Practical Assignment #3

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 third stage, you will need to extend your server program from the echoer to the parser, which interprets the HTTP Request (GET) messages from a Web browser and check if the file requested exists. If the file exists, find out the size of the file. We expect that your parser will be able to send back short messages about whether it finds the file and the size of the file when it receives requests from **Mozilla Firefox** (one of the Web browsers).

Parser Functionality

You will be building on top of your server.c from stage 2 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 simply echoing the messages sent from a client, the server would parse the message from the client as an HTTP GET request and send back some information about the file 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
    i. ‘File found’ string.
    ii. ‘File size: <size in bytes>‘ string

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-PA3.o. server-PA3.o works for MacOS, the operating system used by the workstation 140.112.42.161. server-PA3.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-PA3.o and server-test.html in the same directory on 140.112.42.161. Start server-PA3.o. Test the parser by http’ing to it from **Mozilla Firefox** (one of the popular Web browsers):

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

You should see the following from the Web browser:

File found
File size: 2183

Hints on Parsing

Having received the HTTP Request message (msg, a char array), you will need to extract one word (one token) at a time to figure out 1) whether the message is a GET request and 2) which file is being requested. Much of the trick to play here is about string manipulation. Feel free to use function calls such as ‘strtok’ to grab words (tokens) out of a long message one by one:

```
pch1 = strtok (msg, " ");       // gets the first word, words separated by space
pch2 = strtok (NULL, " ");      // gets the subsequent word from msg
```

strcmp also comes in handy when checking whether the extracted word equals the "GET" string.

```
strcmp(pch1,"GET");          // checks whether the first word is GET
strcmp(pch2,"/");            // "/" for filename meant to return index.html
```

Please refer to http://www.cplusplus.com/ref/cstring/ for the details of string.h library function calls.

Hints on Finding out File Size

After obtaining the name of the file to be requested (filename, a string), you will need to 1) figure out what is the file size and 2) send a short message about the file size back to the client. This part of implementation concerns more the file manipulation. One could use function calls such as 'fseek' and 'ftell' as follows to find out the file size:

```
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, "file size: %d<br>\n", end);
send(new_fd, tmpstring, stringlen, 0);
```

An example about computing the file size is also available from:
http://www.cplusplus.com/ref/cstdio/ for the details of stdio.h library function calls.

Submission

You will rename your server.c following the assignment naming convention to, for example, p3-2-1223-1843.c. Please note that the code for Practical Assignment #3 will be p3. 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 in the port assignment file provided.