EE 4020: Introduction to Computer Networks Spring 2025

Programming Assignment

Due date: 2025/04/04 21:00

1 Introduction

The main goal of this assignment is to get hands-on experience with socket programming and how the server interacts with clients. In addition, this programming assignment consists of three parts:

- TCP Socket Programming: Implement a TCP server and client system that can perform basic arithmetic operations.
- Web Server: Implement a web server that can process basic HTTP requests.
- Proxy Server: Implement a proxy server with caching functionality.

Additionally, you are required to submit a report (in a PDF file) that includes:

- Explanations of the code for each problem.
- Screenshots or results demonstrating the successful execution of each problem.

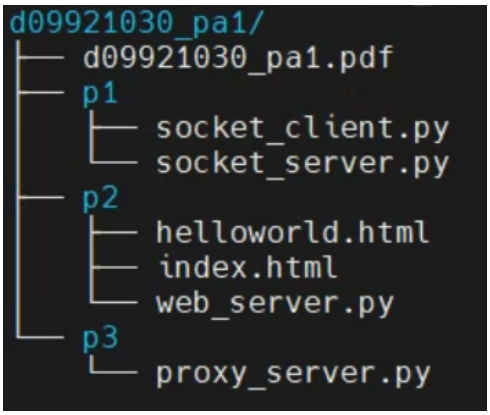


Figure 1: Folder structure

Submission requirements. This assignment is due on the date listed above. Please use (last four digits of your student ID + 1023) % 65535 as the port number. For example, if your student ID is d09921030, the port number would be (1030 + 1023) % 65535 = 2053. If the calculated port number is a bad port, please indicate it in the program and use 8888 as the port for submission.

- Bad ports list: `https://fetch.spec.whatwg.org/#port-blocking`

Save all the files as a zip file without any specific naming requirements and upload to NTU Cool. After unzipping, make sure the folder structure and names exactly mirror the figures below, replacing the placeholder with your student ID.

General grading policies. Please refrain from plagiarizing in your assignments. In the event of suspicion, the TA may request clarification. Any confirmed cases of plagiarism will result in a zero score for the assignment. Late submissions for one day result in a $\frac{1}{3}$ score deduction. Those who exceed one day but less than two days late incur a $\frac{2}{3}$ score deduction. For more than two days, no credit is given. Programs that cannot be compiled by TA will be scored zero. Incorrect file formats or folder structures will result in a 20-point deduction.

Collaboration policy. Collaboration and discussion are encouraged. However, your solution and submission must be coded and written yourself. If you have discussed the assignment with classmates, please note their names and student IDs at the beginning of your code.

Contact. If you have any questions, feel free to contact the TA by email. The email tile should be [2025ICN] PA.
TA's Email: d09921030@ntu.edu.tw

2 Socket Programming - TCP – 20%

In this assignment, you will implement a TCP socket programming application with a client and a server via Python. Fig. 2 illustrates the communication flow between the client and the server.

