ITCS 6166
Spring 2014
Project 1
Socket Programming

In this programming assignment, you will implement an HTTP client and server that run a simplified version of HTTP/1.1. Specifically, you will implement two HTTP commands: GET and PUT. This project can be completed in C++, Java, or Python. Other languages may be allowed, but only with advance permission from the instructor and TA.

1. HTTP Client

Your client should take the following command line arguments (in order): server name, port on which to contact the server, HTTP command (GET or PUT), and the path of the requested object on the server. In other words, assuming that your executable is named "myclient", you should be able to run your program from the command line as follows:

```
myclient hostname port command filename
```

In response to a GET command, the client must:

1. connect to the server via a TCP connection
2. submit a valid HTTP/1.1 GET request to the server
3. read the server's response and display it

In response to a PUT command, the client must:

1. Connect to the server via a TCP connection
2. submit a valid HTTP/1.1 PUT request to the server
3. send the file to the server
4. wait for the server's reply
5. read the server's response and display it

2. HTTP Server

Your server should take a command line argument that specifies the port number that the server will use to listen for incoming connection requests. In other words, assuming that your executable is named "myserver", you should be able to run your server from the command line as follows:

```
    myserver port
```

Your server must:

1. Create a socket with the specified port number
2. Listen for incoming connections from clients
3. When a client connection is accepted, read the HTTP request
4. Construct a valid HTTP response:

    a. When the server receives a GET request, it should either construct a "200 OK" message followed by the requested object or a "404 Not Found" message.
    b. When the server receives a PUT request, it should save the file locally. If the received file is successfully saved, the server should construct a "200 OK File Created" response.
5. Send the HTTP response over the TCP connection with the client
6. Close the client connection
7. Continue to "loop" to listen for incoming connection

Your server will have an infinite loop to listen for connections. To shut down your server, you will have to interrupt it with a termination signal. Upon receiving the termination signal, your server must shut down gracefully, closing all sockets before exiting.

Advice on how to tackle this project:

- Implement your client first.
- Test your client's GET command with an external HTTP server.
    - Before you test your client with your server, test it with a web server that you know works. In other words, use your client to a get a file from some known external HTTP server. For example, from the command line: myclient www.amazon.com 80 GET index.html
- Now implement the server.
- Test the server with a browser (e.g., Firefox) as a the HTTP client.
    - For example, your server is running at host pc1.cs.uncc.edu on port number 12000, and there is a file index.html in the current directory. In the URL box of the web browser, type:
      pc1.cs.uncc.edu:12000/index.html
      The browser should fetch the file and display it.
- Now use your client to get a file from your HTTP server. For example, if your server is running on pc1.cs.uncc.edu port 12000, then enter at the command line: myclient pc1.cs.uncc.edu 12000 GET index.html
- Now use your client to put a file on your HTTP server.

Other tips:

Make sure to choose a server port number that is greater than 1023. Ports from 1-1023 are often used for well-known and widely used network services (e.g., port 80). Choosing a port number over 5000 is advisable.

You can run all of the processes (all clients and the server) on the same machine. In this case, use "localhost" for the machine name. It is strongly recommended that you test your project in this way first, since you may sometimes run into issues with firewalls that are difficult to puzzle out. You can run do this from a Unix command line by running the server process in the background and then starting a client (if you want to open more than one client at a time, open another terminal and start another client). Remember, from a Unix command line, you can use the ampersand (&) after the process name to run that process in the background.

If something goes wrong, you may want to use the Unix command *kill* to stop the server process. To find a process ID, use the Unix *ps* command.

Make sure you close all sockets. If you don't, and you try to bind to a port that was previously in use, you'll get an error. Make sure you are catching exceptions/handling errors. If you want to check to see what port numbers are in use on a Unix machine, use *netstat* from the command line.

If you are testing across multiple machines and want to know the IP addresses of those machines, use *ipconfig*.