COMP6461 – Fall 2022
Data Communications & Computer Networks
Lab Assignment # 2
Due Date: Friday, November 4, 2022 by 11:59PM

Introduction

In this assignment, you will implement a simple HTTP server application and use it with off-the-shelf HTTP clients (including **httpc** client, the result of Assignment #1). Precisely, we aim to build a simple remote file manager on top of the HTTP protocol in the server side. Before starting on this Lab, we encourage you to review the programming samples (provided with Assignment # 1) and the associated course materials.

Outline

The following is a summary of the main tasks of the Assignment:

1. Study-Review HTTP network protocol specifications (Server Side).
2. Build your HTTP server library that implements the basic specifications.
3. Develop a minimal file server on top of the HTTP server library.
4. (optional) Enhance the file server application to support simultaneous multi-requests.
5. (optional) Implement the support for **Content-Type** and **Content-Disposition headers**.

Objective

In the previous assignment, we focused on the client side of the HTTP protocol. We built a simple HTTP client command line and tested it on a real HTTP Servers (Web Servers). In this Lab, we concentrate on the server side of the HTTP protocol. Similarly to Assignment #1, the goal of the Lab is to develop your programming library that implements the basic functionalities of the HTTP server as will be described in the following sections.

The HTTP is a general-purpose protocol. However, it is mostly used in **web servers** to provide access to the **web content**. Apache HTTP Server is the most used implementation of HTTP protocol server side.

A web server is a computer system that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. The term can refer to the entire system, or specifically to the software that accepts and supervises the HTTP requests [1].

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure, efficient and

extensible server that provides HTTP services in sync with the current HTTP standards. The Apache HTTP Server ("HTTPd") was launched in 1995, and it has been the most popular web server on the Internet since April 1996. It has celebrated its 20th birthday as a project in February 2015 [2].

In our context, we use HTTP protocol to manage a remote file system through HTTP request and responses. Therefore, our goal is to build a file server application on top the developed HTTP server library.

Important Note:

You must use only the bare-minimum socket APIs provided by the chosen programming language. Therefore, you must not leverage any library that could abstract the socket programming.

Study and review HTTP Protocol

In this step, you are asked to review HTTP protocol. In this context, you should focus on the server side features, since you are requested to implement some of them. We urge you to consider HTTP version 1.0 due to its simplicity and easy to implement. You can find the complete specifications of HTTP protocol version 1.0 in this web link HTTP[4]. Furthermore, you can make tests by sending requests and receiving HTTP server responses using the **Telnet [3]** command line. The pervious tests could show the typical HTTP responses for your testing requests.

Develop HTTP Server Library

In this part, you are requested to develop your HTTP server library separately from application, to decouple the HTTP protocol specification from the intended end-application. Your HTTP library should be self-contained with a minimum dependency such as Socket library.

You are required to implement only a subset of the HTTP specifications. In essence, the library should include the features that can handle the requests from the **httpc** app of Assignment #1. To this end, you can test your library by implementing testing examples to check if it is working properly with client applications.

When you finish the testing examples you should check with the Lab Instructor.

Build a File Server Application Using Your HTTP Library

In this task, you are required to make an end-application to the previous library functionalities. In other words, you should build a remote file server manager on top the library according to the following requirements:

1- `GET /` returns a list of the current files in the data directory. You can return different type format such as JSON, XML, plain text, HTML according to the `Accept` key of the header of the request. However, this is not mandatory; you can simply ignore the header value and make your server always returns the same output.

2- `GET /foo` returns the content of the file named `foo` in the data directory. If the content does not exist, you should return an appropriate status code (e.g. HTTP ERROR 404).

3- `POST /bar` should create or overwrite the file named `bar` in the data directory with the content of the body of the request. You can implement more options for the POST such as `overwrite=true|false`, but again this is optional.

Secure Access

Your implementation may have a severe access vulnerability. The end-user could access not only the file of the default working directory of the server application but he/she could access most server files (read-write or read-only). To solve the previous problem, you should build a mechanism to prevent the clients to read/write any file outside the file server working directory.

Error Handling

In this task, you need to enhance the file manager with the appropriate error handlers. In this context, each exception on the server side should be translated to an appropriate status code and human readable messages. For example, if the requested file does not exist, the file server should send a message with this information. Similarly, if the server is unable to process the request for security reasons (the requested file is located outside the working directory), an appropriate handling must be performed.

