Due before:	11:59 PM (before Midnight) on Wed Oct 17 2019

CSE-381: Systems 2

Homework #5

Due: Wed October 17 2019 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, Oct 16 2019

Maximum Points: 25

Instructions

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your C++ source code is named `MUID_hw5.cpp`, where `MUID` is your Miami University Unique ID. Ensure your program compiles without any warnings or style violations. Ensure you thoroughly test operations of your program **using a web-browser**. Once you have tested your implementation, upload the following onto Canvas:

1. The 1 C++ source file developed for this homework.

General Note: Upload each file associated with homework individually. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Objective

The objective of this homework is to:

- Recollect (from CSE-278) the use of HTTP protocol for communication
- Develop a web-server that can process HTTP GET requests
  - Your web-server should be able to run a program with command-line arguments and return its outputs back to the client
- Continue to gain familiarity with I/O streams & string processing

Grading Rubric:

The programs submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements will be assigned zero score!

Program that do not compile, have a method longer than 25 lines or badly formatted code as in } } }, etc., or just some skeleton code will be assigned zero score.

1. **5 points** are reserved for good overall program structure, organization, **conciseness**, documentation. **If your methods are not concise or undocumented points will be deducted**.
2. **-1 Points**: for each warning generated by the compiler (warnings are most likely sources of errors in C++ programs)

NOTE: Violating CSE programming style guidelines is a compiler error! Your program should not have any style violations.

Background

In this homework you will be developing a simple program to process HTTP requests and generate responses. Recollect that HTTP is a multi-line text protocol that is used by World Wide Web (WWW), an application running on the Internet. In this homework you will be developing a program to respond to HTTP requests. **Overall this program is essentially just string processing from I/O streams with most of the required code copy-pasted from different exercises/lecture slides.**

Testing via a browser

The starter code enables the program to run as a web-server on an available port number. The starter code prints the port number, say `3456`. Each time the program is run, the port number will change. Given a port number, say `3456`, you **must test the program via a web-server** by suitably changing the command and arguments in the following URL:

1. http://os1.csi.miamioh.edu:**3456**/cgi-bin/exec?cmd=ls&args=-l
2. http://os1.csi.miamioh.edu:**3456**/cgi-bin/exec?cmd=echo&args="hello"
3. http://os1.csi.miamioh.edu:**3456**/cgi-bin/exec?cmd=ls&args=/blah
4. http://os1.csi.miamioh.edu:**3456**/blah.txt

Chunked Response Requirements [Review from lecture slides]

The response/output from the program/server should be in the following format, with each line terminated with a `"\r\n"`:

1. The first line should be `HTTP/1.1 200 OK` or `HTTP/1.1 404 Not Found` (if the `GET` request was not in expected format). See example inputs
2. The second line should be `Content-Type: text/plain` .
3. The next line must be `Transfer-Encoding: chunked` (a fixed constant string)
4. The next line must be `Connection: Close` (a fixed constant string)
5. The header section must be terminated by `"\r\n"`

Rest of the contents of the output of a program are printed line-by-line with line size (in hex on a separate line) followed by the data as shown below:

```
// Send chunk size to client.
os << std::hex << line.size() << "\r\n";
// Write the actual data for the line.
os << line << "\r\n";
```

The end of data must be sent to the client via the following trailer:

```
// Send trailer out to denote end of data
os << "0\r\n\r\n";
```

Request Processing Requirements

You are required to implement the `serveClient` method in the starter code. This method must be implemented (using additional helper methods to be implemented by you) to process multiple lines of one (and only one) HTTP GET request in the following manner:

1. **Base case – Run program & return its outputs [15 points]**: If input line starts with "`GET /cgi-bin/exec`" then this line corresponds to the case where a program is to be run. In this case, this line will be in the format: `GET /cgi-bin/exec?cmd=ls&args=1.txt "hello.cpp" HTTP/1.1` (one line), where `cmd` contains the command to run and `args` has list of space separated command-line arguments to the program. Use `std::quoted` to correctly handle double quoted words in `args`. Note that you need to `url_decode` `cmd` and `args` prior to using them as they will be encoded (for an example see request in `test_files/cgi_bin_test_inputs1.txt`). URL decoding method is included in starter code.

   Appropriately process the `GET` request, run the specified command, obtain its outputs using pipes (see slide #19, #20 in `07_IPC.pdf`, you will need to suitably use simple pipes example but run 1 process and read its outputs using a pipe) and generate the HTTP response using chunked encoding.

   The last line of the output should include the exit code from the program being run. The response payload is finally terminated with a "`0\r\n\r\n`".

   For example, given the request `http://os1.csi.miamioh.edu:3456/cgi-bin/exec/cmd=echo&args="hello\\n" "world"`, your program should generate the following output. Note: The `\r\n` is shown to help with non-printable characters. The `\n` is what causes output to appear in next line. Some of the commands print their own newlines and those are not explicitly shown (as your program should not be printing those anyways).

```
HTTP/1.1 200 OK\r\n
Content-Type: text/plain\r\n
Transfer-Encoding: chunked\r\n
Connection: Close\r\n
\r\n
6\r\n
hello
\r\n
7\r\n
 world
\r\n
c\r\n
Exit code: 0\r\n
0\r\n
\r\n
```

2. **Error case [5 points]**: If the path in the GET request is not `"GET /cgi-bin/exec"` then the program should generate an HTTP 404 error response with message `"Invalid request: <path>"`. For example, given the request `http://os1.csi.miamioh.edu:3456/blah.txt`, the following output should be generated:

```
HTTP/1.1 404 Not Found\r\n
Content-Type: text/plain\r\n
Transfer-Encoding: chunked\r\n
Connection: Close\r\n
\r\n
1a\r\n
Invalid request: blah.txt
\r\n
0\r\n
```

3. **5 points** are reserved for good overall program structure, organization, **conciseness**, documentation. **If your methods are not concise or undocumented points will be deducted**.

Sample inputs & outputs

Sample inputs and outputs are not shown because the HTTP headers contain line endings in the form `"\r\n"` which are not preserved in documents and leads to incorrect results if copy-pasted. Instead use the supplied input files and copy-paste from those input files.

Functional testing

When testing your program during development, simply copy-paste the inputs to your program (mimicking as if you manually typed the inputs). Once you have done the basic testing you may redirect input and output for testing as shown below. Several test inputs and expected output files are also supplied to streamline your testing as shown below:

```
$ ./hw5 test_files/base_case1_inputs.txt > my_base_case1_output.txt
$ diff my_base_case1_output.txt test_files/base_case1_expected_outputs.txt
```

Note: If your solution is correct then the above `diff` command will not print any output.

Ensure you test with all input files and corresponding expected outputs to check all features of your program.

Submit to Canvas

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your program compiles (without any warnings or style errors) successfully. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload the following onto Canvas:

- The 1 C++ source file you developed for this homework