects the first item: “1. Get a file from the server.” Then the program displays the second menu, and the user enters one of the options.
    1. Use HTTP version 1.0 (non-persistent).
    2. Use HTTP version 1.1 (persistent).
4. Then the client program wants to send an HTTP GET request to the server program. However, it first needs to establish a TCP connection with the server.
5. The client creates a TCP SYN message and sends it to the server. The message structure is given in the next section. Set TCP_SYN_FLAG to 1. The client displays a message on the screen reporting that it has sent a TCP SYN request to the server.
6. The server receives that and responds with a TCP SYN ACK message. Set TCP_SYN_FLAG and TCP_ACK_FLAG to 1. The server displays a message on the screen reporting what it has done.
7. The client receives that and responds with a TCP ACK message. The client displays a message on the screen reporting what it has done. Now the client and server are done with TCP connection establishment.
8. The client sends an HTTP GET request to the server and enters the path to the file at server. The client also enters the HTTP version that the user has entered in the menu earlier, which is either 1.0 or 1.1. The client displays a message on the screen reporting what it has done.
9. The server receives this HTTP GET request message and takes the file from the given path. This is a relative path. For example, “file1.html” path means file1.html is stored at the same folder where “server.py” is stored, and “attachments/file1.html” path means file1.html is stored in subfolder “attachments” in the folder where “server.py” is stored. For this step, you can use the OS module in Python or other related modules. A good reference is: https://youtu.be/tJxcKyFMTGo
10. If the requested file does not exist on the server, the server sends a message with status code set to 404. The client receives that and displays “404 Page not found” message on the screen. Then the client sends a TCP FIN message to the server to close the connection. Then the server sends a TCP FIN ACK message. Then the client sends a TCP ACK message. Now the client program will show the first menu again.
11. If the requested file does exist on the server, the server sends a message with status code set to 200 to the client including the text contents of the file. If the file includes a link to an object as well, the server enters the path to that object in the “HTTP_INCLUDED_OBJECT_PATH” field of the response message. It also enters the HTTP version that the client had entered in its request, which is either 1.0 or 1.1. The server displays a message on the screen reporting what it has done.

12. The client receives the response message and displays the content of the file on the screen. It also stores the file on the client machine.
13. If the "HTTP_INCLUDED_OBJECT_PATH" field of the response message is blank, the client closes the TCP connection. The client sends a TCP FIN message, then the server sends a TCP FIN ACK, then the client sends a TCP ACK. Both client and server display messages on the screen reporting what they have done.
14. If the "HTTP_INCLUDED_OBJECT_PATH" field of the response message is NOT blank, the client should send another HTTP GET request to get this file. Now it depends whether the client initially had entered HTTP version 1.0 or 1.1.
15. If the version is 1.1, the client directly sends the HTTP GET request to the server, the server sends the contents of the new file, the client receives that and displays the file contents on the screen. It also stores the file on the client machine. Then the client closes the current TCP connection. The client sends a TCP FIN message, then the server sends a TCP FIN ACK, then the client sends a TCP ACK. Both client and server display messages on the screen reporting what they have done.
16. If the version is 1.0, the client first closes the current TCP connection. The client sends a TCP FIN message, then the server sends a TCP FIN ACK, then the client sends a TCP ACK. Then, the client establishes a new TCP connection. The client sends a TCP SYN message, then the server sends a TCP SYN ACK, then the client sends a TCP ACK. Then, the client sends the HTTP GET request to the server to get the contents of the new file. Then, the server sends the contents of the new file, the client receives that and displays the file contents on the screen. It also stores the file on the client machine. Then the client closes the TCP connection. The client sends a TCP FIN message, then the server sends a TCP FIN ACK, then the client sends a TCP ACK. Both client and server display messages on the screen reporting what they have done.
17. Now the client program will show the first menu again.

6. The structure of the message

In this program, the structure of a message is as following. Each message contains all these fields. If a field does not apply in a message, the value for that field will be blank. For example, in a TCP SYN message, all HTTP fields will be blank.

Note: You can add more fields or slightly change the values that each field accepts if that make the implementation easier for you.

1. PAYLOAD_LENGTH: The length of the payload. The length is 1 for SYN messages and X for a string message of length X characters, including \n.
2. TCP_SYN_FLAG: If this is a TCP SYN message, this field will be set to 1. Otherwise, it is 0.
3. TCP_ACK_FLAG: If this is a TCP ACK message, this field will be set to 1. Otherwise, it is 0.
4. TCP_FIN_FLAG: If this is a TCP FIN message, this field will be set to 1. Otherwise, it is 0.
5. HTTP_GET_REQUEST: If this is an HTTP GET request message, this field will be set to 1. Otherwise, it is 0.
6. HTTP_RESPONSE_STATUS_CODE: Either "200 OK" or "404 Page not found". If this message is NOT an HTTP response message, this field will be set to 0.