Usage

```
httpfs is a simple file server.

usage: httpfs [-v] [-p PORT] [-d PATH-TO-DIR]
```

```
   -v   Prints debugging messages.

   -p   Specifies the port number that the server will listen and serve at.
        Default is 8080.

   -d   Specifies the directory that the server will use to read/write
requested files. Default is the current directory when launching the
application.
```

When you finish your file server application, you need to check with the Lab Instructor.

Optional Tasks (Bonus Marks)

If you have successfully completed the material above, congratulations; as you now understand the implementation of an HTTP server and network protocols in general. For the rest of this lab exercise, we have included the following optional tasks. These optional tasks will help you gain a deeper understanding of the material, and if you can do so, we encourage you to complete them as well. Bonus marks will be given for that.

Multi-Requests Support

Your implementation of the file server may support only one client at given moments. Therefore, if you run multiple clients simultaneously, the server can answer only one request. To solve this problem, you are invited to develop a concurrent implementation of the file server, where the server can handle multiple requests simultaneously. The later means that you should have a dynamic data structure to scale according to the server machine capacity.

To test your concurrent version, you can write a simple script to run multiple client instances (instances of **httpc** or **curl**). The script should take the number of client instances as a parameter. If you support concurrent requests, make sure the following scenarios work correctly:

- Two clients are writing to the same file.
- One client is reading, while another is writing to the same file.
- Two clients are reading the same file.

Content-Type & Content-Disposition Support

Set appropriate values to the headers **Content-Type** and **Content-Disposition** headers for 'GET /file' requests. For more details, the you could consult [5][6].

Testing, Submission and Grading

Important Note: You can this assignment individually or in a group of **at most 2 members** (i.e. you and another student). No extra marks or any special considerations will be given for working individually.

Testing

In order to test the developed file server, you should leverage your HTTP client (httpc) of Assignment #1 for this purpose. If you did not finish your httpc application, you could use cURL as a replacement for this assignment. However, we urge to complete your Assignment #1 HTTP client (httpc) because you will re-use it for the following assignment.

Deliverable

1) Create one zip file, containing the necessary source-code files (.java, .c, etc.)

You must name your file using the following convention:

If the work is done by 1 student: Your file should be called A#_studentID, where # is the number of the assignment. studentID is your student ID number.

If the work is done by 2 students: The zip file should be called A#_studentID1_studentID2, where # is the number of the assignment. studentID1 and studentID2 are the student ID numbers of each of the group members.

2) Assignments must be submitted in the right folder/dropbox on the course Moodle page. Assignments uploaded to an incorrect folder will not be marked and result in a **zero mark**. **No resubmissions will be allowed.**

Demo

A demo is needed for this assignment and your lab instructors will communicate the available demo times to you, where you **must** register a time-slot for the demo, and you must prepare your assignment and be ready to demo at the start of your time-slot. If the assignment is done by 2 members, then **both members must** be present for the demo. During your presentation, you are expected to demo the functionality of the application, explain some parts of your implementation, and answer any questions that the lab instructor may ask in relation to the assignment and your work. Different marks may be assigned to the two members of the team if needed. **Demos are mandatory. Failure to demo your assignment will entail a mark of zero for the assignment regardless of your submission. If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**

Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.

Grading Policy (10 Marks)

1. Implement HTTP server library: 3 Marks
2. Implement **GET /**: 2 Marks
3. Implement **GET /filename**: 2 Marks
4. Implement **POST /filename**: 2 Marks
5. Security Hardening: 0.5 Marks
6. Error handling: 0.5 Marks

Optional Tasks (2 Marks)

1. Support **Concurrent Requests**: 1 Marks
2. Support **Content-Type** and **Content-Disposition**: 1 Marks

---

References

[1] Web server. https://en.wikipedia.org/wiki/Web_server.

[2] Apache. https://httpd.apache.org/

[3] Telnet: https://en.wikipedia.org/wiki/Telnet.

[4] HTTP 1.0: https://www.w3.org/Protocols/HTTP/1.0/spec.html

[5] Content Type: https://www.w3.org/Protocols/rfc1341/4_Content-Type.html

[6] Content-Disposition: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Disposition

---