Programming Assignment 1 (15 pts)

Due: March 1, 2010 (submitted by class time)

A simple web page transfer system

This assignment introduces client-server programming using the TCP/IP protocols. In this assignment, you will write a simple web page client and a web page server in C or C++ using Unix socket commands. The client and web page server must execute on different CCC machines and communicate with each other using TCP. The server has several web pages (.html files, just HTML, no images is fine), in one of its directories that a client can retrieve.

The Client

The client should be designed to accept user input as single line commands from standard input (keyboard). The user interacts with the client by typing in the commands that are specified below (login, add, get, delete, etc). The client program accepts commands from the user, sends these commands to the web page server, receives responses from the web page server and prints them out to the screen.

The client can connect to a server and then interact with it using the following commands:

1) **login:** establish a TCP connection to the server;
2) **list:** obtain a list of the web pages that the server currently has;
3) **get:** get one web page at a time;
4) **add:** send the server a new web page to add to its list;
5) **delete:** Remove a web page from the server's list of pages.
6) **quit:** client program exits

The client should be written to run on any arbitrary CCC Unix/Linux machine. To start the client, the user types:

Webclient WebServerHost

where

WebServerHost is the name of the machine where your web page server is running (e.g. ccc2.wpi.edu)

The client communicates with the web page server assuming knowledge of a unique "well-known" port that you have chosen. The client accepts and relays the following six commands to the server:

login

Upon receiving the login command, the client establishes a TCP connection to the web page server.

list

This command has no parameters. The server responds with a list of all web pages that it has currently. The client receives this list and displays it.

get *webpage*

The client requests the server to send it the file ***webpage*.** The client receives this web page and types a "web page received okay" message to the screen. If the web page requested does not exist or is mistyped, the server should return an appropriate error message that the client displays.

add *webpage*

The client sends a file ***webpage*** to the server to be added to the server's list of current web pages.

delete *webpage*

The client instructs the server to delete the file ***webpage*** from its list of available pages. If the web page to be deleted does not exist or is mistyped, the server should return an appropriate error message that the client displays.

quit

The client program quits

Note: The retrieved web pages may be tested at the client side (by you or me while I am grading), by loading and displaying them in any browser such as Internet Explorer or Mozilla. If you incorrectly transfer files or they get corrupted, the web pages may not load well. So please test all retrieved pages after being transferred.

The Web page server

The web page server is started first and waits for a connection request from a **single** client stream. The web page server maintains web pages in a directory several web pages that it can send to a requesting client. It also maintains in memory a corresponding list of these web pages that it can send to a client (maximum number of pages in the list should be 80). Initially, you can either create a few simple HTML-only web pages or just get some text only web pages from the web and store them on the server. The actual implementation of the server directory structure and list maintenance data structures is the student's choice. You are however advised to keep things simple.

The command line for initiating the server should be:

Webserver

The web page server responds in the following way to each of the following commands:

login

Upon receiving the login command, the server establishes a TCP connection with the client and returns an Alive! message back to the client.

list

Upon receiving this command, the web page server sends back to the client sends back a list of pages that it has available.

get *webpage*

Upon receiving the Get command, the server searches its list of available pages. If the requested webpage is available, it sends it back to the client. Otherwise, it returns a "Page unavailable" error message.

add webpage

Upon receiving this command, the server adds the name of the web page to its in-memory list, receives the actual web page and stores it in the appropriate directory. You may choose not to worry about duplicates such that the client can add an already-added page over and over. After successfully receiving the webpage, the server sends back a "Page received okay" message.

delete *webpage*

Upon receiving this command, the server searches for ***webpage*** in its list of available pages. If it finds it, the entry is deleted both from its in-memory list , as well as from its location in its webpage directory. If the Delete command is successful, the server sends back a "Deletion successful" message. Otherwise, the server returns a "page unavailable" message to the client.

quit

Upon receiving a quit command, the web server sends back an "exiting" message to the client, closes the connection with this client and continues to wait for connections from other clients.

What to turn in

You should test your programs thoroughly so that they do not crash. Nearly all system calls and library routines return some form of error code if the operation was not successful. **You need to check the return value from every routine for an error code.**

You should create a simple readme.txt file that explains what you have done and how to run it. The readme.txt should explicitly list your programming team or state that you have worked alone. If you work in a team, please describe what each team member did. The documentation does not have to be long, but very clear. Put the readme.txt, your client, server program and all project-1-related files into a directory and tar everything up using the following command:

tar cvf FirstName_LastName_hw1.tar *

where FirstName_LastName is either the individual name if one person worked on the project or FirstName1_LastName1_FirstName2_LastName2 if a team of two worked on the project.

Do not email me (or the graders) your programs. Turn in your assignment using the *turnin* program by class time on the due date**.** You can find details on how to turn in your assignments using *turning* at

http://web.cs.wpi.edu/Help/turnin.html

Programming Hints

You may find the following system calls useful for program 1.

For file operations:

fopen, fgets, fclose, feof, fscanf, fprintf, fputs

For string operations:

strcmp/strncmp, strcpy, strtok, sprintf, strstr, strcasecmp/strncasecmp.

For memory operations:

malloc/free, memcpy/memncpy, bzero, memset

Socket operations:

clearerr(3) , gethostname(2) , listen(2) , close(2) , getpwent(3) , getuid(2) , send(2) , recv(2) , getservbyname(3), gethostbyname(3), gethostbyaddr(3), getsockname(2), bind(2), socket(2), connect(2), accept(2) , listen(2) and select(2) .

For more information on how the Unix networking-related library routines and system calls, see the sections Tanenbaum (sections 6.1.3 and 6.1.4) or the book *TCP/IP Sockets in C*, which explains how to use the Unix library routines.