

Network Programming Project 4

HTTP Server and CGI Programs

NP TA

Deadline: Sunday, 2025/5/11 23:59

1 Introduction

The goal of this project is to build a **Remote Batch System** using **Boost.Asio**. It has **two parts** that basically do the same thing from the user's point of view, but they're built on different platforms and have different integration setups.

Part 1: Linux Version

In this part, you are asked to implement a simple HTTP server and a CGI program on the **NP Linux Workstation**. This includes:

- `http_server`: A basic HTTP server using Boost.Asio.
- `console.cgi`: A CGI program to handle web-based console.

Part 2: Windows Version

This part requires developing a single executable called `cgi_server.exe` on **Windows 10/11**, which combines all the functionalities from Part 1 into:

- `cgi_server.exe`: A unified application that combines the roles of `http_server`, `panel.cgi`, and `console.cgi`.

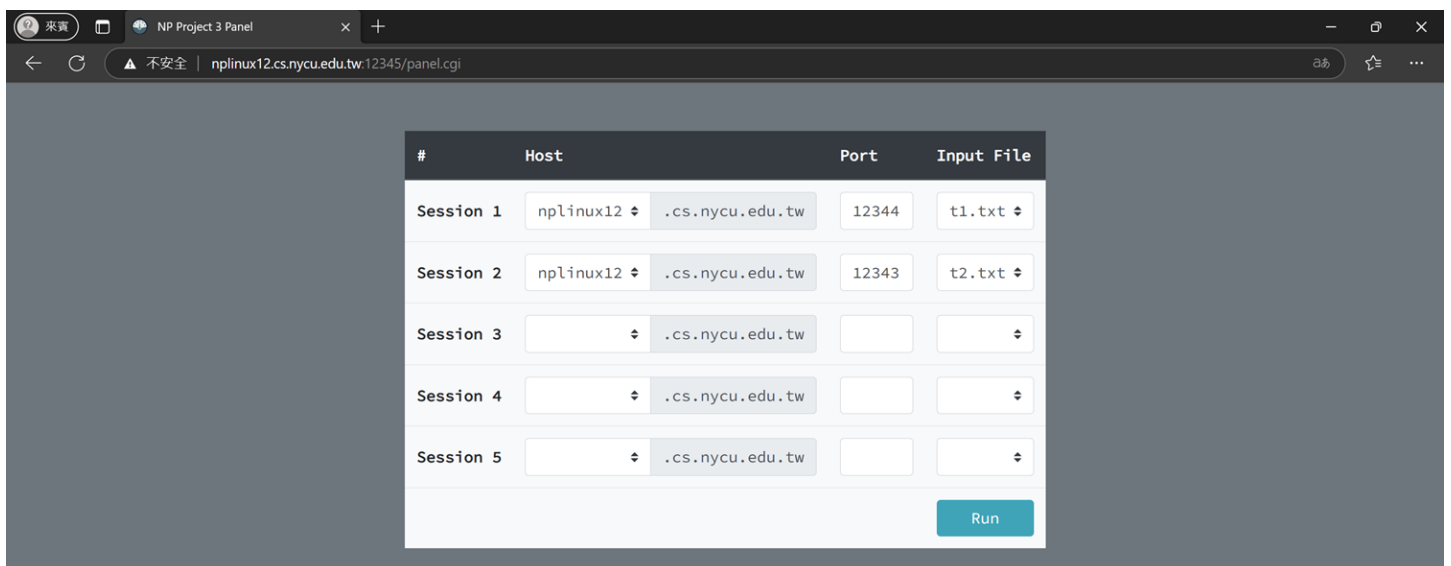
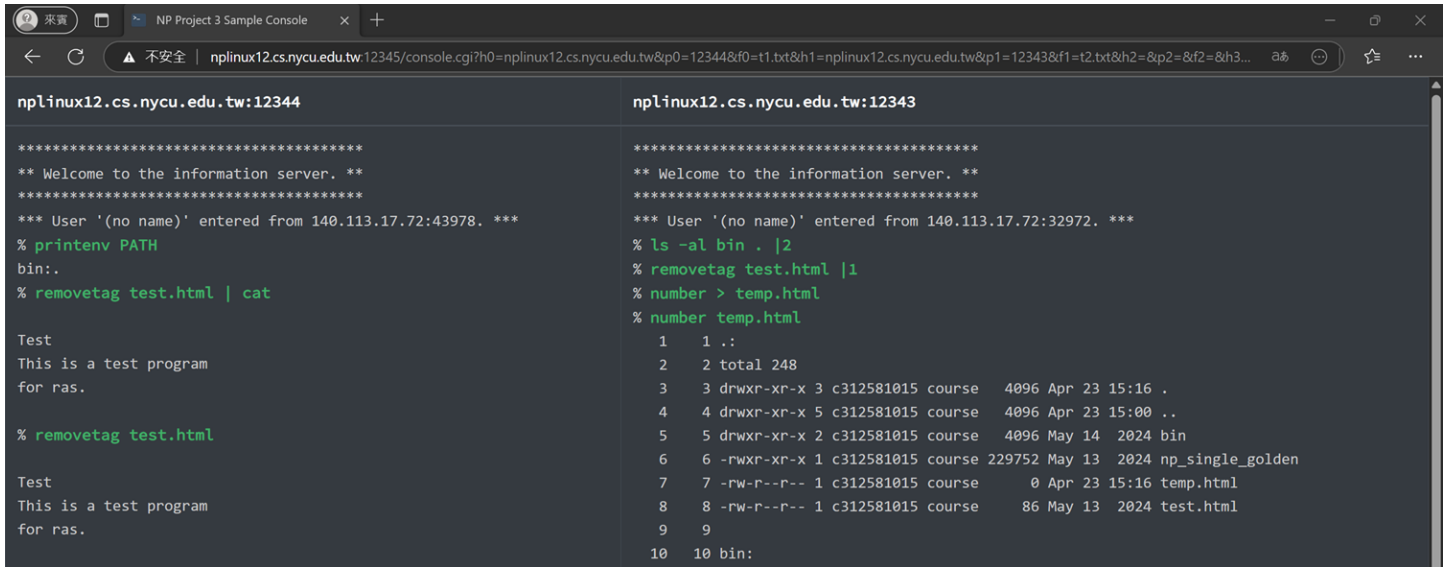


