

## **CSE278: Introduction to Systems Programming (System I)**

### **Homework #5**

Max Points: 50

You should save/rename this document using the naming convention **MUId.docx** (example: ahmede.docx).

**Objective:** The objective of this exercise is to:

1. Experiment with HTTP protocol using `telnet` and `wget`.
2. Experiment with a simple, custom server
3. Experiment and modify a simple web-server.

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from the terminal/output window into this document. You may discuss the questions with your instructor.

**Name:** Ben Hilger



**Wait for your instructor to indicate when you can start working on each part of this exercise.** Your instructor will cover necessary concepts prior to working on this laboratory exercise. Your instructor may ask you work on one question at a time before moving on to the next question.

### **Part #1: Working with a custom server**

*Estimated time: 15 minutes*

**Background:** A server is a standard program, except that instead of reading/writing to standard input streams (`std::cin` and `std::cout`), it reads/writes to a socket stream. So all of the processing logic and operations of a server are exactly the same as any other program.

**Exercise:** In this exercise you are expected to experiment with a simple, custom server (that simply echoes input from the user with an added text), via the following procedure:

1. **You will not be using NetBeans.** Download and `scp` (or use **CyberDuck**) the supplied `sassy.cpp` server to `osl.csi.miamioh.edu`.
2. On your local computer study the code in the `process` method while nothing the following:
  - a. Note how the `process` methods reads inputs and outputs from abstract streams. This enables the `process` method to be used with any type of stream including, BOOST-library's `tcp::iostream`.
  - b. Note the do-while loop, which processes line-byline.

- c. Note the check to clean-up '`\r`' (carriage return) character at the end of the line. This is because most text protocols on the Internet use "`\r\n`" for end-of-line (and not just "`\n`").
- d. You should be able to answer the following question – "When does the do-while loop end?"

- i. The do-while will end when the user enters "bye"

- 3. Compile and run the supplied program (note the `-lboost_system` to link against boost library) as shown below and note the **port** number:

```
$ g++ -g -Wall -std=c++14 sassy.cpp -o sassy -lboost_system
$ ./sassy
```

- 4. Leave the server running. Now, **open a different** Powershell (or Terminal) window. From the new window use `telnet` to send inputs (shown in **bold** below) and display outputs to the sever as shown below. Note that we can run `telnet` on any other machine (obviously, the computer needs to have `telnet` installed on it) and connect to the server.

```
C:\> telnet os1.csi.miamioh.edu 45349
Trying 127.0.1.1...
Connected to os1.csi.miamioh.edu.
Escape character is '^]'.
Yeah, what do you want to know?
what is the weather?
Really? Ask www.google.com about 'what is the weather?'
Yeah, what do you want to know?
bye
What! did you just hangup on me?
Connection closed by foreign host.
```

- 5. Next, ask your neighbor to run their server and connect to it via `telnet` and send inputs. Note that you would be able to connect any ones server if you know their IP address and port number. This idea applies to the whole of the Internet as well.



Ensure you form a strong, clear mental model. Client and server are just standard programs, with the following differences:

- 1. They send and receive inputs over sockets instead of standard I/O. However, the data processing, logic etc. is the same!
- 2. Client and server could be running on different machines.

- 6. Place a screenshot of the terminal showing interactions with the server in the space below:

```
Swap usage:      25%

Last login: Fri Feb  7 14:36:31 2020 from 172.25.55.230
[hilgerbj@ceclnx01:~$ telnet os1.csi.miamioh.edu 43683
Trying 172.17.0.58...
Connected to os1-f13.csi.miamioh.edu.
Escape character is '^]'.
Yeah, what do you want to know?
[what is the weather?
Really? Ask www.google.com about 'what is the weather?'
Yeah, what do you want to know?
[bye
What! did you just hangup on me?
Connection closed by foreign host.
hilgerbj@ceclnx01:~$
```

Screenshot of terminal with telnet command goes here

## Part #2: Understanding a simple web-server

*Estimated time: 15 minutes*

**Background:** A web-server is an application layer program that loops forever, performing the following tasks in each iteration:

1. Listens on a given port number (typically 80, but port number can vary)
2. Accepts incoming connections from a client (such as: web-browser or mobile app etc.)
3. Serves the client by sending a response. Typically response is contents of a file requested by the client.

