Network Programming Project 3
HTTP Server & CGI

NP TA

Deadline: Sunday, 2021/12/05 23:55

1 Introduction

In this project, you are asked to implement 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
   (b) REQUEST_URI
   (c) QUERY_STRING
   (d) SERVER_PROTOCOL
   (e) HTTP_HOST
   (f) SERVER_ADDR
   (g) SERVER_PORT
   (h) REMOTE_ADDR
   (i) REMOTE_PORT

   For instance, if the HTTP request looks like:

```
GET /console.cgi?h0=nplinux1.cs.nctu.edu.tw&p0= ... (too long, ignored)
Host: nplinux8.cs.nctu.edu.tw:7779
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,applica ... (too long, ignored)
Accept-Language: en-US,en;q=0.8,zh-TW;q=0.5,zh; ... (too long, ignored)
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

```
Upgrade-Insecure-Requests: 1
```

Then before executing console.cgi, you need to set the corresponding environment variables. In this case, **REQUEST_METHOD** should be **"GET"**, **HTTP_HOST** should be **"nplinux8.cs.nctu.edu.tw:7779"**, and so on.

2.2 console.cgi

1. 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=nplinux1.cs.nctu.edu.tw&p0=1234&f0=t1.txt&h1=nplinux2.cs.nctu.edu.tw&
   p1=5678&f1=t2.txt&h2=&p2=&f2=&h3=&p3=&f3=&h4=&p4=&f4=
   ```

   It should be understood as:

   ```
   h0=nplinux1.cs.nctu.edu.tw    # the hostname of the 1st server
   p0=1234                       # the port of the 1st server
   f0=t1.txt                     # the file to open

   h1=nplinux2.cs.nctu.edu.tw    # the hostname of the 2nd server
   p1=5678                       # 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
   ```

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

3. 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.

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

5. 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 **"%"**)

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

7. 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 after the shell prompt "% ", but before the execution result received from remote.

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

9. 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 separate.

   (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**.

10. If you use an additional html file, we will **NOT** copy it to the working directory during demo.
    Please make sure your code can work **without** copying the html file to the working directory.

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. Run remote servers (**np_single_golden**)
4. Fill the form in **panel.cgi** with the host, port, and input file, then click **Run**.
5. 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: **http_server** and **console.cgi**.
   The following network operations **MUST** be implemented using the library **Boost.Asio** and in **asynchronous** approaches.
   - The connection between **http_server** and client (e.g., accept, send, receive)
   - The connection between **console.cgi** and remote servers (e.g., DNS query, connect, send, receive)
2. You can only use **C/C++** to do this project. Except for **Boost**, other third-party libraries are **NOT** allowed.
3. We will use NP servers for demo. **Make sure your npshell can be executed in NP servers.**

4 Submission

1. E3
   (a) Create a directory named your **student ID**, and put **ONLY Makefile** and **source codes** into the directory.
   DO NOT put anything else in it (e.g., http_server, console.cgi, panel.cgi, test_case/, .git, __MACOSX).
   (b) You must provide **Makefile**. After typing command **"make"**, **two executables** named **http_server** and **console.cgi** should be generated. The executables should be placed **in the top layer of the directory**.

(c) **zip** the directory and upload the .zip file to E3
Attention!! we only accept .zip format
e.g. Create a directory 0856000, the directory structure may be like:

```
0856000
    |-- Makefile
    |-- http_server.cpp
    |-- console.cpp
    |(other source codes)
```

Zip the folder 0856000/ into 0856000.zip, and upload 0856000.zip to New E3.

2. Bitbucket:

(a) Create a **private** repository with name: **${Your_Student_ID}_np_project3** inside the workspace **nycu_np_2021** and the project **np_project3**.

`e.g., 0856000_np_project3`

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

5 Notes

1. We take plagiarism seriously. **You will get zero points on this project for plagiarism.**
2. You will lose points for violating any of the rules mentioned in this spec.
3. NP projects should be run on NP servers. Otherwise, your account may be locked.
4. Any abuse of NP server will be recorded.
5. Do not leave any zombie processes in the system.

6 Hints and Reminders

1. The version of Boost Library is **1.77.0** 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.nctu.edu.tw:8888/test.cgi?a=b&c=d`