CSc 361: Computer Communications and Networks (Summer 2016)

Assignment 1: Web Client and Web Server

Spec Out: May 4, 2016
Final Due: 3:30 pm, May 27, 2016

1 Goal

The project is to build a simple web client and a simple web server. The purpose of this project is two fold:

- to provide students with hands-on experience with socket programming,
- to help students understand application-layer protocols by implementing partial service primitives of HTTP/1.0.

2 Background

2.1 HTTP

HTTP stands for Hyper Text Transfer Protocol and is used for communication among web servers. In this project, we will use HTTP version 1.0. HTTP has a simple stateless client/server paradigm. The web client initiates a conversation by opening a connection to the server. Once a connection is set up, the client sends up an HTTP request. The server sends an HTTP response back to the client and closes the connection. An HTTP request consists of two parts: a header and a body. Whether a body follows a header or not is specified in the header.

In the project, we consider only the *single-line header of HTTP request*. The first line of any request header should be:

- the method field: The method field can take on several different values, including GET, POST, HEAD, and so on. In this assignment, however, **you are only required to implement GET.**
- the URL field: It is the field to identify a network resource, e.g., "http://www.csc.uvic.ca/index.html".
- the HTTP version field: This field is "HTTP/1.0".

An example is: "*GET http://www.csc.uvic.ca/index.php HTTP/1.0*". The request header and body are separated by two sets of a carriage return and a linefeed. 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 http://www.csc.uvic.ca/index.php HTTP/1.0\r\n\r\n*".

The response from a server also has two parts: a header and a body. The first line of a header should be:

- the HTTP version field,
- the status code field,
- the phrase field.

In the project, you are required to implement two status codes: 200 and 404. The status code 200 means that the request succeeded and the information is returned in the response. The status code 404 means that the requested document does not exist on this server. Two example response messages are: "*HTTP/1.0 404 Not Found\r\n\r\n*" and "*HTTP/1.0 200 OK\r\n\r\n data data data ...*"

2.2 URI

URI stands for Uniform Resource Identifier and is also known as the combination of Uniform Resource Locators (URL) and Uniform Resource Names (URN). It is a formatted string which identifies a network resource. It generally has the format: *protocol://host[:port]/filepath*. When a port is not specified, the default HTTP port number of 80 is used.

Note: In the final lab test, you should set up a URI such that the traffic goes through the lab router, which loops the traffic back to the lab desktop.

3 Project Description

In this project, you are required to write two programs: one for the client, named as *SimpClient*, and the other for the server, named as *SimpServer*. The goal is to learn socket programming and simple HTTP primitives.

3.1 The Client

The client program first accepts URI from stdin and parses it. Then it connects to a server, sends an HTTP request, and receives an HTTP response. You should write a routine that gets the hostname and port of the server as input, opens a socket, and connects to it. The return value of the routine is the socket fd (file descriptor) that will be used for sending/receiving HTTP requests/responses. You should also implement a routine that prints out the response from the server, marking the header and the body. When you finish the client, you can try to connect to any HTTP server before you write the server program. For instance, type "http://webhome.cs.uvic.ca/∼wkui/index.html" as the input to the client program and see what response you get.

As an example output, after you run your code with

```
% SimpClient http://www.uvic.ca/index.html
```

Your SimpClient should output:

```
---Request begin---
GET http://www.uvic.ca/index.html HTTP/1.0
Host: www.uvic.ca
```

```
Connection: Keep-Alive

---Request end---
HTTP request sent, awaiting response...

---Response header ---
HTTP/1.1 200 OK
Date: Wed, 06 Jan 2016 00:39:39 GMT
Server: Apache/2.2.31 (Unix) mod_jk/1.2.40

--- Response body ---
Body Body .... (the actual content)
```

Your output should follow the above example. In particular, in the HTTP respond, the information regarding "HTTP version," "status code," "phrase field," "date," "server," "response body", if any, should be printed out.

3.2 The Server

Now you have a client program that connects to a server and can retrieve a simple HTML file. The server program you will write is to accept the GET request from the client and responds with the requested file. If the requested file does not exist in the given directory, the server should respond with "HTTP/1.0 404 Not Found." For simplicity, the server responds to any other HTTP methods (PUT, POST, HEAD, etc.) with "HTTP/1.0 501 Not Implemented." In the server program, you need to open a socket, bind the server port number to it, and listen to any incoming connection. If there is a connection, then accept it, receive the request, and send back the response.

