Project1: Socket Programming & P2P network

M2608.001200 Introduction to Data Communication Networks (2022 Fall)

Instructors: Prof. Kyunghan Lee
TAs: Gibum Park, Jongseok Park, Sanghyeon Han
Due date: Nov. 7, 2022, 11:59 pm.

Notes:

- Be aware of plagiarism. You are allowed to use the eTL for QnA, but do **NOT** discuss it with your classmates **directly**.
- **Grading**: 200 pts total (40% Web server, 60% torrent)
- If you find bugs, typos, or issues in the given code, please report them to the TAs by posting on the ETL. We will look into them and fix them as soon as possible. Once fixed, we will upload the new version on the ETL by naming the proper version number for the code and making an announcement on the ETL. Please pay attention to it and make sure you are using the latest version before submission.
- If you do not want other students to see your questions, you can always make them private.

Objective

In this project, we will implement simple servers that require http authentication using socket programming, and then implement torrents using p2p communication.

Program Specification

1) Web Server

The Skeleton code is provided in the assignment tab of *ETL*. Copy the code to your Linux System (WSL, MacOS, Ubuntu, etc.), and modify it to implement your own web server. Your web server should send requested files to the client through HyperText Transfer Protocol (HTTP).

Check the message format of HTTP (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol). Basically, you should fill up empty parts in the skeleton code to run this code successfully. This code consists of two functions: main() and respond()

main() function lets a socket bind to address and port (using bind() function) and listen on the socket (using listen() function). Then, it accepts a connection from a client (using accept() function) and sends the requested file to the client through HTTP message format. The server should continuously accept other clients after handling a request. Furthermore, the http 401 authentication process has been added to this project.

respond() function reads a request which contains a file path. In the HTTP header, you can find the requested file name. Then, you can take the file path (e.g, index.html, public/image1.jpg, bootstrap.js), read the files, attach the HTTP header to make a response message, and send the response message back to the client. You must explicitly implement

the exception-handling part in **respond()** for the requests of resources or files that the server cannot handle. If you do not implement the exception-handling, you cannot get a full score. Refer https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

2) Torrent-esque file sharing system

In this assignment, you will implement a torrent-esque P2P file sharing system. This system defines a custom network protocol for file sharing. You must implement the said protocol.

Our torrent system partitions a file into blocks and distributes them to peers. The torrents are referenced using a hash value of the file in our torrent network. Management of the torrent information and data is not the focus of this assignment. What you will be implementing is the custom network protocol of our torrent network. You will create a working torrent program that can communicate with the model implementation of our torrent network, at the end of this assignment.

Our custom network protocol is implemented as follows:

- A peer first requests torrent info from a seeder using the hash value of the torrent, using REQUEST_TORRENT command.

- The seeder responds with the torrent info, using the PUSH_TORRENT command. The requester receives the torrent info, creates a new torrent_file struct and adds it to its global torrent list. (Refer to torrent_functions.h) The seeder is also added to the peer list of the new torrent.

- The requester requests block info of the seeder using the REQUEST_BLOCK_INFO command. The seeder responds with the block info of the torrent, using the PUSH_BLOCK_INFO command.

- With the block info of the seeder, the requester can now check which block the seeder has. The requester then requests its missing blocks from the seeder using the REQUEST_BLOCK command. The seeder responds with the requested block, using the PUSH_BLOCK command.

- At random intervals, the requester can request a peer list from a random peer(seeder) using the REQUEST_PEERS command. The random peer responds with its peer list of the torrent, using the PUSH_PEERS command. This way, the requester can discover new peers of the torrent.

- The requester can also request block info of the newly discovered peer using REQUEST_BLOCK_INFO command, and request missing blocks from those new peers using REQUEST_BLOCK command.

- Of course, being a P2P system, any requester can also act as a seeder, and any seeder can also act as a requester.

The requester routines are managed in client_routines() function, while the seeder routines are managed in server_routines() function. Please refer to the comments in main.c, network_functions.h, torrent_functions.h and the code structure of the skeleton code for more details.

The Skeleton code is provided in the assignment tab of *ETL*. Copy the code to your system (WSL, MacOS, Ubuntu, etc.), and modify main.c to complete your assignment.

Note: A more detailed explanation of the assignment can be found in the main.c file.
Please read the explanation in main.c file CAREFULLY before starting your assignment!

Implementation Guides

For both problems 1 and 2 please read the comments carefully.
Both problems 1 and 2 are tested on WSL, Ubuntu, M1 Mac environments.

1. Simple Web server

The server must respond to the client's first request with a 401 HTTP error message and request an ID and password. If you send proper message login and password field will pop up in browser

If you enter your ID and password in the browser window, you can compare the base64 encryption values, and if they are the same, authentication passes and exits the authentication loop.

After that it will wait for next request and process the request

2. Torrent-esque file sharing system

For this assignment, you must implement the following functions.

- request_peers_from_peer (Points: 5)
- push_peers_to_peer (Points: 5)
- request_block_info_from_peer (Points: 5)
- push_block_info_to_peer (Points: 5)
- request_block_from_peer (Points: 5)
- push_block_to_peer (Points: 5)
- server_routine (Points: 40)
- client_routine (Points: 30)

Total of 100 points.

The model implementation of the functions you have to implement are provided with a *_ans suffix.
You must achieve the same functionalities as the model implementation.
Points will be deducted if the functions are not implemented correctly.
(Crash/freezing, wrong behavior, etc.)

Please refer to the “Program Details” section in main.c for more details.

