Computer Network project's work report

Project number 1

Author: Amir Hamidi

Student Number: 9907794

November 2021

Contents

INTRODUCTION:.....2
DNS SERVER:.....3
CONSIDERATIONS:..... 5
WEB SERVER:.....7
CONSIDERATIONS:..... 7
WEB CLIENT:.....9
CONSIDERATIONS:..... 10

Introduction:

In this project, we are going to implement a web system. This web system consists of 3 main components. These components are **Web Client**, **Web Server** and **DNS Server**. Since all three of these components are fully explained in the project's description, we avoid the repentance of these information and only explain how we implement the Web system and how to work with it.

First, we are going to explain what DNS Server does and How it Does it. Following DNS Server, we will investigate Web Server and answer the same questions. Finally, we will examine Web Client and try to establish a connection between all three components. Each of these components are implemented in a separated python file and I refer to each one of them by the mentioned name. (DNS server(or DNS), Web client(or client), Web server(or Web))

In this work report I do my best to explain all the details of this project, however, I may not be able to explain what is really happening in the codes. That is why beside this PDF file, there is a video file (**named Presentation**), right next to this PDF, where I am going to explain and run the entire project by myself. This video can help you understand what is exactly happening in the core of this project. So, if you are confused by any part this report, feel free to check that video file.

Also, I should emphasize that all the works done in this project is based on my understanding of the projects. Thus, when I explain some concept in this project, this explanation might not be completely accurate, but it should be near what is actually right.

DNS Server:

DNS Server is one of the most important components of this project. In this project, DNS Server has 2 responsibilities. First responsibility is to provide IP Address and Port Number of the Web Server for the Web Client. Second responsibility is to deliver a Hostname(From Client), and sends a corresponding IP Address and Port Number back to the client. here I will explain each of these responsibilities in detail.

First responsibility: in order to handle this task, first we should turn the DNS Server on. Now that the DNS Server is on, it waits for any incoming request. These requests can be of two types.

First type of request is an automatic request that is initiated by the Web Server (When it turns on) in order to delivers the IP Address and Port Number of itself to the DNS Server. When this type of request is delivered by DNS server, it sends another message back to the Web Server, saying that it has received the IP Address and the Port Number of the Web Server. This way the web server will be sure that DNS server has delivered its information.

Second type of request is a request from Web Client in order to receive IP Address and Port Number of Web server, which was given to DNS server previously by the Web server itself. This way web client can connect to the web server through the DNS server.

Second responsibility: for this task, after the DNS Server and Web client Have been turned on, Web client sends a request, including a Hostname, to the DNS server, asking for an IP Address and Port Number. DNS server then looks into a file (here we used DNS_Database.jason) and search for that specific hostname. If it finds the hostname, it will return the IP Address and Port Number of that Hostname and sends it back to the Web client. Figure 1 shows a view of database file.

DNS_Database.json

```
{
    "root": {
        "domains": {
            "com": {
                "google.com": {
                    "ip": "192.168.1.100",
                    "port": "11001",
                    "subdomain": {
                        "sub1.google.com": {
                            "ip": "192.168.1.107",
                            "port": "11001"
                        },
                        "sub2.google.com": {
                            "ip": "192.168.1.105",
                            "port": "11009"
                        }
                    }
                },
                "yahoo.com": {
                    "ip": "192.168.1.200",
                    "port": "12001",
                    "subdomain": {
                        "sub1.yahoo.com": {
                            "ip": "192.168.1.210",
                            "port": "12008"
                        },
```

FIGURE 1 PART OF DNS DATABASE FILE

This file is a database and it holds IP Address and Port Number of Hostnames. I used the proposed structure of the Jason file in the project description.

As you can see there are IP and Port values for each hostname and if there is a request for that hostname, these two parameters will be returned in response. There is also a subdomain which is similar to the main domain and has a slightly

different IP and Port values. These values are also acceptable and can be returned, if the client request them. We now discuss how DNS server deliver and search for Hostname and returns the corresponding IP Address and Port Number if such Hostname exists.

Searching mechanism in DNS server for finding Hostname is just like what happens in the internet. Consider "**google.com"** as an example. DNS server will first find the latest part of the Hostname which here it is "**com**". It will search to see if there is any "**com**" domain in the database file. If it does not find such domain, it responds with "**This Hostname Is Not Valid**", otherwise it tries to find "**google.com**" domain in the "**com**" domain. Again, if it does not find such domain, it responds with "**This Hostname Is Not Valid**". But if it does find Hostname, it will send the IP Address and Port Number of "**google.com**" to the client and closes the connection.

Considerations:

In this section we will mention which parts of the mentioned notes and steps in the project description are implemented and which parts are not. Before I continue, I should mention again that everything that I will claim is based on my understanding of the project's description. it is completely possible that some of my claims be wrong due to my misunderstanding. This fact is also true in implementation of Web server and Web client.

Considered Notes: (all 5 notes are considered, notice the bold part of note 5)

1. The file above is an example of your database for the DNS server, and you should only use the structure of it.
2. DNS server runs above UDP protocol and port 53, but you should use port 5353 in this project.
3. In communication with this DNS server, you can adopt whatever format (message structure) you like.
4. Your program should be able to cope with dynamic changes of this file during runtime. So, for example, if we change one IP when the server is active, we must get the updated IP in subsequent responses.

