Network Programming Project 3 (Part 1)
HTTP Server and CGI Programs

NP TA

Deadline: Wednesday, 2024/5/15 23:55

1 Introduction

The project is divided into two parts. This is the first part of the project.
Here, you are asked to write a simple HTTP server called **http_server** and a CGI program **console.cgi**. We will use **Boost.Asio** library to accomplish this project.

2 Specification

2.1 http_server

1. In this project, the URI of HTTP requests will always be in the form of /${cgi_name}.cgi (e.g., /panel.cgi, /console.cgi, /printenv.cgi), and we will only test for the HTTP GET method.

2. Your **http_server** should parse the HTTP headers and **follow the CGI procedure** (fork, set environment variables, dup, exec) to execute the specified CGI program.

3. The following environment variables are required to set:

   (a) REQUEST_METHOD: The method with which the HTTP request was made. e.g. GET.

   (b) REQUEST_URI: Extra path information, which is decoded by server and pass to CGI script.

   (c) QUERY_STRING: The information which follows the "?".

   (d) SERVER_PROTOCOL: The name and revision of the information protcol.

   (e) HTTP_HOST: Specifies the Internet host and port number of the resource being requested.

   (f) SERVER_ADDR: The ip address to which the request was sent.

   (g) SERVER_PORT: The port number to which the request was sent.

   (h) REMOTE_ADDR: The ip address of the remote host making the request.

   (i) REMOTE_PORT: The port number of the remote host making the request.

   For instance, if a client from 140.113.167.57:63633 sends a http request to http server nplinux12.cs.nycu.edu.tw:12345, and the first two lines of HTTP request look like:

```
GET /console.cgi?h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&
    h1=nplinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&h2=&p2=&f2=&
    h3=&p3=&f3=&h4=&p4=&f4= HTTP/1.1
Host: nplinux12.cs.nycu.edu.tw:12345
```

Then before executing console.cgi, you need to set the corresponding environment variables. In this case:

(a) REQUEST_METHOD=GET

(b) REQUEST_URI=/console.cgi?h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&
h1=nplinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&
h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=

(c) QUERY_STRING=h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&
h1=nplinux12.cs.nycu.edu.tw&p1=12343&f1=t2.txt&
h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=

(d) SERVER_PROTOCOL=HTTP/1.1

(e) HTTP_HOST=nplinux12.cs.nycu.edu.tw:12345

(f) SERVER_ADDR=140.113.235.235 (ip address of nplinux12.cs.nycu.edu.tw)

(g) SERVER_PORT=12345

(h) REMOTE_ADDR=140.113.167.57

(i) REMOTE_PORT=63633

2.2 console.cgi

1. You are highly recommended to inspect and run the CGI samples before you start this section. For details about CGI (**C**ommon **G**ateway **I**nterface), please refer to the course slides as well as the given CGI examples.

2. The **console.cgi** should parse the connection information (e.g. host, port, file) from the environment variable **QUERY_STRING**, which is set by your HTTP server (see section 2.1).

   For example, if **QUERY_STRING** is:

```
h0=nplinux12.cs.nycu.edu.tw&p0=12344&f0=t1.txt&h1=nplinux12.cs.nycu.edu.tw&
p1=12343&f1=t2.txt&h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=
```

   It should be understood as:

```
h0=nplinux12.cs.nycu.edu.tw    # the hostname of the 1st server
p0=12344                       # the port of the 1st server
f0=t1.txt                     # the file to open

h1=nplinux12.cs.nycu.edu.tw    # the hostname of the 2nd server
p1=12343                       # the port of the 2nd server
f1=t2.txt                     # the file to open

h2=                           # no 3rd server, so this field is empty
p2=                           # no 3rd server, so this field is empty
f2=                           # no 3rd server, so this field is empty

h3=                           # no 4th server, so this field is empty
p3=                           # no 4th server, so this field is empty
f3=                           # no 4th server, so this field is empty
```

```
h4=                             # no 5th server, so this field is empty
p4=                             # no 5th server, so this field is empty
f4=                             # no 5th server, so this field is empty
```

3. After parsing, **console.cgi** should connect to these servers. Note that the maximum number of the servers never exceeds **5**.

4. If we select N sessions, then you can assume them to be session 1 to session N.
   For example, we will **NOT** select session 1 + session 3 but skip session 2 during demo.

5. For the selected sessions, you can assume **host**, **port**, and **file** fields will **NOT** be empty.

6. The remote servers that **console.cgi** connects to are Remote Working Ground Servers with shell prompt "% " (NP Project2), and the files (e.g., t1.txt) contain the commands for the remote shells.

   However, you should not send the entire file to the remote server and execute them all at once. Instead, send one line whenever you receive a shell prompt "% " from remote.
   (You can assume the output of all commands will **NOT** contain **"%"**)

7. Your **console.cgi** should display the **hostname** and the **port** of the connected remote server at the top of each session.

8. Your **console.cgi** should display the remote server's replies in real-time. Everything you send to remote or receive from remote should be displayed on the web page as soon as possible.