To start SimpServer, you should run

```
% SimpServer portNumber nameOfdirectory
```

By default, portNumber is 80. Of course, you can use other port number. Since SimpServer will not print out anything, the best way to test your SimpServer is to make your SimpClient work correctly first.

3.3 Other Notes

1. To make your life easy, simple skeleton code will be provided in connex → resource
2. Two useful functions, readn() and writen(), are in a separate file called util.c and will be used by both the client and the server.
3. When a process is aborted, sockets that process opened may hang around even after the process is gone, making future bind() to the port number fail. If a client uses a dedicated port number, this problem will block a client from running if the previous client has been aborted. To avoid this problem, the client should not pick its own port number. Instead, it uses whatever the system allocates to it when bind() is called. For a server, this tactic does not work, because a server uses specific well-known port.

4. If you use a character string IP address instead of a numeric IP addresses (e.g., 192.116.11.2) in your URI, you need to use gethostbyname() to convert the char string IP address to

   `sockaddr_in.sin_addr,`

   with the following code:

```
#include "netdb.h"
struct sockaddr_in server_addr;
struct hostent *server_ent;

server_ent= gethostbyname(hostname);
memcpy(&server_addr.sin_addr, server_ent->h_addr, server_ent->h_length);
```

5. You may reuse any c source code on TCP server/client (e.g., the typical one is TCP echo server/client). Nevertheless, you are required to put a reference in the readme file to the site from which you obtain the sample code.

6. Regarding other printout: Anything not specified in Assignment 1 is optional. For example, you can decide whether or not to print out the IP address, port number, and so on. When TAs test your code, if your code works fine without any problem, you are fine even if you do not print out anything not required in Assignment 1. Nevertheless, if your code does not work, TAs will not spend time to figure out what is wrong and you get a zero mark on the required function (Refer to the table in Section 5 of Assignment 1). In this case, if your code includes some printout to show intermediate results, TAs will have an idea on how far you have achieved and give you some partial mark based on their own judgement.

7. Regarding readme file. Readme file is important. Without it TAs will not know how to compile your code and how to run your code. It would waste our time to deal with your complaint if TAs cannot run your code and give you a zero.

8. For more information on HTTP, HTML, URI, etc., please refer to http://www.w3.org. It is the home page of W3 Consortium and you will find many useful links to subjects related to the World Wide Web.

4 Schedule

In order to help you finish this programming assignment successfully, the schedule of this assignment has been synchronized with both the lectures and the lab tutorials. Before the final deadline, there are three lab sessions arranged during the course of this assignment. A schedule is listed as follows:

Session	Tutorial	Milestones
Lab 1	lab environment, P1 spec go-through, design hints, system calls	design and code skeleton
Lab 2	socket programming and testing, DNS	alpha code done
Lab 3	socket programming and last-minute help	beta code done and demo

5 Deliveries and Marking Scheme

For your final submission of each assignment you are required to submit your source code to connex. You should include a readme file to tell TA how to compile and run your code. At the last lab session that you attend, you need to demo your assignment to TAs. Nevertheless, before the final due date, you can still make changes on your code and submit a *change.txt* file to connex to describe the changes after your demo.

The marking scheme is as follows:

Components	Weight
Make file	5
Error handling in SimpClient	10
Correct output in SimpClient	30
status code 200 and correct content in SimpServer	20
status code 404 in SimpServer	15
status code 501 in SimpServer	10
Code style	5
Readme.txt and change.txt(if any)	5
Total Weight	100

6 Plagiarism

This assignment is to be done individually. You are encouraged to discuss the design of your solution with your classmates, but each person must implement their own assignment.

7 Extra Info: Code Quality

We cannot specify completely the coding style that we would like to see but it includes the following:

1. Proper decomposition of a program into subroutines (and multiple source code files when necessary)—A 500 line program as a single routine won't suffice.
2. Comment—judiciously, but not profusely. Comments also serve to help a marker, in addition to yourself. To further elaborate:
   (a) Your favorite quote from Star Wars or Douglas Adams' Hitch-hiker's Guide to the Galaxy does not count as comments. In fact, they simply count as anti-comments, and will result in a loss of marks.
   (b) Comment your code in English. It is the official language of this university.
3. Proper variable names—`leia` is not a good variable name, it never was and never will be.
4. Small number of global variables, if any. Most programs need a very small number of global variables, if any. (If you have a global variable named `temp`, think again.)
5. **The return values from all system calls and function calls listed in the assignment specification should be checked and all values should be dealt with appropriately.**

The End