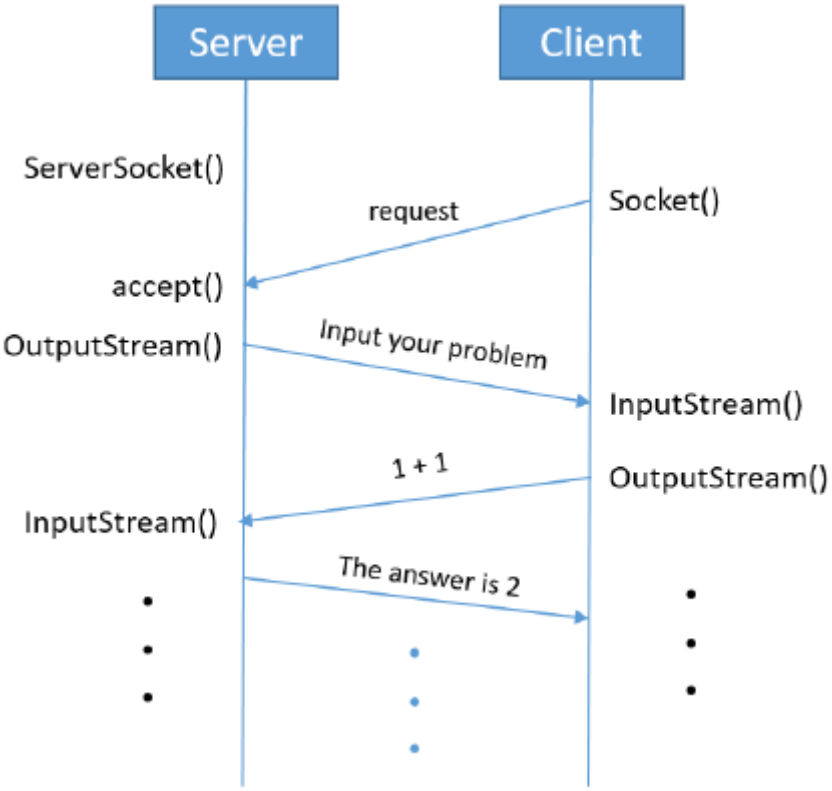


Figure 2: Message flow between the client and server.

In brief, the client sends arithmetic expressions to the server. The server evaluates the received expressions and sends the results back to the client. The client and server interact in a loop, where the server prompts the client for expressions, and the client responds with the expression. The client also responds a decision to continue or terminate the loop.

2.1 Programming Part

(a) Complete the **TODO** part in `socket_server.py` . The server should work as a calculator that supports basic operations such as addition (+), subtraction (-), multiplication (×), and division (÷).

(b) Complete the **TODO** part in `socket_client.py` so that it can:

- Read the messages sent from the server.

- Send the messages (from `p1_testcase_basic`) to the server and receives the answer again from the server.

(c) Compile and run `socket_server.py` first, and then execute your `socket_client.py`.

(d) Test it on your local machine (127.0.0.1) first. If you have done it correctly, the execution and output of `socket_client.py` should resemble Fig. 3.

```
The Client is running..
Received the message from server:
Please input a question for calculation
Question: 1+1
Get the answer from server:
2.0
Do you wish to continue? (Y/N)
Response to server prompt: Y
Received the message from server:
Please input a question for calculation
Question: 2-4
Get the answer from server:
-2.0
Do you wish to continue? (Y/N)
Response to server prompt: Y
Received the message from server:
Please input a question for calculation
Question: 3*5
Get the answer from server:
15.0
Do you wish to continue? (Y/N)
Response to server prompt: Y
Received the message from server:
Please input a question for calculation
Question: 4/2
Get the answer from server:
2.0
Do you wish to continue? (Y/N)
Response to server prompt: N
```

Figure 3: The result of connecting to the server on the local machine (127.0.0.1).

(e) When you finish (d), try to run `socket_client.py` and connect to TA's computer with TA's IP address 140.112.42.104 and port 7777.

2.2 Submission and Grading Policy

1. Please put your source code (including `socket_client.py` and `socket_server.py`) under `[student_id]_pa1/p1`.
2. We provide `p1_testcase_basic` for you to test your program. The result should be exact the same as `p1_testcase_golden`. Please make sure your program is available to save the result to a file `[student_id]_p1_client_result_basic.txt`. TA will use this file to grade your program.
3. Your program should be able to adapt different input parameters, such as the server name, the input testcase, and the student ID. TA will utilize them for grading in the execution of `socket_client.py`. The example for executing `socket_client.py` is shown in Fig. 4

```
python3 socket_client.py --server local --testcase basic --studentID r12942144
```

Figure 4: Executing `socket_client.py` with input parameters.

4. You will earn 15% for correct output. If your program can successfully connect to the TA's computer, you will earn 5%.
5. In `[student_id]_pa1.pdf`, you have to
   - Briefly explain how `socket_server.py` and `socket_client.py` work.
   - Show the result of 2. in **2.2 Submission and Grading Policy** section.

