


Part 1: Socket Programming (25 points)

In this programming assignment, you will implement an HTTP client and server that run a simplified version of HTTP/1.1. Specifically, you will implement two HTTP commands: GET and PUT. This project can be completed only in C++ and Java.

HTTP Client

Your client should take the following command line arguments (in order): *server name*, *port* on which to contact the server, HTTP command (**GET** or **PUT**), and the path of the requested object on the server. In other words, assuming that your executable is named “*myclient*”, you should be able to run your program from the command line as follows:


`myclient hostname port command filename`

In response to a GET command, the client must:

1. connect to the server via a TCP connection
2. submit a valid HTTP/1.1 GET request to the server
3. read the server's response and display it

In response to a PUT command, the client must:

1. Connect to the server via a TCP connection
2. submit a valid HTTP/1.1 PUT request to the server
3. send the file to the server
4. wait for the server's reply
5. read the server's response and display it

HTTP Server

Your server should take a command line argument that specifies the port number that the server will use to listen for incoming connection requests. In other words, assuming that your executable is named “*myserver*”, you should be able to run your server from the command line as follows:


`myserver port`

Your server must:

1. Create a socket with the specified port number
2. Listen for incoming connections from clients
3. When a client connection is accepted, read the HTTP request
4. Construct a valid HTTP response:
 - a. When the server receives a GET request, it should either construct a “**200 OK**” message followed by the requested object or a “**404 Not Found**” message.
 - b. When the server receives a PUT request, it should save the file locally.
 - c. If the received file is successfully saved, the server should construct a “**200 OK File Created**” response.
5. Send the HTTP response over the TCP connection with the client
6. Close the client connection
7. Continue to “loop” to listen for incoming connection

Your **multithreaded server** will have an infinite loop to listen for connections. To shut down your server, you will have to interrupt it with a termination signal. Upon receiving the termination signal, your server must shut down gracefully, closing all sockets before exiting. You can design your program that best matches your programming styles.

Advice on how to tackle this project:

- Implement your client first.
- Test your client's GET command with an external HTTP server.
 - Before you test your client with your server, test it with a web server that you know works. In other words, use your client to get a file from some known external HTTP server. For example, from the command line: `myclient www.amazon.com 80 GET index.html`
- Now implement the server.
- Test the server with a browser (e.g., Firefox) as a HTTP client.
 - For example, your server is running at host `pc1.cs.uncc.edu` on port number 12000, and there is a file `index.html` in the current directory. In the URL box of the web browser, type: `pc1.cs.uncc.edu:12000/index.html`
The browser should fetch the file and display it.
- Now use your client to get a file from your HTTP server. For example, if your server is running on `pc1.cs.uncc.edu` port 12000, then enter at the command line: `myclient pc1.cs.uncc.edu 12000 GET index.html`
- Now use your client to put a file on your HTTP server.

Other tips:

Make sure to choose a server port number that is greater than 1023. Ports from 1-1023 are often used for well-known and widely used network services (e.g., port 80). Choosing a port number over 5000 is advisable. You can run all of the processes (all clients and the server) on the same machine. In this case, use "localhost" for the machine name. It is strongly recommended that you test your project in this way first, since you may sometimes run into issues with firewalls that are difficult to puzzle out. You can run do this from a Unix command line by running the server process in the background and then starting a client (if you want to open more than one client at a time, open another terminal and start).

Part 2: UDP vs. TCP - A Performance Evaluation (25 points)

Objective: To become familiar with TCP and UDP performance characteristics in practice by using socket programming.

Procedure:

1. Using TCP port number 8777, write a sender program and a receiver program. Each program runs on a different computer. First, the sender opens a connection to the receiver (in case of TCP). Next, the sender starts the testing process by recording the current time just before sending a test message, as shown in figure 1 below, with sequence number 1. Initially, the padding should be a 10 character string. Upon receiving this message, the receiver copies it into an array, then sends it back to the sender, which can now measure the End-to-End (ETE) delay by subtracting from current time the time recorded earlier and dividing the result by two. Next, the sender sends another test message, this time with sequence number 2, and again measures the ETE. Then another message with sequence number 3, and so on and so forth. Have the sender repeat this process 1000 times, recording the overall average and the maximum ETE.

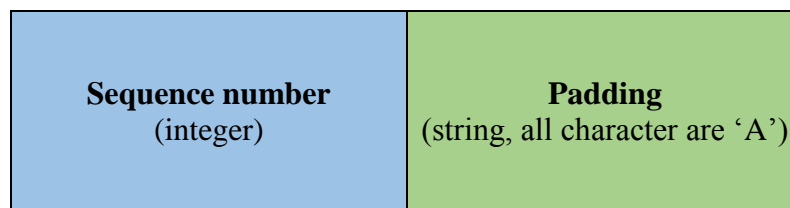


Figure 1: Format of the test message.

2. Repeat 1, 5 times, and record the average and maximum ETE for each of the 5 times.
3. Repeat 2 with a 200-character string.
4. Repeat 2 with a 1000-character string.
5. Repeat 2, 3, and 4, but this time with UDP instead of TCP.

Questions

- 1) Are the average ETE values, obtained from step 2, different for each of the 5 times that step 1 is repeated? Why?
- 2) Compare the average and maximum ETE for steps 2, 3, and 4 for TCP only. Explain the differences.
- 3) Compare the average and maximum ETE between TCP and UDP, for each of steps 2, 3, and 4. Explain the differences between TCP and UDP. What conclusion can you draw in terms of the performance of TCP and UDP?
- 4) If the tests were run over the Internet, where there is a chance of packet loss, how would the ETE values be different between UDP and TCP?

Submission Instructions and Notes

All work must be original. Duplicate or very similar assignment will receive zero. Also actions will be taken according to the rules of the department, school, and university.

1. You can work alone or as a team of TWO. Only one submission per team. Team members name cannot be changed after submission.
2. You can only use the following programming languages - **C/C++, Java, and Python.**
3. Submit a report covering the different aspects of project components.
4. Submit error free code so that TA can run. Make sure you include instructions so that TA can run easily.
5. Direct any question to TA, Ting Li (tli8@uncc.edu).