Network Programming Project 3 (Part 1)
Remote Batch System

NP TA

Deadline: Monday, 2019/12/16 23:59

1 Introduction

The project is divided into two parts. This is the first part of the project.
Here, you are asked to write a Remote Batch System called **console.cgi** and a simplified HTTP server called **http_server**. Given that there already exist numerous open-source libraries for networking and **Boost.Asio** is one of the best candidates among them, we will (and you must) use it to accomplish this project.

2 Specification

2.1 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 samples.

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 2.2).

   For example, if **QUERY_STRING** is:

   ```
   h0=nplinux1&p0=1234&f0=t1.txt&h1=nplinux6&p1=5678&f1=t2.txt&h2=&p2=&f2=&h3
   =&p3=&f3=&h4=&p4=&f4=
   ```

   It should be understood as:

   ```
   h0=nplinux1       # the hostname or IP of the 1st server
   p0=1234           # the port of the 1st server
   f0=t1.txt         # the file to open

   h1=nplinux6       # the hostname or IP 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=
   f2=
   ```

```
h3=
p3=
f3=

h4=
p4=
f4=
```

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

4. In this project, the remote servers we connect to are chat room servers with shell prompt "% " (NPPJ2), and the files we sent (e.g., t1.txt) are 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 them line-by-line whenever you receive a shell prompt "% " from remote.

5. Your CGI program should display the remote server's replies in real-time. I.e., 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 need to be preserved, you should make sure that commands are displayed right after the shell prompt "% ", but before the execution result received from remote.

6. Regarding how to beautifully 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 style up the web page. Just make sure the **commands** are displayed in the right order, at the right time, and are easy to distinguish from the **output of the shell** (we use bold <b> for commands in the sample).

2.2 http_server

1. You probably need to have a refresh on how **HTTP** works before starting this section.

2. The **http_server** is in charge of accepting TCP connections and parsing the HTTP requests.
   In this project, the URI of HTTP requests will always be in the form of /${cgi_name}.cgi (e.g., /panel.cgi, /printenv.cgi), and we will only test for the HTTP GET method.
   Your **http_server** should parse the HTTP headers and execute the cgi programs (which was specified by the HTTP request) with the CGI procedure (fork, set environment variable, dup, exec).

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 exec() 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", and so on and so forth.

2.3 panel.cgi (Provided by TA)

1. Written in Python 3. But sorry, you can not use Python in this project :)
2. This CGI generate the web page dynamically. It detects all files in the directory **test_case** and display them in the selection menu.

2.4 test_case/

1. Contains test cases that are literally the commands going to run remotely. You can put new test cases into this directory, and select if by panel.cgi.

2.5 The Execution Flow

2.5.1 Initial Setup

The structure of your working directory:

```
working_dir
    |----- console.cgi       # remote batch system
    |----- http_server       # HTTP server
    |----- panel.cgi         # provided by TA
    |----- test_case/        # test cases to run remotely
```

2.5.2 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 your want 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 Remote Batch System (console.cgi) should start now.

3 Requirements

1. You need to implement two programs in this part: **console.cgi** and **http_server**.
   Every function that touches networking 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 a **Makefile**, which compiles your source code into two **executables** named **http_server** and **console.cgi**, respectively. The executables should be placed in the same directory as the source codes. We will use them for demo.

4 About Submission

1. E3
   (a) Create a directory named as your student ID, and put **ONLY** your code and Makefile in the **same directory layer**. 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 0856000, the directory structure may be like:

```
0856000
    |-- Makefile          # created in part 1
    |-- http_server.cpp   # created in part 1
    |-- console.cpp       # created in part 1
    |-- main.cpp          # created in part 2
    |(other source codes)
```

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

2. Bitbucket:
   (a) Create a **private** repository named as [your_student_ID]_np_project3 (e.g., 0856000_np_project3), set it's owner to **nctu_np_2019**, and place it under the project **np_project3**.
   (b) You can push anything you need to Bitbucket, but please make sure you've committed 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. 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`.

   For example: `http://nplinux1.cs.nctu.edu.tw/~tzuyee/panel.cgi`.

   Note that:

   - the 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.

2. You can use the command line tool **nc** to inspect the HTTP request sent from the 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. Consider add some query parameters and check the results, for example:
     `http://nplinux3.cs.nctu.edu.tw:8888/test.cgi?a=b&c=d`

3. Consider **C++11's** `<regex>` and **Boost.StringAlgorithms** for string parsing and manipulation.