3 Web Server – 35%

In this assignment, you will develop a simple web server that is capable of processing basic requests. Different from the above problem. the client is now your browser. The client sends the HTTP request to the web server, and the web server should be able to handle the HTTP request and send the HTTP response back to the client.

3.1 Programming Part

(a) Create two HTML files. One named `index.html` and the other `helloworld.html`. The browser should be able to access both HTML files through the web server, and access helloworld.html through index.html.

- Show "Hello World!" in `helloworld.html`. Show your name and student ID in `index.html`.

(b) Complete the **TODO** part for web server functions.

- Set up the web server socket to listen for incoming connections from the client.
- When a client browser requests a file, parse the HTTP request to identify the requested file.
- Read the requested HTML file from the server's file system.
- Construct an HTTP response with the file content and send it back to the client.
- Handle 404 Not Found errors if the requested file does not exist. For example, if the client requests a file that is not on your server, your web server should return a 404 Not Found message to the client.

(c) The message flow between the client and the web server is shown in Fig. 5.

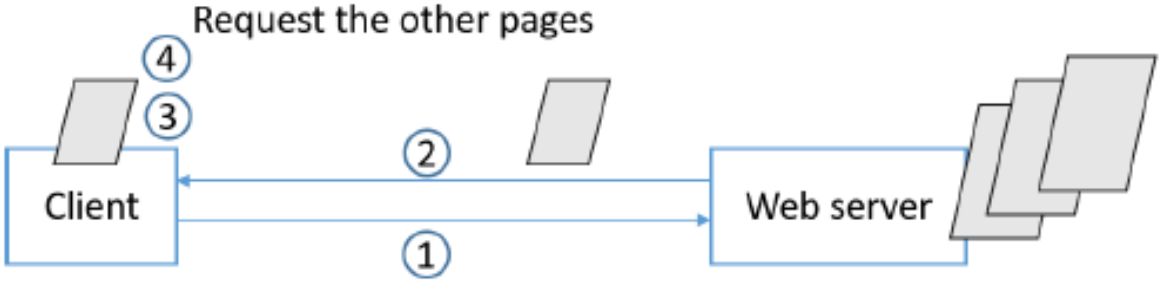


Figure 5: Message flow between the client and the web server.

3.2 Hints

1. Ensure that HTML files are placed in the directory where the web server is running, so that you can access HTML files through the browser.
2. To test whether your program can access the HTML files, please use the URL for the browser: http://HOST:PORT/index.html
3. To test the 404 NOT FOUND error, simply use the URL: http://HOST:PORT/testnotfound.html
4. Firefox is recommended since it can set the proxy and port for itself independently. However, different types of browser are irrelevant to this assignment.
5. You can ignore the error message from "FAVICON.ICO".
6. The example results are shown in Fig. 6

