COP4635 Sys & Net II **Project 2: HTTP Client** **Spring 2025**

HTTP Server and Client

OVERVIEW

The objective of this project is to demonstrate the interactions between an HTML server with an HTML client. The code for both the server and the client programs are to be written by the student (use your server program from project 1). The server and this client would interact through a command line interface and the responses would be seen on the command line.

THE PROGRAMS

The HTTP client-server system consists of two programs: a ***httpClient*** and a ***httpServer***. The ***httpServer*** program serves HTML files to the ***httpClient***. Both programs use **TCP** for communication.

IMPLEMENTATION

Program an HTTP client in C++ (you can use C style constructs inside, though, if required), which will request the user to enter the name of the http server followed by the name of a file on that server. The client program should send a valid HTTP/1.1 request to the server for the file. The client program should display all the responses received from the server. Make sure that your client program guides the user about the right usage (way to request for the file) before prompting the user and waiting on a client request. Also make sure you let the user request for more files in the same way through the client, and not exit immediately. If the server is still active and listening, it is supposed to respond accordingly. If the server has stopped working, this should also be indicated in the running client program. Make sure that you have provisions for handling both - normal and abnormal terminations of the client program, displaying appropriate codes and descriptive messages.

IMPLEMENTATION SUGGESTIONS

I suggest reading lecture notes and RFC for details on HTTP protocol and TCP networks discussed in class. You may also examine the reading material and the provided sample program on TCP programming. Read these pages thoroughly before starting this project. For your implementation you need to use the following system calls (not an exhaustive list – and you are not limited to just the below, or even using any of them for your implementation) to create a socket, bind it, send and receive messages and perform file service.

socket()	connect()	gethostname()	bind()
read()	write()	close()	listen()
fflush()	sprintf()	getaddrinfo()	freeaddrinfo()
fscanf()	sendto()	recvfrom()	sigaction()

Some videos have also been posted to get you started with the project. You don't have to follow the system calls suggested or discussed in the videos. You might come across several ways and means to implement the project. If it lets you achieve all the objectives listed in the project document, feel free to use any system calls as it is your project. What we have in the videos is just one possible way to get the project done.

WORKING IN TEAMS

Teams of up to two students can work on this project. If you wish to work in teams, you must make sure that you inform the instructor about your team ASAP, and at least a week before the deadline. Last-minute information about any teams to the instructor will not be entertained and such groups will not be allowed.

DELIVERABLES & EVALUATION

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of the instructor as published on eLearning under the Content area.

2. You should submit all the following files for this assignment:

   a. source code files for httpClient and httpServer, and supporting code files if any
   b. a single Makefile to compile the client and server programs,
   c. Relevant HTML and image files
   d. a protocol document
   e. a README file containing names of all project members, directions on compiling and running your files indicating usage and examples, and any other relevant details like an analysis if project wasn't completed
   f. Screen captures of your running program showing all windows involved, in two or three stages of communication.

The protocol document must describe the protocols used by the system to make it work. At the minimum the document must describe the message exchange between httpClient and httpServer. The README file should only be included if you submit a partial solution. In that case, the README file must describe the work you were able to complete.

Your program will be evaluated according to the steps shown below. Notice that the instructor/GA will not fix your syntax errors. However, they will make grading quick!

1. Program compilation with Makefile. The options –g and –Wall must be enabled in the Makefile. See the sample Makefile that I uploaded in eLearning.
   - If errors occur during compilation, the instructor/GA will not fix your code to get it to compile. The project will be given zero points.
   - If warnings occur during compilation, there will be a deduction. The instructor/GA will test your code, though.
2. Program documentation and code structure.
   - The source code must be properly documented, and the code must be structured to enhance readability of the code.

- Each source code file must include a header that describes the purpose of the source code file, the name of the programmer(s), the date when the code was written, and the course for which the code was developed.

3. Perform several test-runs with input of the grader's own choosing. At a minimum, the test runs address the following questions.
   - Do the httpServer and httpClient program compile properly and run, and is the communication between the two processes evident?
   - Is there ample and evident feedback on what is going on in the httpClient and httpServer programs while they are running, and when they exit?
   - Does the httpClient handle all responses from the httpServer appropriately and display the content received from the server in the terminal?

Keep in mind that documentation of source code is an essential part of computer programming. The better your code is documented, the better it can be maintained and reused. If you do not include comments in your source code, points will be deducted. You should refactor your code to make it more manageable and to avoid memory leaks. Points will be deducted if you don't refactor your code or if we encounter memory leaks in your program during testing.

DUE DATE

The project is due as indicated by the Dropbox for Project 2 in eLearning. Upload your complete solution to the dropbox and the shared drive. I will not accept submissions emailed to me or the GA. Upload ahead of time, as last-minute uploads may fail. You can use the shared VM for creating and testing your code. Please mention in the comments at the top of your code if it tested perfectly with the shared VM and/or on the UWF-CS ssh server. Else, if you used WSL, please do that in the code as well as in the README file.

TESTING

Your solution needs to compile and run on the CS department's SSH server. I/the GA will compile and test your programs on the SSH server and a Linux VM. Therefore, to receive full credit for your work it is highly recommended that you test & evaluate your solution to make sure that we will be able to run your programs and test them successfully. For security reasons, most of the ports on the servers have been blocked from communication. Ports that are open for communication between the public servers' range in values from 60,000 to 60,099.

Some students have found it easier to develop utilizing WSL for the project rather than the ssh server and/or the provided VM image. If your program works in WSL please mention this in your README file and that is how we will grade them.

GRADING

This project is worth 100 points total. I will be updating this file with the rubric and will email you of this update. Please remember that there will be deductions if your code has memory leaks, crashes, has endless loops or is otherwise poorly documented or organized and there will be zero points for code that does not compile.

BONUS

Your client should be able to send text to the server and have the server acknowledge that this text was received. The server should be able to clearly distinguish between the client asking for a file and the client sending text to the server. You may implement this distinction through whatever method you like. If completed, you will be provided with 10 bonus points.

This bonus may be completed for either Project 1, Project 2, or both.

SAMPLE OUTPUTS

Following are some sample outputs from bare-bones HTTP server and client code in C. This code isn't as elaborate, efficient or informative as your project code should be. I used port address 46000 locally, for the test. The three screenshots show the server and client working together, then the client is terminated, and a standard browser is tested for requesting and displaying the valid and existent pages, followed by testing a non-existent file called fake.html, and finally testing the browser client with the server shut down.