

CS536 Lab1: Socket Programming and Building Your Own Web Server Using Sockets

Due: Sep 19, 2023
Total points: 100 points

1 Goal

This is a programming lab where you will learn about client-server socket programming for both TCP and UDP. Afterwards, you will build a concurrent web server using TCP sockets in C.

2 Part A: Build Client-Server over Socket Programming (30 points)

- Set up server and client over Socket programming. The sample code for TCP socket programming are provided in the assignment (lab1.zip). Please compile the sample codes (using gcc or write a MAKEFILE) and generate two executive files **server** and **client**. Begin by opening two terminals. Start the server in one terminal first. Then, proceed to run the client in the other terminal. You will be prompted to enter a message, which will then be sent to the server. Upon receiving the message, the server will convert it to uppercase and send it back. The modified message will be displayed on the client's terminal.
- (15 points) You must modify the provided sample codes to enable the client and server to operate on separate machines and save them as **clientTCP.c**, **serverTCP.c**. You can specify the target server machine by using the IP address and port number as command-line arguments. Additionally, the client should be able to transmit an image to the server, with the image file path provided as the third input. The client will terminate the connection right after all the data is send. You should run the code using the following commands:

```
./serverTCP [port_server]
```

```
./clientTCP [ip_server_addr] [port_server] [image file path]
```

The image should be divided into multiple packets, each with a fixed size of **1KB** (1024 Bytes). The server will display basic socket information and transmission status on the terminal for every 1000 packets, using the following format (We also provide the sample output file in the lab1.zip, out_tcp.txt):

Server is listening on port [port_server]

Client connected from [client_ip]:[port_client]

Packet 1 received from client [client_ip]:[port_client]

Packet 1001 received from client [client_ip]:[port_client]

Packet 2001 received from client [client_ip]:[port_client]

...

Client [client_ip]:[port_client] disconnected

For example, if your image size is 2.4MB, then you are expected to have Packet 1, 1001 and 2001 printed in the terminal.

Please ensure that the print output of the server can be redirected to a text file. We will use an autograder to test with the output and will use (`./server [port_server] > out.txt`), and it is expected

that all output is immediately flushed to the file without terminating the server. Alternatively, you can implement a server timeout using `select`, where the server automatically closes after a specified time (for example, 30s), allowing the output to be successfully redirected to `out.txt`.

You can begin by running both of server and client at the `localhost`. Namely, use `localhost` as `[ip_server_addr]`. After it works, you need to finally make the client and server run on two different machines at HAAS G050. Below are the machine names:

```
amber01.cs.purdue.edu
amber02.cs.purdue.edu
...
amber30.cs.purdue.edu
```

- (15 points) Change the code to use UDP socket for the video transmission and locally store the transmitted video as `'received_video.mp4'`. Save the code as `clientUDP.c`, `serverUDP.c`. You should run the code using the following commands:

```
./serverUDP [port_server]
./clientUDP [ip_server_addr] [port_server] [video_file_path]
```

Note that UDP is an unreliable transmission protocol, and the transmission may result in out-of-order delivery. To check for out-of-order delivery and packet loss, we consider sending the image with a two-byte sequence number. Namely, the client will add a two-byte sequence number to each packet before sending it. Now the packet size is **1KB + 2** (1026 Bytes). The sequence number is started from 0. The server will display the sequence number in the following format (We also provide the sample output file in the `lab1.zip`, `out_udp.txt`):

```
Received sequence number [#]
Received sequence number [#]
Received sequence number [#]
```

```
...
```

```
Video received and saved as 'received_video.mp4'
```

Please conduct tests on both the `localhost` and two different machines at HAAS G050. Repeat these tests several times (at least 5 times) and provide a summary of your observations. In case of any out-of-order delivery, please verify the size of `'received_video.mp4'` and compare it with the original size. Note that larger data sizes can increase the likelihood of packet drop in network communications when using UDP protocols. You can also change your own video file to test its transmission performance. Remember you need to guarantee all the print output can be redirected to a text file.

