CITRINI Mathias

# Project 1: Web server

## Architecture

Une image contenant texte, écran, noir, capture d’écran

Description générée automatiquement

### /

Root

#### .gitignore

File to ignore certain type of files on GitHub

#### makefile

The MakeFile

#### report.pdf / report.docx

Report file

#### README

File readme on github

#### Run.sh

Script shell which can make and execute the server (use: ./run.sh <PORT>)

### /include

Contains header.

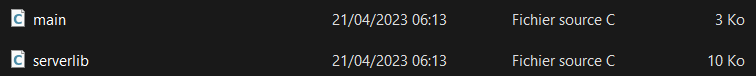


#### serverlib.h

File where all the useful functions and structures are declared (header)

### /src

Contains all the .c files.



#### main.c

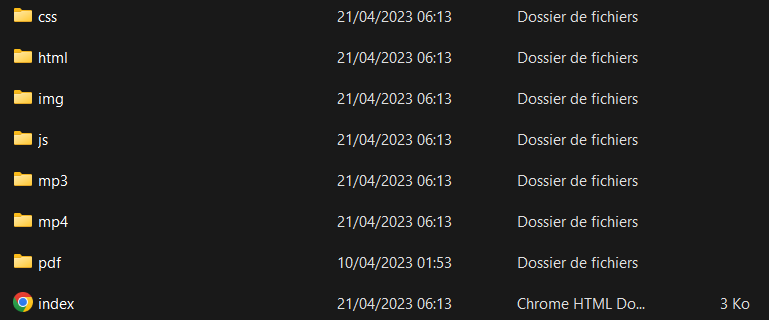
Main file of the project

#### serverlib.c

File where all the functions are implemented

### /www

Contains all the website content



#### css & html

Contains respectively the different style sheet file and html file.

#### img

Contains 3 WEBP files, 1 png file, 1 jpg file and 1 gif file.

#### js

Contains a simple js script, who print the actual data/time in index.html.

#### mp3 & mp4

Contains respectively “audio.mp3” and “video.mp4”.

#### pdf

Contains a pdf file of the French calendar.

#### index.html

Homepage of the website

## Commands

### To compile the project, you just have us the makefile.

Command: make

### To clean the generated files, use :

Command: make clean

### To start the server, use :

Command: ./server <port>

Replace <port> with your own port (ex: 8077)

## Description of the server

* PartA:

Line chart

Description automatically generated with low confidence

. socketGenerator() :

Create a socket with the port wanted, bind() and listen().

Return the socket number.

. responseCreator() :

Create a response by concatenating the header of an HTTP response with the browser request and parts of HTML file.

There is an infinite loop to be able to answer to several browser requests.

* PartB:

Timeline

Description automatically generated

. socketGenerator() :

Create a socket with the port wanted, bind() and listen().

Return the socket number.

. sendText() :

Send data in plain text (for HTML pages)

Return 0 if it’s OK and 1 if there is an error.

. sendBinary() :

Send data in binary text (mp3, jpeg, png, pdf, …)

Return 0 if it’s OK and 1 if there is an error.

. sendBrowserRequest() :

Send an HTML page with the browser request.

Return 0 if it’s OK and 1 if there is an error.

* Other functions:

. whatExt() : return the extension of the file requested

. sendNotFound() : when a file doesn’t exist or the program can’t open it, a response with the code 404 is sent

. replaceWord() : replace a string in a string

## Webserver hub

I created a dummy website to test every requirement easily. This website is a basic one and use only one js script and one css file. It can display images, load video etc…

To access this website, please go to: <ip address>:<port>/index.html

[Add pictures]

## Request Analysis

Graphical user interface, text, application

Description automatically generated

* Line

#1: There is the method used (“GET”), path of the file wanted (here “/index.html”) and the name of protocol (“HTTP”) with its version (“1.1”)

#2: There is the host (web server) address and its port (“127.0.0.1:8077”)

#3: Indicates that the client would like to keep the connection open

#4: The server can now redirect to a secure version of the site

#5: Some information about the browser who request and the OS of the computer

#6: Used to inform the server by the client that which content type is understandable by the client expressed as MIME-types

#7: GPC is the latest attempt at allowing customers to specify how their browsing data is to be shared online

#8: specifies the relationship between the origin of the request initiator and the requested resource’s origin

#9: specifies the request mode

#10: only sent when a user is activated

#11: specifies the destination of the request

#12: indicates the content encoding

#13: indicates the natural language and local that the client prefers

## Difficulties

The first difficulty I was when I had to parse the HTTP request. Sometimes, I had core dumped so I had to choose the best size for each field and optimize my “unparseHTTPRequest” function.

The second difficulty was to understand how to send binary files (Images, Audio …). In fact, I took some time to understand how binary files work and how to read it. I learned the best way was to use void\* instead of char\*, and so, memcpy instead of strcat. After long hours of pain, I finally managed to get the proper size of the file, read it with fread and copy the binary data in a string to send it to the client.

To summarize, memory problem and binary data were my biggest difficulties in this project. Every other things were easy to do and didn’t take me too much time.

## For the future

The server is 100% working but an interesting feature to add could be being able to handle different connections (if many clients are connected at the same time).

Maybe we can also optimize the memory, with some free() functions, and allocate more dynamically the memory (ex: deleting all char[x] and replace them with char\*).

## Useful links

Apart from Stack Overflow and the course, I didn’t used any website to make this project.

<https://stackoverflow.com/>