7. HTTP_CLIENT_VERSION: If this is an HTTP GET request message, this field will be set to either 1.0 (needs new TCP connections to request objects that belong to the same initial HTTP request) or 1.1 (does not need new TCP connections to request objects that belong to the same initial HTTP request). Otherwise, it is 0.
8. HTTP_ REQUEST_PATH: A string that contains the path to the requested file on the server. For example, "attachments/file1.html" refers to "file1.html" file, which is a file in the "attachments" folder at server. This folder is in the same folder where the server program is. If this message does not request a file, this field is 0.
9. HTTP_INCLUDED_OBJECT_PATH: A file at the server may include links to other objects. For example, it may include a link to an object, like an image. In this project, for simplicity, we assume the included objects are also .html files. When the client receives a file, it should check whether this file includes any link to other objects or not. But that is not easy to implement. For simplicity, we assume if this file includes another object, the server enters the path to that object in the HTTP_INCLUDED_OBJECT_PATH field. When the client receives a message with HTTP_INCLUDED_OBJECT_PATH field set to a path, it will send another HTTP GET request message to get this file. If the client's HTTP_ VERSION is 1.0, the client should close the current TCP connection, establish a new TCP connection, and then send the HTTP GET request message. However, it is 1.1, the client can directly send the HTTP GET request message and the server will reply that file.

7. Notes to test the program

Run the following steps and **take a snapshot of your terminal at each step**. Zoom in before you take the snapshots to make them more readable. Make a Report.doc file and enter the snapshots of each step to the report file. The program should run on Python3.

Before testing the program, at the folder where your server program is stored, create a folder named "attachments" and three files named file1.html, file2.html, file3.html. We assume file3.html is an object that is included in file2.html. Enter the following texts in the files.

1. The file1.html contains "<html><body> Hello, this is the first file. This file does not include any other object. </body></html>"
2. The file2.html contains ""<html><body> Hello, this is the second file. This file includes a link to the third file. </body></html>"
3. The file3.html contains ""<html><body> Hello, this is the third file. There is a link to this file in the second file. </body></html>"

8. Test your program as following:

1. Run the server program. Use port 18000 at the server.
2. Run the client program. The client program displays the menu. The user selects option 1. Then the user enters this path: "attachments/file4.html". The client program displays another menu asking what HTTP version will be used. The user selects "1.1". The client establishes a TCP connection with the server with TCP SYN messages. Then the client sends an HTTP GET request to get the file contents. The server responds with "404 File not found". The client

displays this message on the screen. The client closes the TCP connection with FIN messages. At all these steps, both client and server display messages reporting what they have done.

3. The client program displays the menu. The user selects 1. Then the user enters this path: “attachments/file1.html”. The client program displays another menu asking what HTTP version will be used. The user selects “1.1”. The client establishes a TCP connection with the server with TCP SYN messages. Then the client sends an HTTP GET request to get the file contents. The server opens the file and enters its contents to the response message with status code “200 OK”. The client displays the status code and the received file contents on the screen. The client stores the file. The client closes the TCP connection with FIN messages. At all these steps, both client and server display messages reporting what they have done.
4. The client program displays the menu. The user selects 1. Then the user enters this path: “attachments/file2.html”. The client program displays another menu asking what HTTP version will be used. The user selects “1.1”. The client establishes a TCP connection with the server with TCP SYN messages. Then the client sends an HTTP GET request to get the file contents. The server opens the file and enters its contents to the response message with status code “200 OK”. The server also sets “HTTP_INCLUDED_OBJECT_PATH” to “attachments/file3.html”. The client displays the status code and the received file contents on the screen. The client stores the file. Then the client automatically sends another HTTP GET request for “attachments/file3.html”. The server will send the contents of that file as well. The client displays the status code and the received file contents on the screen. The client stores the file. The client closes the TCP connection with FIN messages. At all these steps, both client and server display messages reporting what they have done.
5. The client program displays the menu. The user selects 1. Then the user enters this path: “attachments/file2.html”. The client program displays another menu asking what HTTP version will be used. The user selects “1.0”. The client establishes a TCP connection with the server with TCP SYN messages. Then the client sends an HTTP GET request to get the file contents. The server opens the file and enters its contents to the response message with status code “200 OK”. The server also sets “HTTP_INCLUDED_OBJECT_PATH” to “attachments/file3.html”. The client displays the status code and the received file contents on the screen. The client stores the file. Then the client closes the TCP connection with TCP FIN messages. Then, it establishes a new TCP connection with TCP SYN messages. Then, it sends another HTTP GET request for “attachments/file3.html”. The server will send the contents of that file as well. The client displays the status code and the received file contents on the screen. The client stores the file. The client closes the TCP connection with FIN messages. At all these steps, both client and server display messages reporting what they have done.

9. Submission

Submit a zip file containing the following files:

1. server.py
2. client.py
3. Report.doc

The Report file should include:

1. The team members names
2. The 5 steps of the test and snapshots of each step of test.

Put all the files in a folder named Project1_YourTeamMembers and compress it into a .zip file. Make sure your report file contains at least 5 snapshots, one snapshot of both client and server for each step. **It is important to follow the test instruction given above. Otherwise, you will miss points.**