3 Part B: Build a Concurrent Server Over TCP (30 points)

- You need to revise the TCP socket code to support multiple clients on the server using `Pthreads` and save it as `serverMul.c`. You can test the new server with at least two clients running on different machines or one machine with two different terminals. For these two clients, you should transmit distinct types of media. For instance, you could send one image and one video, or two images of different sizes.

You should run the code using the following commands:

```
./serverMul [port#]
./clientTCP [ip_server_addr] [port_server] [media_file_path_1]
./clientTCP [ip_server_addr] [port_server] [media_file_path_2]
```

Note that you need to add `-pthread` when you compile the `serverMul.c`.

We anticipate that the server can print messages from two clients interleaved with each other. Please note that you should run the second client before the first client terminates its connection. To ensure

that the first client program does not terminate quickly, you can use the `usleep` function to pause its execution speed. The server will display the sequence number in the following format (We also provide the sample output file in the lab1.zip, out_mul.txt):

```
Server is listening on port [port_server]
Client connected from [client_ip1]:[port1_client]
Packet 1 received from client [client_ip1]:[port1_client]
Packet 1001 received from client [client_ip1]:[port1_client]
Packet 2001 received from client [client_ip1]:[port1_client]
...
Client connected from [client_ip2]:[port2_client]
Packet 13001 received from client [client_ip1]:[port1_client]
Packet 1 received from client [client_ip2]:[port2_client]
Packet 1001 received from client [client_ip2]:[port2_client]
Packet 14001 received from client [client_ip1]:[port1_client]
Client [client_ip2]:[port2_client] disconnected
Packet 15001 received from client client_ip1]:[port1_client]
...
Client [client_ip1]:[port1_client] disconnected
```

Please be aware that this is merely a sample output. Your specific interleaved output may differ. Your only need to ensure that you follow the same semantics of the output and guarantee all the print output can be redirected to a text file.

4 Part C: Develop Your Web Server over HTTP/1.1(40 points)

Please read Chapter 2 of the textbook and the lecture slides carefully. Please play with the in-class demo to see how the existing websites (web servers) respond to various GET messages. All these will help you to understand how HTTP and HTTP/1.1 works. You need to pick a port number for your web server. Instead of using port 80 or 8080 or 6789 for the listening socket, you should pick your own to avoid conflicts. Please do not use any port number between 0 and 1024.

This simple web server needs to implement the following features:

- When the client sends GET to request for one .html which exists, it should respond “*200 OK*” and return this .html file.
- When the client sends GET to request for one .html which **doesn’t** exist, it should send the response “*404 Not found*”.
- When the client sends GET with the wrong format (e.g. URL String Syntax Error), it should send the response “*400 Bad Request*”.
- When the client sends GET with a different HTTP version, it should send the response “*505 HTTP Version Not Supported*”.

Before you run the server code, please create a folder named `www` in the folder containing your server program and place all the web objects in the `www` folder. Include the relative path the `www` folder in your server’s code and support the URL with and without `www`. That is, if you enter a URL `http://server/file:port\#` to get a webpage `http://server/file`, the server converts its URL into a local path as `./www/file`.

Several webpage files are provided in the zipped file to test with your web server. Of course, please feel free to create and use your own test webpage files and copy them into the `www` folder.

- `text.html`: a html file which contains text only.
- `picture.html`: a html file which contains text and a small picture (< 200KB).

- `bigpicture.html`: a html file which contains text and a big picture (> 1MB).
- `video.html`: a html file which contains text, a big picture, a small picture and a video.

We will use a web browser to test your server code and it is not necessary to revise the client code. But you can still use client code if you want. To test for HTTP responses 200, you can open a web browsers with the URL of `http://<machine name>:<port number>/<html file name>`. For example, you can try `http://localhost:8888/text.html` or `http://server-ip-address:8888/text.html` if your web server uses the port number of 8888 and has a file called `text.html` in its root folder (`www`) for all the web contents. Upon clicking “Enter”, the web browser will automatically generate the GET response and dispatch it to your server. Your expected outcome should be successful display of `text.html` in the web browser.

To test for HTTP responses (404, 400, 505), you can follow our in-class demo to send the GET messages through the command `nc -C -v [ip_server_addr] [port_server]`. Then enter `GET /non-exist.html HTTP/1.1`. You can get the response printed in your client terminal.

