CSE461
Homework 1: Roll-Your-Own WWW

ASSIGNED: Jan 3, 2006
DUE: Jan 12, 2006

In this homework, you will gain an understanding of the *lingua franca* of the internet, TCP and HTTP. TCP (*transmission control protocol)* is a connection-oriented protocol used for establishing reliable datastreams between programs. HTTP *(hyper-text transfer protocol)* is a higher-level protocol for fetching web pages from web servers.

In order to gain this understanding, you will first build a simple command line web client that can be used to fetch particular web pages from any web server on the internet. Then, you will build a simple web server to which any web client (including your command line version) can connect.

This assignment is moderately difficult, especially if you have limited C and/or Unix experience. You should be sure to get started early.

Part 1. The Web Client

Your web client is a command-line tool invoked from the shell as follows:

% path-to-web-client/webclient hostname port filename

path-to-web-client is the directory where the **webclient** program can be found.

This will fetch and print to standard out the object named **filename** from the web server at host **hostname**, waiting to serve on port **port.** By default, web servers typically listen in on port 80, so you could say:

% ./webclient www.cs.washington.edu 80 index.html

to fetch the top level web page from the CS web server.

Your web client will need to:

1. establish a connection to the designated web server
2. send an appropriately formatted HTTP request message
3. receive an appropriately formatted HTTP reply message
4. deconstruct the HTTP reply message in order to extract the web page itself
5. print the web page to standard out.

One of the tricky aspects of this assignment is going to be constructing and deconstructing appropriately formatted HTTP messages. First, you must discover the HTTP format for request messages. There are a number of ways you can do this, including:

1. reading the HTTP spec.
2. connecting to the the simple little "server" program from lecture via a standard web browser and seeing what gets displayed. To do this, you would enter as a URL hostname:port/objectname, where hostname could be "localhost" if your browser is running on the same machine as your server program.
3. watching tcp traffic using tcpdump

Once you understand how to construct an HTTP request, you will need to deconstruct an HTTP reply from the server. As before, you can read the spec, you can use your nearly functioning webclient program to dump and then inspect the entire response message, or you can use tcpdump to observe live traffic.

Part 2. The Web Server

For the second part of this assignment, you will build a web server that serves up a small collection of static (file) web pages. To start your web server, you will type:

% path-to-web-server/webserver port service-directory

port is the port on which the web server should await requests. **service-directory** is the directory from which the web server will find pages.

For example,

% ./webserver 9999 ~/tmp/web

would serve up web pages from your home directory's **tmp/web** directory via port 9999.

You should make sure your web server works both with your command line web client, and from a standard web browser. Of course, you will need to populate your **service-directory** with content.

Please make sure that your webserver prints a message to standard out announcing each incoming client request including:

- time of request
- hostname of calling client
- port number of calling client
- requested filename

What to turn in

You should turn in a single tar file containing:

- two source files: **webserver.c** and **webclient.c**
- a Makefile, such that "make" builds webclient & webserver.
- a typescript called "typescript.client" that shows your client fetching index.html from www.cs.washington.edu the splash page from google.com, and a file called "hello.html" from your own personal webserver.
- a typescript called "typescript.server" that shows your server processing the request for hello.html

Details on how to submit this tar file will be provided in section and/or on the class web site.

Evaluation Criteria

This assignment will be graded on an integral scale of 0 to 4. As with all assignments in this class, your evaluation will be based on what the work you have done demonstrates in terms of your understanding of the core concepts being evaluated.

4: You have demonstrated a clear understanding of the use of sockets in a client/server application, and the role of HTTP in such an application.
3. You have demonstrated a clear understanding of sockets or HTTP, but not both.
2: You have demonstrated a partial understanding of sockets and HTTP.
1. You have demonstrated little or no understanding of sockets or HTTP.
0. You have not demonstrated anything of relevance to this assignment.

Some Things to Keep in Mind

1. You may do this assignment on any UNIX or UNIX-like (including Mac OS) machine. Please do not do it on windows.

2. If you are doing this assignment on a shared computer (such as the departmental linux cycle server), then you need to keep in mind that the **port** space is both limited and shared on any single machine. If you try to bind your server to a port number that is already being used by another server instance, the OS will report an error (***Address already in use***; see *man bind*) You should pick another one. You may not use ports less than 1024, or greater to 64k. That leaves 63,000 server ports for you to share with others in the class who are currently working on their assignment. The UNIX command "netstat -n" will show you which ports are currently in use. Alternatively, you can devise amongst yourselves a port-space sharing protocol that will reduce the likelihood of conflict. *One tricky aspect of this is that if your server program terminates while messages are outstanding on a port, then the port goes into a state where it can't be used for a few minutes.* Either wait, or find another unused port.

3. A typescript is a record of all shell activity. See *man script* for information on how to produce.

4. You should study the sample client and server programs discussed in lecture.