3.3 Submission and Grading Policy

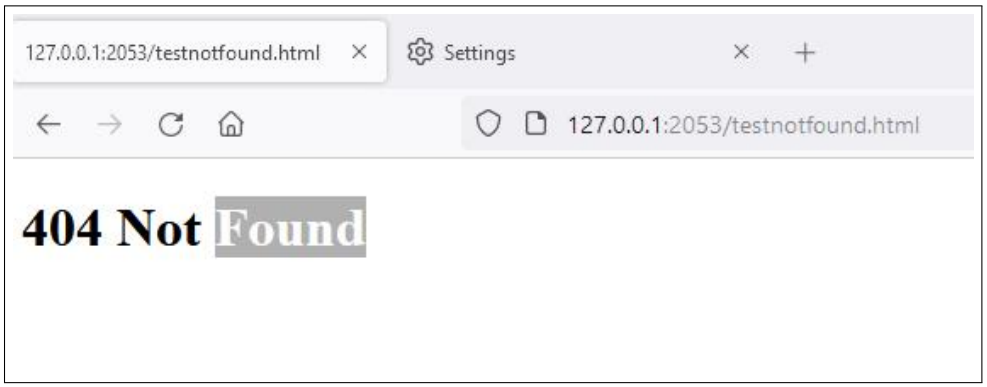
1. Please put your source code (`web_server.py`) and HTML files under `[student_id]_pa1/p2`. Also, please screenshot your output results and put them into `[student_id]_pa1.pdf`.
2. [20%] Your program should be able to access HTML files through a local machine.
3. [10%] Your program should be able to show 404 NOT FOUND in the browser when the requested file is not on the web server.
4. [5%] Your program should be able to access `helloworld.html` through `index.html`.
5. In `[student_id]_pa1.pdf`, you have to
   - Briefly explain how `web_server.py` works.
   - Show the results of 2., 3, and 4 in **3.3 Submission and Grading Policy** section.

```
Ready to serve...
('127.0.0.1', 14182) connected
client's request message:
 GET /INDEX.HTML HTTP/1.1
HOST: 127.0.0.1:2053
USER-AGENT: MOZILLA/5.0 (WINDOWS NT 10.0; WIN64; X64; RV:109.0) GECKO/20100101 FIREFOX/117.0
ACCEPT: TEXT/HTML,APPLICATION/XHTML+XML,APPLICATION/XML;Q=0.9,IMAGE/AVIF,IMAGE/WEBP,*/*;Q=0.8
ACCEPT-LANGUAGE: EN,ZH-TW;Q=0.5
ACCEPT-ENCODING: GZIP, DEFLATE, BR
CONNECTION: KEEP-ALIVE
UPGRADE-INSECURE-REQUESTS: 1
SEC-FETCH-DEST: DOCUMENT
SEC-FETCH-MODE: NAVIGATE
SEC-FETCH-SITE: NONE
SEC-FETCH-USER: ?1


Extract the filename: INDEX.HTML
Ready to serve...
```

(a) Access `index.html`

```
Ready to serve...
('127.0.0.1', 14728) connected
client's request message:
 GET /HELLOWORLD.HTML HTTP/1.1
HOST: 127.0.0.1:2053
USER-AGENT: MOZILLA/5.0 (WINDOWS NT 10.0; WIN64; X64; RV:109.0) GECKO/20100101 FIREFOX/117.0
ACCEPT: TEXT/HTML,APPLICATION/XHTML+XML,APPLICATION/XML;Q=0.9,IMAGE/AVIF,IMAGE/WEBP,*/*;Q=0.8
ACCEPT-LANGUAGE: EN,ZH-TW;Q=0.5
ACCEPT-ENCODING: GZIP, DEFLATE, BR
CONNECTION: KEEP-ALIVE
UPGRADE-INSECURE-REQUESTS: 1
SEC-FETCH-DEST: DOCUMENT
SEC-FETCH-MODE: NAVIGATE
SEC-FETCH-SITE: NONE
SEC-FETCH-USER: ?1


Extract the filename: HELLOWORLD.HTML
Ready to serve...
```

(b) Access `helloworld.html`



(c) Access a non-existing `testnotfound.html`

Figure 6: Access HTML files

4 Proxy Server – 35%

In this assignment, you will implement a proxy server that can receive requests from the client, fetch the requested files from a web server, cache the files locally, and serve the cached files to subsequent client requests. The proxy server should also handle scenarios where the requested file is not found and return appropriate error messages.

4.1 Programming Part

(a) Complete the **TODO** part for the proxy server functions.

- Creating a socket on the proxy server to receive requests from the client.
- Parsing the client's request to determine the requested file.
- Checking if the requested file is available on the local proxy server cache. If yes, send the file back to the client (as you did in the previous problem).
- If the file is not in the cache, create another socket to connect to the web server and request the file.
- Receive the response from the web server, send the requested file to the client, and cache the file on the local proxy server.
- If the file is not found on the web server, return a 404 NOT FOUND error message to the client. Do not cache files that are not found.

(b) The message flow is shown in Fig. 7.