   For example:

   ```
   % ls
   bin
   test.html
   ```

   Here, the blue part is the content (output) you received from the remote shell, and the brown part is the content (command) you sent to the remote. The output order matters and needs to be preserved. You should make sure that commands are displayed right after the shell prompt "% ", but before the execution result received from remote.

9. You should **NOT** change the order of outputs received from the remote servers.
   Besides, **DO NOT** add delay between commands.

10. Regarding how to display the server's reply (console.cgi), please refer to **sample_console.cgi**.
    Since we will not judge your answers with **diff** for this project, feel free to modify the layout of the web page. Just make sure you follow the below rules:

    (a) Each session should be separated.

    (b) The **commands** and the **outputs of the shell** are displayed in the right order and at the right time

    (c) The **commands** can be easily distinguished from the **outputs of the shell**.

2.3 panel.cgi (Provided by TA)

1. This CGI program generates the form in the web page. It detects all files in the directory **test_case/** and display them in the selection menu.

2.4 test_case/ (Provided by TA)

1. This directory contains test cases, and each of which lists the commands to run remotely. You can put new test cases into this directory, and select it in the form generated by **panel.cgi**.

2.5 np_single_golden (Provided by TA)

1. This executable file is a Remote Working Ground Server in project2. We will use it for demo. You do **NOT** need to use your code for this server.

2.6 The Execution Flow

2.6.1 Initial Setup

The structure of your working directory:

```
working_dir
    |-- http_server
    |-- console.cgi
    |-- panel.cgi
    |-- test_case/
```

2.6.2 Execution

1. Run your **http_server** by **./http_server** [**port**]

2. Open a browser and visit `http://[NP_server_host]:[port]/panel.cgi`

3. Fill the form with the servers to connect to and select the input file, then click **Run**.

4. The web page will automatically redirected to `http://[NP_server_host]:[port]/console.cgi` and your **console.cgi** should start now.

3 Requirements

1. You need to implement two programs in this part: **http_server** and **console.cgi**.
   Every function that touches network operations (e.g., DNS query, connect, accept, send, receive) **MUST** be implemented using the library **Boost.Asio**. Directly using low-level system calls such as ‘read‘, ‘write‘, ‘listen‘, are thereby **NOT** allowed.

2. All of the network operations should implement in **non-blocking (asynchronous)** approaches.

3. You must provide **Makefile**. After typing command **”make part1”**, **two executables** named **http_server** and **console.cgi** should be generated. The executables should be placed **in the top layer of the directory**.

4. You can only use **C/C++** to implement this project. Except for **Boost**, other third-party libraries are **NOT allowed**.

4 Submission

1. E3

   (a) Create a directory named as your student ID, and put your **Makefile** and **source codes in both part 1 and part 2** into the directory. Do **NOT** put anything else in it (e.g., **.git**, **__MACOSX**, **panel.cgi**, **test_case/**).

   (b) **Zip** the directory and upload the .zip file to the E3 platform
   Attention!! we only accept .zip format
   e.g. Create a directory 311551048, **zip the folder 311551048 into 311551048.zip**, and upload 311551048.zip to E3. The zip file structure may be like:

```
311551048.zip
    |-- 311551048 (dir)
        |-- Makefile
        |-- http_server.cpp   # created in part 1
        |-- console.cpp       # created in part 1
        |-- cgi_server.cpp    # created in part 2
        |(other source codes)
```

2. Bitbucket:

   (a) Create a **private** repository named as [your_student_ID]_np_project3 (e.g., 311551048_np_project3) inside the **nycu_np_2024** team, place it under the project **np_project3**, and set the ownership to **nycu_np_2024**.

   (b) You can push anything you need to Bitbucket, but please make sure to commit at least 5 times.

3. **We take plagiarism seriously.**

```
All projects will be checked by a cutting-edge plagiarism detector.
You will get zero points on this project for plagiarism.
Please don't copy-paste any code from the internet, this may be
considered plagiarism as well.
Protect your code from being stolen.
```

5 Notes

1. NP project should be run on NP servers, otherwise, your account may be locked.
2. Any abuse of NP server will be recorded.
3. Don't leave any zombie processes in the system.
4. You will lose points for violating any of the rules mentioned in this spec.
5. Enjoy the project!

6 Hints and Reminders

1. The version of Boost Library is **1.81.0** in nplinux server.

2. You can use the HTTP server that already hosted on NP servers to test your CGI programs. Simply put all of the CGI programs as well as **test_case/** into **~/public_html**, and then visit `http://[NP_server_host]/~[your_user_name]/[your_cgi_name].cgi`.

   Note that:

   - The filenames of CGI programs **MUST** end with **.cgi**.
   - **~/public_html** is a directory named "public_html" in your home directory.
     Manually create it if it does not exist.

3. You can use the command **nc** to inspect the HTTP request sent from browser:

   - Execute the command **nc -l [port]** on one of the NP servers (e.g., run **nc -l 8888** on nplinux3)
   - Open a browser and type **http://[host]:[port]** in the URL.
     You can add some query parameters and check the result. For example,
     `http://nplinux3.cs.nycu.edu.tw:8888/test.cgi?a=b&c=d`