Simple HTTP Client and Server

Overview

In this project, you will implement a simple HTTP client and a simple HTTP server using socket API. This project should be completed in C. The client and server programs must compile and run on pyrite.cs.iastate.edu to receive credits.

Part I: The HTTP Client

I.1. Program Description:

You will write a simple HTTP client that does the following: (1) construct an HTTP 1.0 request message based on the user's command line input, (2) print the constructed request message on the standard output, (3) send the constructed request message to the HTTP server specified in the user's command line input, and (4) receive and write the response message to an output file.

I.2. What You Need to Do:

```
Command line usage: client [-h] URL
```

The executable file is named *client*. The input and output descriptions of the client program are as follows.

Client program's input: The client program takes one option *-h* and one required argument *URL*. The option identifies the operation to be used in the request message. The argument *URL* specifies the URL of the object that the client is requesting from the server. The URL format is `http://hostname:port/pathname`. If `port` is not specified, the default port number 80 will be assumed.

Client program's output: The client program writes the constructed request message to the standard output and writes the response message received from the HTTP server to an output file named *response.txt*.

The client program should do the following:

1. Parse the command line argument *URL* into *hostname*, *port*, and *identifier*. For example, if the URL is `http://www.someschool.edu:1234/index.html`, then *hostname* is `www.someschool.edu`, *port* is `1234`, and *identifier* is `/index.html`. If the URL does not include `pathname`, then the default identifier / will be used. For example, if the URL is `http://www.someschool.edu`, then *identifier* is `/`.
2. Open a socket and connect it to the HTTP server running on *hostname*.

3. Construct an HTTP1.0 request message based on the *option* command line argument. The request message should be in the form of a string, in which all lines of the message are concatenated.
    a. If the option *-h* is NOT specified, construct an HTTP1.0 request message with the *GET* operation. For example, the command line `client http://www.someschool.edu:1234/index.html` will result in the HTTP1.0 request message `GET /index.html HTTP/1.0\r\n\r\n`.
    b. If the option *-h* is specified, construct an HTTP1.0 request message with the *HEAD* operation. For example, the command line `client -h http://www.someschool.edu:1234/index.html` will result in the HTTP1.0 request message `HEAD /index.html HTTP/1.0\r\n\r\n`.
4. Write the constructed request message to the standard output. Every line, including the last blank line, of the constructed request message should be printed onto the standard output.
5. Send the constructed request message, receive the response message, and write the response message to an output file named *response.txt*.

I.3 Compiling the Client Program and Creating the Executable File

Use `gcc -o client client.c` to compile the client program and create the executable file. The executable file is named *client*.

I.4 Testing Your Client Program

You can use your client to connect to any Web server using port number 80. You should try different URLs and options to make sure that the client works correctly.

You can put the message body of the response message you received from the Web server into a .html file and use any Web browser to view the .html file. Compare the result with when you directly specify the URL in your Web browser.

I.5 Grading

Total Score: 100 points

1. Program Correctness 80 points
2. Program Robustness 10 points
    - Properly handle errors in socket operations.
    - Properly handle other errors. For example, your program should not continue if the number of arguments specified in the command line is not correct or non-recognized options are specified.
3. Documentation 10 points
    - You should add plenty of comments to describe the purpose of your code.
    - Comments should be added at the top of your program file, above every function, and in line where it is not obvious what you are trying to do.

Part II: The HTTP Server

II.1. Program Description:

You will write a simple HTTP server that handles one service request at a time. Your HTTP server receives and responds to the request sent from the simple HTTP client you write in part I. The server program (1) waits for a connection request. If there is a connection request, the server program (2) accepts the connection request, (3) writes the IP address and the port number of the connecting client onto the standard output, (4) reads the HTTP request message, and (5) responds to the HTTP request message. After finish serving this connecting client, the server program (6) goes back and waits for a future connection request.

II.2. What You Need to Do:

The executable file is named *server*. The input and output descriptions of the server program are as follows.

Server program's input: The server program does not take any input from the user.

Server program's output: The server program prints the IP address and the port number of the connecting client on the standard output.

The server program should do the following.

1. Open a socket, bind the socket to a local protocol address, and wait for a connection request from a client.
2. If there is an incoming connection request, accept the connection, print the IP address and port number of the client to standard output.
3. Read the HTTP request message and construct the HTTP response message based on the following two cases.
   Case 1. The client submits a `GET` request message.
   If the requested object is not found, the response message contains the status line `HTTP/1.0 404 Not Found` and an empty body. If the requested object is found, the response message contains the status line `HTTP/1.1 200 OK` and the `Content-Length` header line, and the requested object is included in the message body. Please be sure to insert an empty line between the header line and the response message body.
   Case 2. The client submits a `HEAD` request message.
   If the requested object is not found, the response message contains the status line `HTTP/1.0 404 Not Found` and an empty body. If the requested object is found, the response message contains the status line `HTTP/1.0 200 OK` and the `Last-Modified` header line.
4. Send the HTTP response message back to the client and go back to 2.

Hints:

1. You need to choose a port number for the server to listen to. The port number must be between 1024 and 65535. You may use the last four digits of your university ID as the port number to avoid any contention for the same port number on pyrite.
2. You may find the following functions useful:

`int stat(const char *filename, struct stat *buf) -` Obtain information about the specified file.

`char *ctime(const time_t *clock) -`convert the time pointed to by clock to local time in the form of a string.

`const char *inet_ntop(int family, const void *addrptr, char *strptr, socklen_t len) -` convert an IP address from binary form to text form.

II.3. Compiling the Server Program and Creating the Executable File

Use `gcc -o server server.c` to compile the server program and create the executable file. The executable file is named *server*.

II.4. Testing Your Server Program

1. Put an HTML file named testfile.html in the directory where your server program is located.
2. Start your HTTP server by invoking your server program.
   `$ server <enter>`
3. Start your HTTP client by invoking your client program. Assume the server program has been invoked on `pyrite-n1` and server port number is 1234.
   `$ client http://pyrite-n1.cs.iastate.edu:1234/testfile.html`
4. Verify the file your client program has received from your HTTP server.

Note: pyrite is a cluster with 2 nodes. You can find out which node you are using by looking at the command line prompt. If your prompt is `pyrite-n2:~>`, then your hostname is `pyrite-n2.cs.iastate.edu`.

II.5. Grading

Total Score: 100 points

1. Program Correctness 80 points
2. Program Robustness 10 points
   - Properly handling errors in socket operations.
3. Documentation 10 points
   - You should add plenty of comments to describe the purpose of your code.
   - Comments should be added at the top of your program file, above every function, and in line where it is not obvious what you are trying to do.

Submitting Your Project

You must submit your project on Canvas.

Put all your C source files in a folder. Then use the `zip` command to create a .zip file. For example, if your Net-ID is `ksmith` and `project1` is the name of the folder that contains all your source files, then you will type `zip -r ksmith project1` to create a file named `ksmith.zip` and then submit this file.