The server expects to print the following information on its terminal. Upon receiving a GET request from a client, the server will print out:

```
message-from-client: [client_ip]:[client_port]
[firstline-of-request]
```

Upon sending a response message to a client, the server will print out:

```
message-to-client: [client_ip]:[client_port]
[firstline-of-response]
```

where `[firstline-of-request]` or `[firstline-of-response]` are the first line of the request/response. We will give your sample output in the server side for all the above features:

```
GET /bigpicture.html HTTP/1.1\r\n (firstline-of-request); HTTP/1.1 200 OK\r\n (firstline-of-response).
GET /non-exist.html HTTP/1.1 \r\n (firstline-of-request); HTTP/1.1 404 Not found\r\n (firstline-of-response).
GETs /bigpicture.html HTTP/1.1\r\n (firstline-of-request); HTTP/1.1 400 Bad Request\r\n (firstline-of-response).
GET /bigpicture.html HTTP/2\r\n (firstline-of-request); HTTP/1.1 505 HTTP Version Not Supported\r\n (firstline-of-response).
```

Note `\r\n` is used to indicate the end of a line of text and the start of a new one. Please make sure all of the above print statement can be redirected to the text file for our grading.

Please keep in mind that you are still responsible for sending both the HTML file and its corresponding object to the client. However, we do not require you to print the entire file on your server side. The samples of HTTP responses are provided in `lab1.zip` as well. Please note that here we have simplified the HTTP request headers. They should contain more information such as `Date`, `Last-Modified`, `Content-Length`, `Content-Type`, and so on. You can add these fields if you want, before the body, but it is not required.

- Please save your server code as `serverWeb.c`. Please make your server code support multiple clients at the same time as you did in part B. Run your server. You should run the code using the following commands:
`./serverWeb [port#]`
- (20 points, 5 points per one test html files: `text.html`, `picture.html`, `bigpicture.html`, `video.html`). We provide one sample output for `video.html` in the `lab1.zip` (`out_web.txt`). Please follow the same semantics for other html files. Note that the number of GET requests and the content is differ for different html files. Please make sure the server should work with the existing browser (Chrome, Firefox, etc) to get the test webpages. Note that in Chrome, the video may not be interactive, but it will automatically initiate the download. In Firefox and Safari, the video link is clickable and will play the video.
- (15 points) Test with other HTTP responses (404, 400, and 505). In each test case, please copy the messages displayed at the client terminal into your lab report.

- (5 points) Run the above tests with two or more web browsers and screenshot the rendering of web browser and put it into your lab report.

5 Materials to turn in

You will submit your assignment at gradescope. Your submission should zip all the files into “lab1_PUID.zip”, including the following files:

1. All your source code files (`clientTCP.c`, `serverTCP.c`, `clientUDP.c`, `serverUDP.c`, `serverMul.c`, `serverWeb.c`).
 - Please add appropriate comments to your code and make it easy to understand.
2. Your project report called `lab1.*` (txt, pdf or word format). In the report,
 - Please include your name and student ID at the top of the first page.
 - Include a brief manual (readme) in the report to explain how to compile and run your source code (if the grader can’t compile and run your source code by reading your manual, the project is considered not to be finished).
 - Include the answers and results as specified above.
 - Describe what difficulties you faced and how you solved them, if applicable.
3. Please DO NOT INCLUDE any file under “www” folder OR ANY .html files IN YOUR SUBMISSION. You will lose at least 5 points without following the submission guideline.
4. Never submit .rar or other types of compressed file. You must submit .zip file. Otherwise, you will lose some points.

6 More Tips and Support

1. Please strictly follow the semantics of sample output, otherwise you will lose some points.
2. Do not change the file names, otherwise you will lose some points.
3. If connecting from off-campus, you will first need to connect to the Purdue VPN -<https://webvpn.purdue.edu/> and then access the Linux machines.
4. If you want to use any late day, please send an email to cs536-ta@cs.purdue.edu and tell us.

More questions about the assignment should be posted on [Campuswire](#) or asked during [PSOs](#).