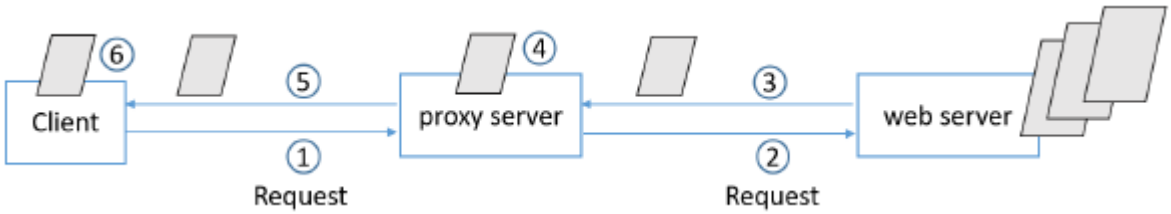


Figure 7: Message flow among Client-Proxy-Server.

4.2 Hints

1. Please set the port of the proxy server to **9999**.
2. Make sure to configure your web browser to use the proxy server. Do not connect directly to the web server.
3. To test whether your program can access the HTML files, please use the URL for the browser: http://HOST:9999/index.html. Please make sure that there are no HTML files in the p3 directory initially.
4. To test the 404 NOT FOUND error, simply use the URL: http://HOST:9999/testnotfound.html
5. You can ignore the error message from "FAVICON.ICO".
6. The example results are shown in Fig. 8.

4.3 Submission and Grading Policy

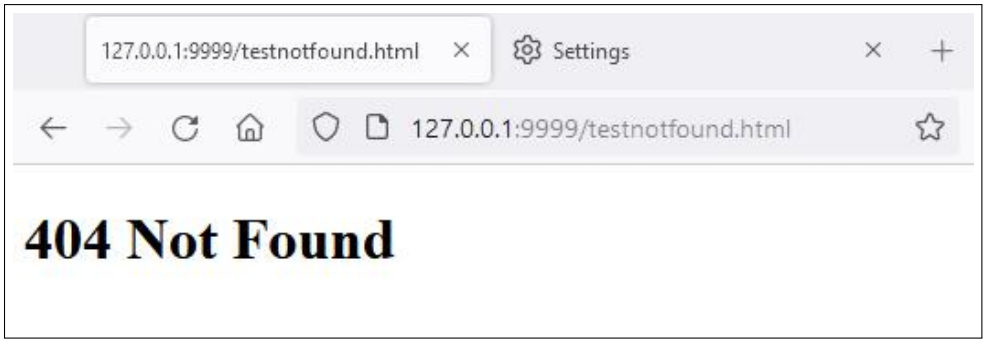
1. Please put your source code (`proxy_server.py`) under `[student_id]_pa1/p3`. Also, please screenshot your output results and put them into `[student_id]_pa1.pdf`.
2. [25%] Your program should be able to access HTML files (which is located at p2 directory) through the proxy server.
3. [10%] Your program should be able to show 404 NOT FOUND on the browser when the requested file is not in neither the proxy server or the web server.
4. In `[student_id]_pa1.pdf`, you have to
   - Briefly explain how `proxy_server.py` works.
   - Show the results of 2 and 3 in **4.3 Submission and Grading Policy** section.

```
index.html
/index.html
Read from cache
Ready to serve...
Received a connection from: ('127.0.0.1', 4507)
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:9999
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/117.0
Accept: image/avif,image/webp,*/*
Accept-Language: en,zh-TW;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1:9999/index.html
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

(a) Access index.html

```
HELLOWORLD.html
/HELLOWORLD.html
Host name is HELLOWORLD.html
Trying to connect to the web server
Connected successfully
GET /HELLOWORLD.html HTTP/1.0

Sent the request to the web server successfully
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html lang = "en">
<head>
        <meta charset ="utf-8">
        <title> Hello World! </title>
</head>
<body>
    Hello World
</body>
</html>
```

(b) Access helloworld.html



(c) Access a non-existing testnotfound.html

Figure 8: Access HTML files

5 Report (`[student_id]_pa1.pdf`) – 10%

- For each problem
  - Show your results.
  - Briefly describe how each program works.
- Grading
  - 3% for **2 Socket Programming - TCP**
  - 3% for **3 Web Server**
  - 4% for **4 Proxy Server**