

Programming Assignment 1

*Assigned: Feb. 11, 2009**Due: Feb. 27, 2009*

In this programming assignment, you will implement a HTTP server and a HTTP client in **C** or **C++**. The goal is to use basic socket programming interfaces to implement a very small (yet core) part of the HTTP protocol and practice TCP socket programming using multiple processes. The basic requirement is that your client can request web pages from your server and other web servers such as www.cs.uky.edu, news and search servers, and you can also use Firefox/Internet Explorer/Netscape to get web pages (possibly with embedded images) from your server. You can start with the sample codes from the “TCP/IP Sockets in C” book, but you have to implement the major functions of the assignment. The sample codes are `TCPEchoClient.c` and `TCPEchoServer-Fork.c`. They can be found at <http://cs.ecs.baylor.edu/~donahoo/practical/CsSockets/textcode.html>.

Server

The server can host any number of web pages and its interface should be:

```
myserver port_no http_root_dir_name
```

where `port_no` is the port number that the server will listen to, and `http_root_dir_name` is the directory under which you will put all the html and image files of the web pages. The server should be able to process GET requests from clients. It will ignore request messages using other methods. If the requested document exists at the server, it will send the client a reply formatted according to the HTTP protocol with status code 200 and the content of the file. The server should be able to serve four types of files: txt, html, jpg, and gif. File types are determined by the extension part of file names. If the document does not exist, it will reply with the status code 404 and other relevant information based on the HTTP protocol for the browser to display.

When the server is processing a request from one client, requests from other clients may come. In order to process these requests immediately, the server should use multiple processes. Basically, the main process of the server keeps listening to new connection requests from clients. Once a connection request is accepted, the main process will create a child process to respond the request from the client on that connection while the main process is watching out for other connection requests.

Client

The client can request a web page and store it to a file in the current directory. Its interface should be:

```
myclient URL local_filename
```

where URL should be in the form of `http://host_name:port_no/file_name` and `port_no` and `file_name` are optional. Here is an example:

```
myclient http://squill.cs.uky.edu:9872/mywebpage.html storepage.html.
```

When we use the client to access a web page from other web servers, we may run the client as

```
myclient http://www.yahoo.com yahoopage.html.
```

HTTP Protocol

You will get a better understanding of the HTTP protocol through implementing its core part. Here we outline the major steps taken by the client and the server, and messages exchanged between them.

When a client is running with a URL `http://host_name:port_no/file_name`, it will use DNS (i.e., `gethostbyname()`) to get the IP address of the host represented by `host_name`. It opens a TCP connection to the host at port indicated by `port_no` (use the default port number 80 if no `port_no` is given). Then the client will form a HTTP request and send it through the TCP connection to the server. If the URL is in the form of `http://host_name:port_no`, i.e., without `file_name`, the client should use “/” as the file name in the “GET” request. The last step is to read from the TCP connection and store the reply (header+file) obtained to the local file named `local_filename`.

At the server side, it needs to listen at the `port_no` specified. When a server gets the request for a web page represented by `file_name`, it will check whether the file exists. If yes, it will send the file back to the client, with HTTP response header (status code 200) added. If the file does not exist, it will send a message back indicating that the document requested is not available. Note that `file_name` sent by the client is a path relative to the `http_root_dir_name`, and you need to form a full path name for the file, i.e., `http_root_dir_name/file_name`. The server should be able to deal with those cases in which the file requested is simply `"/`". It should read file `http_root_dir_name/index.html` and send it back. (Note that we do not require that your server deal with the general case that the filename is a directory name.)

You need to provide a makefile or tell how to compile and link your programs. Your programs will be tested on **multilab** machines. Robustness of the server/client is one of important aspects when your codes are tested. Comments are required. Also write a README file to give general descriptions about your programs and instructions to run them, and state any limitations of the implementation. Mail your programs to `fei@netlab.uky.edu` as one attachment. You should tar all the files together instead of using multiple attachments.