Com S 352 Spring 2018
Project 2
Communication between HTTP Client and Multi-Threading Server using Socket Programming
100 points
Due: Friday, April 13, 11:59pm

Goals:

The purpose of this project is to:

1) Give students hand-on experience with
   Socket Programming
   Multi-Threading
and
2) Help students better understand application-layer protocols by implementing a well-known application-layer protocol—HTTP.

In this programming project you will write HTTP client and server programs using sockets. You are not required to implement the full HTTP specification, but only a very limited subset of it.

Background:

What is HTTP?
HTTP stands for Hyper Text Transfer Protocol and is used for communication among web clients and servers. HTTP has a simple stateless client/server paradigm.

The web client initiates a conversation by opening a TCP connection to the server.
Once a TCP connection is set up, the client sends an HTTP request to server.
Upon receiving the HTTP request from the client, the server sends an HTTP response back to the client.

An HTTP request consists of two parts:

A header and A body.

In this project, HTTP request from a client doesn't contain a body. The first line of any request header should be:

Method/URL/Version.

An example HTTP1.1 request is:

GET /index.shtml HTTP/1.1
Host: www.cs.iastate.edu

Note that “Host:” header line must be included in an HTTP1.1 request. The request header and body are separated by two sets of carriage return and line feed. Since we do not need the body, the end of a header marks the end of a request.

Using a C char string, the example request above should be:

"GET /index.shtml HTTP/1.1\r\nHost: www.cs.iastate.edu\r\n\r\n".

A response from a server also has two parts:
A header and a body separated by two sets of carriage return and line feed.

The header consists of a status line and a set of header lines.

The body contains the requested object.

In this project, the response header contains only the status line, which has three fields:

HTTP-Version Status-Code Status-Phrase.

What is a URL?

Uniform Resource Locators (URLs) are formatted strings that identify resources in the web: documents, images, downloadable files, electronic mailboxes, etc. It generally has the format:

Protocol://Hostname[:port]/Filepath

In this project, when a port is not specified, the default HTTP port number of 80 is used. For example, a file called "foobar.html" on HTTP server "www.yoyodyne.com" in directory "/pub/files" corresponds to this URL:

http://www.yoyodyne.com/pub/files/foobar.html.

In the above URL, the default HTTP server port number 80 is used.

If a HTTP server resides on a different port, say, port 1234, then the URL becomes:

http://www.yoyodyne.com:1234/pub/files/foobar.html.

What You Need to Do:

You will write two programs:

1. HTTP client
2. HTTP server

Part I: The HTTP Client

Command line usage: (here "client" is the executable file)

client [–h] [–d <time-interval>] <URL>

The client takes **two options** "-h" and "-d" and a **required argument** <URL>.

1. <URL> specifies the URL of the object that the client is requesting from server. The format is:
   http://hostname[:port]/filepath.

2. Option "-h" specifies that the client only wants the response header to be sent back from server, i.e., the server should not include the requested object in the response. You should use HEAD method in your HTTP request when "-h" is specified in the command line.
3. Option "-d" along with its argument <time-interval> specify that the client wants the object only if it was modified in the past <time-interval>.
   <time-interval> is in the format "day:hour:minute".
   For example, if "–d 1:2:15" is included in the command line, then the client wants to get the object only if it was modified in the past 1 day, 2 hours and 15 minutes.
   When –d option is specified, the client should include a "If-Modified-Since:" header line in the HTTP request and the value of this header is computed using the current time and the <time-interval> argument.
   For example, suppose the current date and time is 3/5/2018 15:20:00, and the <time-interval> argument is "1:2:15", then the client wants the object only if the object has been modified since 3/4/2018 13:05:00.

In Client program, you need to:

- Parse the <URL> given in the command line (A URL should be parsed into hostname, port number and resource identifier. For example, if the URL is http://www.cs.iastate.edu/index.html, then hostname is www.cs.iastate.edu, port number is 80 (the default value), and the resource identifier is /index.html.),
- Connect to the server,
- Construct an HTTP request based on the options specified in the command line,
- Send the HTTP request to server,
- Receive an HTTP response and save the response to a local file named "response".

Part II: The Multi-Threaded Server

You will write a HTTP server (the server will be multi-threaded) that is capable of processing the service requests sent by the HTTP client you have written in Part I. On Server side, use multi-threading, such that Server can handle requests from more than one client, using one thread to handle each client's request. A multi-threaded server creates a new thread for each communication it accepts from a client. A thread is a sequence of instructions that run independently of the program and of any other threads.

In server program, you need to:

- Open a TCP Socket,
- Bind the server address and Listen for incoming connection requests.
- Upon receiving a connection request, Accept the connection request,
- Create a Thread to handle this client
- Read the request from the socket, display the request on screen and
- Send back an appropriate HTTP response to the client.

The server's response depends on the type of request sent by the client:

(1) The client sends a regular GET message.

If the requested file is not found in the current directory (i.e. the directory where the server is running), the response message contains the status line "HTTP/1.1 404 Not Found" and an empty body.

If the requested file is found in the current directory, the response message contains the status line "HTTP/1.1 200 OK" and the requested file is included in the message body.

(2) The client sends a conditional GET message:

If the requested file has been modified since the specified date, then the response message contains the status line "HTTP/1.1 200 OK" and the requested file is included in the message body.

Otherwise, the response message contains the status line "HTTP/1.1 304 Not Modified" and an empty body. You can assume the requested file is located in the current directory.

(3) The client's request uses HEAD method:

If the requested file is not found in the current directory, the response message contains the status line "HTTP/1.1 404 Not Found" and an empty body.

If the requested file is found in the current directory, the response message contains the status line "HTTP/1.1 200 OK" and an empty body.

Hints:

1. The value for "If-Modified-Since" header is a string like:
"Mon Mar 05 2018 13:12:10".

The following code can be used to compute the value for "If-Modified-Since" header based on the <time-interval> argument of the "-d" option.

```
int day, hour, min; // get from <time-interval> argument
char s_time[30];  //value to be used in "If-Modified-Since" header
time_t n_time;
n_time=time(0);
n_time=n_time-(day*24*3600+hour*3600+min*60);
strcpy(s_time, ctime(&n_time));
```

2. When the client reads an HTTP response from the socket to a buffer, you should not assume that the buffer is big enough to hold the entire response. Therefore, a loop should be used to read from the socket repeatedly until the entire response message has been read and saved to a local file named "response".

3. You can use the **stat()** function to get the last modification time of a file.

Test the client and server:

Your client program should be able to connect to any HTTP server. You should try different URLs and options to make sure that the client works correctly.

You should use the http client you write in Part I to test your server. You should start the server before you start the client.

Submission

- You should use C to develop the code.
- You need to turn in electronically by submitting a zip file named: Firstname_Lastname_Project2.zip.
- Source code must include proper documentation to receive full credit (you will lose 10% of your score, if the code is not documented).
- All projects require the use of a **make file** or a certain script file (accompanying with a **readme file** to specify how to use the script file to compile), such that the grader will be able to compile/build your executable by simply typing "make" or some simple command that you specify in your readme file.
- Source code must compile and run correctly on the department machine "pyrite", which will be used by the TA for grading.
- You are responsible for thoroughly testing and debugging your code. The TA may try to break your code by subjecting it to test cases.
- You can have multiple submissions, but the TA will grade only the last one.

Start as early as possible!