**Exercise:** Studying a custom web-server:

1. **You will not be using NetBeans.** Download and scp (or use Cyberduck) the supplied `Homework5.cpp` server to `os1.csi.miamioh.edu`.
2. Study the main method to note the following key aspects:
  - a. How the port number is determined
    - i. It is determined by selecting any free port
  - b. The use of boost namespaces for convenience
  - c. Strategy for creating an endpoint on localhost
    - i. `myEndpoint(tcp::v4(), 0)`, where 0 represents any free port
  - d. Creating an `tcp::acceptor` object called `server` to accept connections from a client
    - i. `server(io_service, tcp::endpoint)`
  - e. The `while`-loop that loops forever (as per a web-server's operation) at accepts a client connection and processes the data.
3. Next study the `serveClient` method while noting the following aspects:

- a. What is the input and output stream being passed to this method?:
  - i. The input stream is the read data of the client
  - ii. The output stream is to send data to the client
- b. Note that the first line is the GET request line. It is of the form "GET file HTTP/1.1". In this simple example, we are ignoring the actual file the user requested. Of course, you will get to implement this feature (of returning data from a file) in the next homework.
- c. Rest of the HTTP request lines are headers from the browser. In this example we are ignoring those headers.
- d. How is the end of HTTP headers detected in the `while`-loop? It's found by either an empty string or the `"\r"` character.
  - i. Why does the while-loop check for `" "` (empty string) and `"\r"` and not `"\n"` or `"\r\n"`?
    1. Because some HTTP lines only have `"\r"`, which indicates the end of a header
- e. The approach to send the HTTP response back to the client:
  - i. Note how each HTTP header line end with `"\r\n"`
  - ii. Note the extra `"\r\n"` to end the indicate end of HTTP headers
  - iii. The data sent does not need the newline requirement because the data can be any information (text or binary).
- f. Note how the number of bytes of response data is determined.

### Part #3: Running the simple web-server

*Estimated time: 10 minutes*

**Exercise:** Compile, run, and test the web-server via the following procedure:

1. Compile the supplied starter code using `g++` as follows:

```
$ g++ -g -Wall -std=c++14 Homework5.cpp -o Homework5 -  
lboost_system  
$ ./Homework5
```

2. Run the web-server via a BASH Terminal. It will bind to a port and print the port number.
3. Once your server starts running, using your web-browser on your local laboratory computer access your web-server via the following URL (where **Port** is the port number you are using):

```
http://os1.csi.miamioh.edu:Port/
```

4. You should be able to see a simple message in your web-browser.
5. Next, request your neighbor to run their web-server. Using your neighbor's port number and the above URL, try to access their web-server. Note how multiple web-servers are running on the same machine, but at different port numbers.
6. Congratulations! You are now running your own web-server. This is how all real web-servers are also programmed to run, that is:
  - a. Most of them are C/C++ programs
  - b. They are run just like you run your program
  - c. Unlike the simple server, they perform more operations and commands based on HTTP protocol

## Part #4: Modify the simple web-server

*Estimated time: 20 minutes*

**Exercise:** Modify the web-server to generate a different message and test it using the following procedure:

1. First ensure your web-server is not running from the previous task.
2. Modify the existing message in the web-server code to correspond to the message shown in `ex.html` file supplied along with this exercise.
3. Modify the content type of the response to be `text/html` (instead of `text/plain`).
4. Run your web-server on a specific port similar to the previous task. Use a web-browser to check as in the previous task. Note: You should see HTML now instead. This is because we can see the content type above.
5. Now let's continue to remember the handy `wget` command. From a **different terminal connected ssh to** `osl.csi`, (ensure that understand and note that your web-server needs to continue to run during this testing procedure) test correct operation of your program using `wget` and `diff` as shown below:

```
$ wget -q "http://osl.csi.miamioh.edu:Port/" -O my_ex.html
$ diff my_ex.html ex.html
```

Where, `wget` is a simple console program that acts as web-browser to GET data from a any given URL (you can use `wget` with `google.com` or `facebook.com` for that matter) with the following options:

- The `-S` option tells `wget` to print headers sent by the web-server
- The `-q` option tells `wget` to be as quiet (not print additional info)
- The `-O` option tells `wget` to write the data from the web-server to a given file.

In the above command, `diff` (that displays differences) should not generate any output indicating the data generated by your web-server is correct.

