CS3516 (A-Term 2020)

Programming Assignment 1: Socket Programming

Due Date
Thursday September 17, 2020 (11:59pm)

Total Points
100 (One Hundred)

Goal
In this assignment you will be asked to implement an HTTP client and server running a simplified version of the HTTP/1.1 protocol. This project will be completed in either C/C++ using the Unix Socket commands. The program has to run without errors in any WPI machine, e.g., linux.wpi.edu. See more details of how to access WPI Linux servers at the link below.
http://users.wpi.edu/~yli15/courses/CS3516Fall20A/BestPaper.html

Step 1: The HTTP Client

Execution: Your client should take command line arguments and the GET method to download a web-page. The arguments are as follows:

- The **server URL** (http://www.server.com/path/to/file) or an IP address
- The **port number** on which to contact the server
- The **options** that which add functionality to the client
  - `-p`: prints the RTT for accessing the URL on the terminal before server's response (i.e., the webpage)

For example
```
./http_client [-options] server_url port_number
```

Workflow: The client should work as follows:

- Connect to the server via a TCP socket
- Submit a valid HTTP/1.1 GET request for the supplied URL.
  - Note that: HTTP expects all lines in a request to be terminated with a \r\n. The last line of the request should only contain a \r\n (i.e., empty line)
  - Read the server's response and display it on the terminal
- Submit a valid HTTP/1.1 GET request for the supplied URL with the –p option.
  - Print the RTT for accessing the URL on the terminal
  - Read the server's response and display it on the terminal

Testing: Use the client to get a file of your choosing from an actual webserver on the Internet. For example,
```
./http_client www.google.com 80
```

Use the client to get the provided index.html from your own server program. For example,

```
./http_client linux.wpi.edu/index.html 7890
```

Use the client to get a file of your choosing an *actual* webserver (of your choosing) on the Internet and *your own* web server using the –p option to compute the RTT between you and the webserver. For example,

```
./http_client -p www.google.com 80
./http_client -p linux.wpi.edu/index.html 7890
```

- Do this 10 times and compute the average RTT time for both cases. Create a table and show each of the RTT values and the average value.
- You should print the RTT value in milliseconds.

Step 2: The HTTP Server

Execution: Your server should start first (i.e., before the client) and take a command line argument specifying a port number. For example,

```
./http_server 7890
```

Workflow: The basic workflow of the server is as follows

- Start the server in an infinite loop (you must interrupt your server with a termination signal (e.g., SIGTERM) to stop it)
- Wait for a client connection on the port number specified by command line argument.
- When a client connection is accepted, read the HTTP request.
- Construct a valid HTTP response including status line, any headers you feel are appropriate, and, of course, the requested file in the response body.
    - For example, if the server receives the "`GET index.html HTTP/1.1`" request, it sends out "200 OK" to the client, followed by the file index.html.
    - The file index.html has been provided.
    - If the requested file doesn't exist, the server sends out "404 Not Found" response to the client.
- Close the client connection and loop back to wait for the next client connection to arrive.
- Make sure your server code shuts down gracefully when terminated. That means closing any open sockets, freeing allocated memory, etc.
- **[BONUS POINTS:]** Make the server multi-threaded, which means it can handle as multiple clients at the same time. We will test with at least 5 to 10 clients. (10 extra points)
- The link to the file **TMDG.html** to be downloaded from your implementation of the web server is available in project documents. Use this as your index.html file at the server.

Testing: Run your Client on a WPI Linux server (e.g., linux.wpi.edu) to connect to the web server you just setup, for example,

```
./http_client linux.wpi.edu/TMDG.html 7890
```

At Client side, it should be able to get a "200 OK" message, and download the TMDG.html from the server side successfully.

Deliverables

1. All the code for the web client and the web server (.c and .h files)
2. A *makefile* that compiles the client and the server code (Tutorial on writing a makefile)
3. A README (txt) file that spells out exactly how to compile and run the client and the server

4. A PDF file that lists the RTT for an actual web server you connected to. We will test your client program using www.google.com and www.mit.edu

Grading

Grade breakup (%) assuming all documents are present:

1. Web client = 45 points
2. Web server = 40 points (BONUS Question: 10 points)
3. RTT results = 10 points
4. makefile = 5points

The grade will be a **ZERO** if:

1. If the code (client or server) does not compile or gives a run-time error
2. If README with detailed instructions on compiling the running the code is not present
3. If makefile is erroneous and the README does not provide a way tell us how to compile and run the code

To get the **FULL CREDIT** for the Web client and Web server:

1. On Client side, the client should obtain the entire html file from the server it is talk to, when its HTTP request succeeds.
2. On Server side, the entire TMDG.html test file can be successfully fetched and saved at a Client program.

Notes

- The HTTP 1.1 standard is available at http://www.w3.org/Protocols/rfc2616/rfc2616.html. Use this to make your client robust in handling the responses you get from different sites while displaying the page. For example, some site specify the size of the page they are returning in the header instead of strictly following the /r/n rule mentioned above.
- Beej's is an excellent tutorial for socket programming the old-fashioned way (http://beej.us/guide/bgnet/). Check it out.
- You must chose a server port number larger that 1023 (to be safe, choose a server port number larger than 5000).
- The `gettimeofday()` system call can be used to determine and record the time prior to your call to `connect()` and to determine the time after `connect()` has completed. The difference between these two times is an estimate of the RTT.
    - You will have to do conversion from the second and microsecond fields used by `gettimeofday()`
- I would strongly suggest that everyone begin by writing a client and getting its test cases to work. Then write the web server and test it with your client.
- In writing your code, make sure to check for an error return from your system calls or method invocations, and display an appropriate message. In C this means checking and handling error return codes from your system calls.
- If you need to kill a background process after you have started it, you can use the UNIX *kill* command. Use the UNIX *ps* command to find the process id of your server
- Make sure you perform error handling and close every socket that you use in your program.

- If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error.
- On UNIX systems, you can run the command "netstat" to see which port numbers are currently assigned.
- You need function `getaddrinfo()` to translate a URL (e.g., www.google.com) to an IP address to construct the socket address structure. See more details from Beej’s guide to network programming and socket programming at http://beej.us/guide/bgnet/html/