5. Use arguments for setting your configurations instead of hard-coding them like port numbers. **(database file can be specified with arguments. This file can be run also without specified arguments due to the consideration of default setting. Here this default setting calls DNS_Database.jason to be its database automatically)**

Considered Steps: (Steps 1 through 4 are implemented)

1. Generate a JSON file as your database and put it in your DNS server directory.
2. Create a UDP socket with port 5353.
3. Listen and wait for incoming requests on this port.
4. Whenever you receive a request:
    a. Validate it if it's invalid drop it (here, you have to validate the incoming request with your desired message structure).
    b. Otherwise, look in your database for request objectives.
    c. Create a response and send it back to the client.

Web Server:

Web server is playing the role of some website in this project. Web server has one responsibility. This responsibility is to deliver a standard HTTP request from Web Client and sends back a file corresponding to that request. This standard HTTP Request consists of a GET method, a URL (which indicates address of the intended file in a file directory), version of the HTTP request and some other data which helps the Web server to understand what is the request of the Web Client exactly.

After delivering the request from the Web Client, Web Server will extract the URL part of the request and try to find the file which the URL refers to in a file directory (here it is the "**Files Directory**" folder). If this file exists, the Web server finds it and sends it back to the Web Client and if it does not, the Web server will respond with "**404 NOT FOUND**".

Web server has a reverse search mechanism compare to the DNS Server in a sense that It will start searching from beginning instead of ending. So, for example for a URL like "**google/drive/file.txt**", it will search for **google** to see if this directory exists and then **drive** and then **file.txt**. other than that, everything in searching mechanism of Web server is similar to the DNS Server searching mechanism.

Considerations:

In this section we will mention which parts of the mentioned notes and steps in the project description are implemented and which parts are not.

Considered Notes: (both notes are considered, notice note the bold part of note 2)

1. We have to be able to modify the file's directory at runtime without restarting the server. These modifications comprise:
    a. Adding new domain directory
    b. Editing existing domains name
    c. Deleting existing domain
    d. Adding new file in any domain directory
    e. Editing any file in any domain directory

f. Deleting any file in any domain directory

2. Use arguments for setting your configurations instead of hard-coding them. Like port numbers. **(Port Number and IP Address can be configured with arguments. This file can be run without arguments due to the consideration of default settings for these two parameters. This default setting sets IP Address and Port Number to be 127.0.0.1 and 8080 respectively)**

Considered Step: (steps 1 through 4 are considered)

1. Create a file directory and generate some random files, and put them in your file's directory.
2. Create a TCP socket listening on port 8080.
3. Listen and wait for incoming requests on this socket.
4. Whenever you receive a request:
    a. Validate it to be a valid HTTP request.
    b. Parse its parameters and extract request file names.
    c. If you found these files in the appropriate domain, send them back to the client.
    d. Otherwise, send 404 NOT FOUND.

Web Client:

Web client is the last part of this project and it will connect these tree components together. Web client acts like a browser and just like browser, it waits for an incoming request from client to initiates some process.

Web browser in this project can accept tree kinds of request. These request's types include "exit", "dns" and "http". each one of these requests initiates a certain process and deliver a certain service to the client.

The **first** and simplest type of request is "exit" request. This request causes the web client to shut down. This mean that you can no longer send request to the web client. in order to use this request, you should only run Web_Client (it's a python file) and then enter **exit** as request.

The **second** type of request is dns request which can return an IP address and Port Number to the client. in order to use this request, you should turn on DNS_Server (it's a python file), then you should enter your request as **dns Hostname** where client want IP Address and Port Number of the Hostname. So, for example if we want IP Address and Port Number of **google.com**, we should enter **dns google.com** (after running Web_Client file) for the DNS server to deliver this hostname and return the IP Address and Port Number, corresponding to it.

The **third** type of request is http request. This request can return a file from a file directory. In order to do so, we should first run DNS_Server file, Next We should run Web_Server file and then Web_Client file. when we run the Web_Server file, it will send a message to the DNS Server that include the IP Address and Port Number of the Web Server itself. This information would be saved in the DNS server and whenever the client wants to contact the web server, it first contacts the DNS Server for this information. After receiving this information from DNS Server, it will try to connect the web server and receive the file from file directory.

http request's format is exactly like dns request's format. So, for example if you want to receive a file from web server in directory say **google/drive/file.txt**, you should enter this request as **http google/drive/file.txt**. this request will return file.txt to you and you can do whatever you wanted to do with it.

Considerations:

In this section we will mention which parts of the mentioned notes and steps in the project description are implemented and which parts are not.

Considered Notes: (only note 2 is considered)

1. Web client must send the standard HTTP requests to the web server.

Considered steps: (steps 1 through 4 is considered)

1. Wait for the user to enter a command.
2. Upon receiving a command from the user, do as explained.
3. Print received result
4. Repeat steps 1 to 3 until the user enters the exit command.

Overall, all the necessary parts are completely implemented and all optional parts (the ones with ***this format*** in the project's description) are completely ignored.