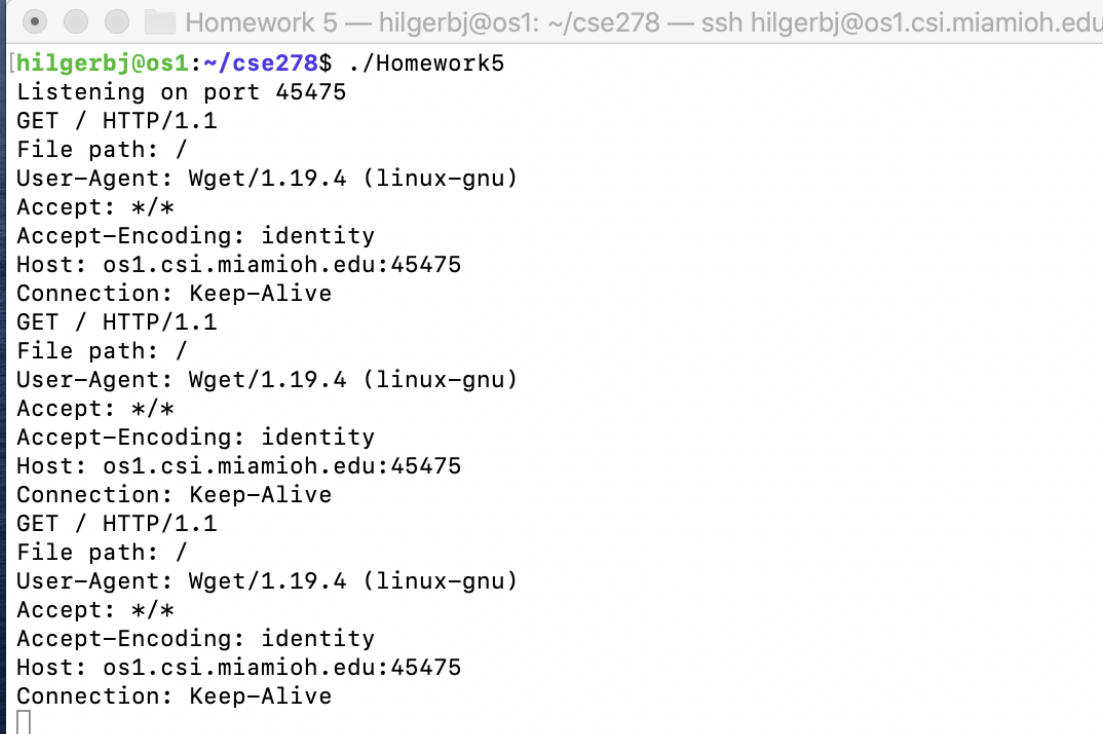
6. The previous step does not verify that the HTTP headers (that contains vital data such as the size, Mime type etc.) generated by your program is correct. Let's do that as well by redirecting wget's header output as shown below:

```
$ wget -S -q "http://os1.csi.miamioh.edu:Port/" -O my_ex.html 2> my_hdrs.txt
$ diff my_ex.html ex.html
$ diff my_hdrs.txt ex_headers.txt
```

Needless to add, both of the diff commands should not generate any output indicating that your program is operating correctly. Copy paste a screenshot showing the above 3 commands correctly running in the space below:



```
Homework 5 — hilgerbj@os1: ~/cse278/Homework 5 — ssh hilgerbj@os1.csi.miamioh.edu
[hilgerbj@os1:~/cse278/Homework 5$ wget -q "http://os1.csi.miamioh.edu:45475/
my_ex.html
[hilgerbj@os1:~/cse278/Homework 5$ diff my_ex.html ex.html
[hilgerbj@os1:~/cse278/Homework 5$ wget -S -q "http://os1.csi.miamioh.edu:454
-O my_ex.html 2> my_hdrs.txt
[hilgerbj@os1:~/cse278/Homework 5$ diff my_ex.html ex.html
[hilgerbj@os1:~/cse278/Homework 5$ diff my_hdrs.txt ex_headers.txt
[hilgerbj@os1:~/cse278/Homework 5$ ]
```



```
Homework 5 — hilgerbj@os1: ~/cse278 — ssh hilgerbj@os1.csi.miamioh.edu
[hilgerbj@os1:~/cse278$ ./Homework5
Listening on port 45475
GET / HTTP/1.1
File path: /
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: os1.csi.miamioh.edu:45475
Connection: Keep-Alive
GET / HTTP/1.1
File path: /
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: os1.csi.miamioh.edu:45475
Connection: Keep-Alive
GET / HTTP/1.1
File path: /
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: os1.csi.miamioh.edu:45475
Connection: Keep-Alive
]
```

### **Part #5: Submit to Canvas**

- No late assignments will be accepted!
- This work is to be done individually
- This MS-Word document (duly filled-in) saved as a PDF document.
- The submission file will be saved with the name ***Homework5\_yourMUID.pdf***
- Submit all the modified code ***Homework5\*\_yourMUID.cpp***
- Assignment is due Friday, March 20, 2020 before Midnight
- On or before the due time, drop the *electronic copy* of your work in the *canvas*

Don't forget to Turn in the files! Homework5\_yourMUID.pdf & Homework5\*\_yourMUID.cpp