Grading

The grading policy is as follows:

- Simple Web server : 40%
- Torrent-esque file sharing system : 60%

Total score (200) = 0.4 * Web server score + 0.6 * Torrent score

You have to write comments in your codes for logically explaining what you did.

Please include short report that explains your output and codes

(If you don't write comments or report , you will get **-10% reduction each**)

Late submission : 10% deduction per 12 hours

Compile and Usage

1) Web server

File name:

server.c , base64.c

Compile:

make (Use brew to install gcc and make on MAC)

Usage:

1) Check the ip address of your VM (type in “ifconfig” in terminal).

2) Compile your code (make).

3) Run server (./server).

4) Open a web browser on your desktop or smartphone.

5) Open link “http://IP:25000”. (e.g., http://192.168.0.1:25000)

2) Torrent-esque file sharing system

File name:

main.c , torrent_function.h , network_function.h
(please read comments in the three files carefully)

Compile:

(LINUX) “gcc -o main main.c torrent_functions.o network_functions.o”
(MAC) “ gcc -o main main.c torrent_functions_MAC.o network_functions_MAC.o”
(Use brew to install gcc on MAC)

Usage:

1) Compile your code

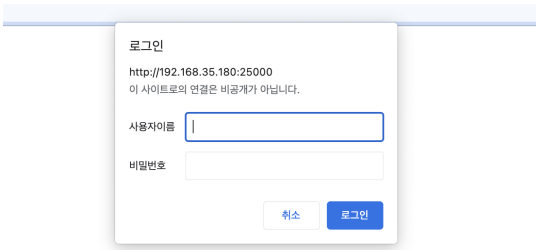
2) Open two terminals

3) Run “./main 127.0.0.1 1” on one terminal and “./main 127.0.0.1 2” on the other.

4) The downloaded files will be saved under the “Downloaded” directory.

Output Specification

1) Web server



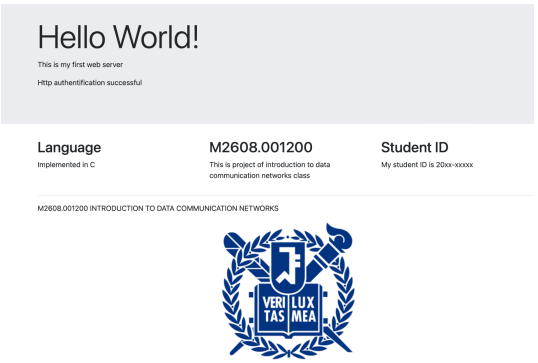


Figure1: Authentication phase **Figure2: index.html**

Before submitting your code, change the “index.html” file to include your student ID.

2) Torrent-esque file sharing system

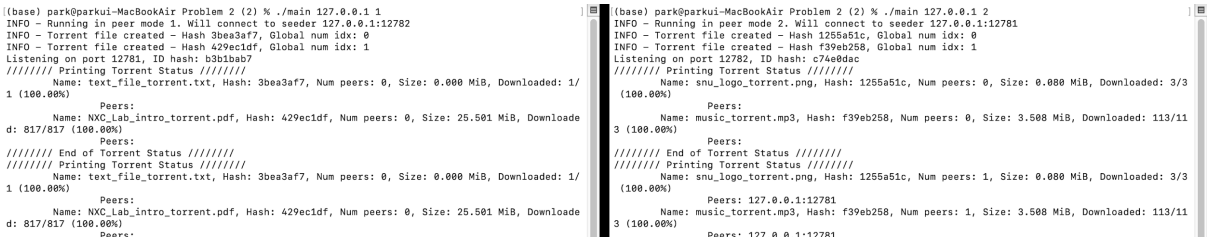


Figure1:Running two terminals

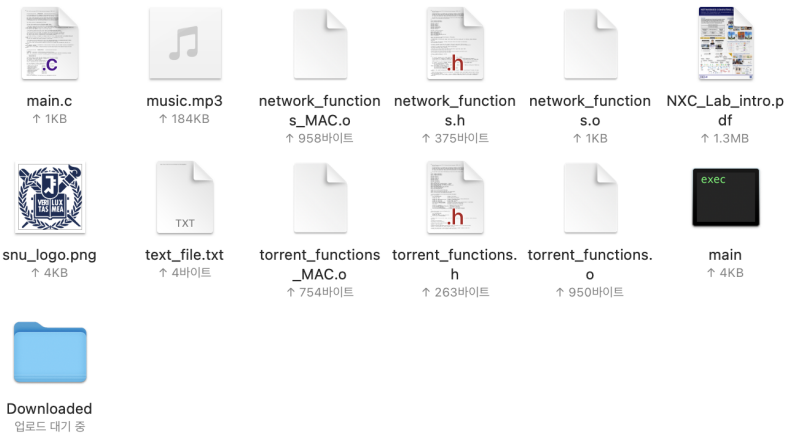


Figure2:”Downloaded” directory created

“Downloaded” directory will contain a copy of “text_file_torrent.txt”, “snu_logo_torrent.png”, “music_torrent.mp3”, and “NXC_Lab_intro_torrent.pdf”, if all functions work properly.
The file can also be broken, if not implemented properly or if the file download was halted preemptively. Always open the downloaded file to check if the torrent works as intended.

Submission

- Submit your codes and report explaining your output and codes
- Include your report and code to a folder “project1_<student id>” and compress the folder.
- Make the compressed zip file name as “project1_<student id>”.
- You should submit only one compressed zip file to ETL.