Practical Assignment #4

Introduction to Computer Networks

Description

Through Practical Assignment #1 to #4, you will build by the end of the semester a simple Unix-based Web server. That server will be developed in C on a Unix-based OS. This simple Web server will be capable of serving one request at a time. To simplify the programming task and to proceed incrementally, we will lead you through the simple Web server implementation in four stages. At the first stage, you have been asked to get on a Unix-based system and practice a number of basic commands to get around the Unix system.

From the second stage and on, you will be implementing towards a simplified HTTP/1.0 (as defined in RFC 1945) Web server. In that, you will implement a Web server, which listens for a Web client at a time. The entire Web server implementation will be divided into three parts: 1) the echoer, 2) the parser, and 3) the responder. One will be implemented based on another.

At the final stage, you will extend your server program from the parser to the simple Web server, which sends back the requested file as an HTTP Response message. We expect that your Web server will be able to send back the test web page and display on **Mozilla Firefox** (one of the Web browsers).

Web Server Functionality

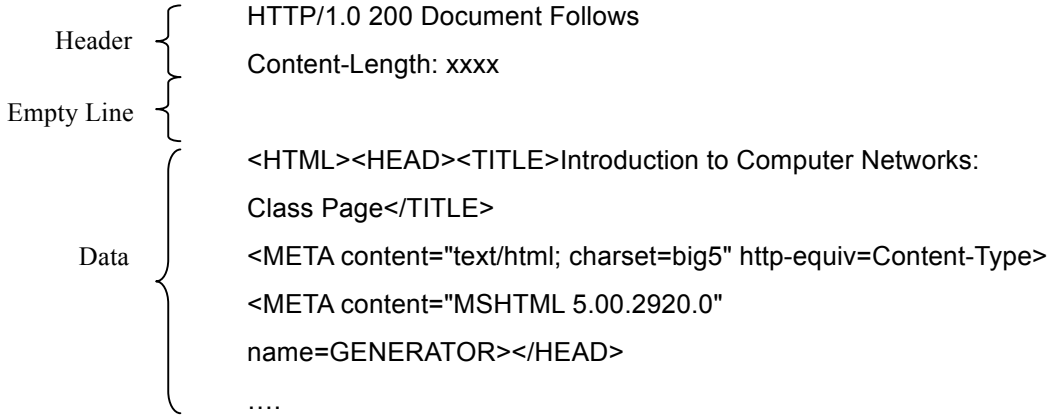
You will be building on top of your server.c from stage 3 of the assignment. The convention is still that you would edit the server.c in a text editor (for example: emacs) and compile it with the Unix C compiler.

In your server.c, instead of just parsing the GET messages from the client, you need to send back the page requested by the GET message. More specifically, your server.c will need to do at least the following:

1. Check whether the message is a GET message
2. If not, send back ‘Bad request’ string.
3. If so, find out the </path/name> of the file the client is requesting.
4. Check if the file, </path/name>, exists
5. If not, send back ‘File not existed’ string.
6. If so, send back an HTTP Response with the file

HTTP Response Format

The reply your server sends back should follow the HTTP Response message format. For example:



Testing

For your reference, a sample parser program is available from the course website: http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/server-PA4.o. server-PA4.o works for MacOS, the operating system used by workstation 140.112.42.161. server-PA4.o listens from port 3499. A simple html file, server-test.html, is also available for the testing purpose. You may find the file from: http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/server-test.html. Place server-PA4.o and server-test.html in the same directory on 140.112.42.161. Start server-PA4.o. Test the parser by http'ing to it from **Mozilla Firefox** (one of the Web browsers):

http://140.112.42.161:3499/server-test.html

You should see the following from the Web browser:

Introduction to Computer Networks

Fall 2004

Fall 2003

Class#: 901 31110

Course in English

Latest News | DiscussionBoard | Admin | Syllabus | Recordings | Slides | Written Assignments | Practical Assignments | Exams

……

Hints on generating the HTTP Response Header

Use the sprintf() function call well to send the lines in the HTTP Response header. For example, after obtaining the name of the file to be requested (filename, a string), you may use the following lines of code to send the 'Content-Length: xxxx' part of the header.

```
requested_file = fopen(filename, "rb");         // open file
fseek(requested_file, 0, SEEK_END);             // move to the end of the file
end=ftell(requested_file);                      // get the position of the end of file
stringlen=sprintf (tmpstring, "Content-Length: %d\n", end);
send(new_fd, tmpstring, stringlen, 0);
```

For details about using sprintf() and fread(), please refer to the manual of stdio.h library function calls in http://www.cplusplus.com/ref/cstdio/.

Hints on generating the HTTP Response Data

Use the fread() function call well to send the content in requested file. For example, after opening the file to be requested (requested_file, the file descriptor) and allocating a 1024-character temporary string (buffer), you may use the following lines to read from the file and send 1024 characters at a time.

```
stringlen=fread (buffer, 1, 1024, requested_file);
send(new_fd, buffer, stringlen, 0);
```

For details about using fread(), please refer to the manual of stdio.h library function calls in http://www.cplusplus.com/ref/cstdio/.

Submission

You will rename your server.c following the assignment naming convention to, for example, p4-2-1223-1843.c. Please note that the code for Practical Assignment #4 will be p4. Then, upload the file to the sftp server by the due date and time.

Port Assignment

To avoid conflicts, each team will be using a specified port number as shown as in the port assignment file provided.