Figure 1: panel.cgi



```
*****
** Welcome to the information server. **
*****
*** User '(no name)' entered from 140.113.17.72:43978. ***
% printenv PATH
bin:.
% rmovetag test.html | cat

Test
This is a test program
for ras.

% rmovetag test.html

Test
This is a test program
for ras.
```

```
*****
** Welcome to the information server. **
*****
*** User '(no name)' entered from 140.113.17.72:32972. ***
% ls -al bin . | 2
% rmovetag test.html | 1
% number > temp.html
% number temp.html
1 1 .:
2 2 total 248
3 3 drwxr-xr-x 3 c312581015 course 4096 Apr 23 15:16 .
4 4 drwxr-xr-x 5 c312581015 course 4096 Apr 23 15:00 ..
5 5 drwxr-xr-x 2 c312581015 course 4096 May 14 2024 bin
6 6 -rwxr-xr-x 1 c312581015 course 229752 May 13 2024 np_single_golden
7 7 -rw-r--r-- 1 c312581015 course 0 Apr 23 15:16 temp.html
8 8 -rw-r--r-- 1 c312581015 course 86 May 13 2024 test.html
9 9
10 10 bin:
```

Figure 2: console.cgi

2 Specification

2.1 (part 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=nplinux12.cs.nycu.edu.tw&p0= ... (too long, ignored)
Host: nplinux12.cs.nycu.edu.tw:12345
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/javascript ... (too long, ignored)
Accept-Language: en-US,en;q=0.8,zh-TW;q=0.5,zh;q=0.4 ... (too long, ignored)
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

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 "nplinux12.cs.nycu.edu.tw:12345", and so on and so forth.

2.2 (part 1) 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 (part 2) cgi_server.exe

1. The **cgi_server.exe** accepts TCP connections and parse the HTTP requests (as **http_server** does), and we will only test for the HTTP GET method.
2. You don't need to fork() and exec() since it's relatively hard to do it on Windows. Simply parse the request and do the specific job **within** the same process. We guarantee that in this part the URI of HTTP requests will be **"/panel.cgi"** or **"/console.cgi"** plus a query string:
 - (a) If it is **/panel.cgi**,
Display the panel form just like **panel.cgi** in part 1. This time, you can hard code the input file menu (t1.txt ~ t5.txt).
 - (b) If it is **/console.cgi?h0=...**,
Connect to remote servers specified by the query string. Note that the behaviors **MUST** be the same as part 1 **in the user's point of view** (though the procedure is different in this part).

2.4 panel.cgi (Provided by TA)

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.

2.5 test_case/ (Provided by TA)

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**.

2.6 np_single_golden (Provided by TA)

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.

2.7 The Execution Flow

2.7.1 Initial Setup

The structure of your working directory:

(Part 1)

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

(Part 2)

```
working_dir
|-- cgi_server.exe
|-- test_case/
```

2.7.2 Execution

1. (In part 1) Run your **http_server** by `./http_server [port]`
(In part 2) Run your **cgi_server.exe** by `cgi_server.exe [port]`
2. Run your **console servers** by `./np_single_golden [port]`
3. Open a browser and visit [http://\[NP_server_host\]:\[port\]/panel.cgi](http://[NP_server_host]:[port]/panel.cgi)
4. Fill the form with the servers to connect to and select the input file, then click **Run**.
5. The web page will automatically redirected to [http://\[NP_server_host\]:\[port\]/console.cgi](http://[NP_server_host]:[port]/console.cgi) and your **console.cgi** should start now.

3 Requirements

1. You need to implement three programs in project: **http_server**, **console.cgi**, and **cgi_server.exe**. 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. Build requirements:
 - For **Part 1** (Linux):
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**.
 - For **Part 2** (Windows):
Please also provide a **Makefile** that builds **cgi_server.exe** via the command "**make part2**". The executable must also reside in the top directory.

※ We will use the **MinGW 19.0 distribution** from <https://nuwen.net/mingw.html> to compile and test your **cgi_server.exe**.
Below is an example compilation command under MinGW:

```
g++ cgi_server.cpp -o cgi_server.exe -lws2_32 -lwsck32 -std=c++14
```
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 312581015, **zip the folder 312581015 into 312581015.zip**, and upload 312581015.zip to E3. The zip file structure may be like:

```
312581015.zip
|-- 312581015 (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. Github Classroom:
 - (a) Make sure that you have accepted the invitation to Project 4.
 - (b) You can push anything you need to Github, 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.

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](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>