CS260 Assignment #4:
HTTP Proxy Server

Task
This assignment builds off assignment 3 to create a simple HTTP proxy server.

As with assignment 3, you will construct an HTTP request. Rather than getting the URL from the command-line arguments, however, you will open a socket and listen for incoming HTTP requests. (Get the port number for this listening socket from argv[1].) Then you will *proxy* those requests by opening a socket to the destination server, sending the HTTP request, receiving the response and sending that response back to the original client. In effect, it will appear to the client as if your program is the server, and it will appear to the server as if your program is the client.

For example, an incoming request to your server might look like this:

```
GET /mrrobot HTTP/1.1
Host: www.usanetwork.com
```

You would then perform the following steps:

- Use DNS to fetch the IP address of www.usanetwork.com.
- Open a TCP socket to port 80 on that IP address.
- Send that server the same request your server received. Pass along every header you were sent, even the ones you didn't parse.
- As you receive response packets from the target server, send them along to the client. You will *also* need to parse the response packets so that you know when the target server has finished the request.
- Close the socket to the target server cleanly.
- Close the socket to the client cleanly when it has received the last of the response.

In addition, your server will need to use non-blocking IO throughout, because your program will be tested with three clients simultaneously. (You're free to make your program support *more* than three simultaneous proxy requests, but it will only be tested with three.)

Linux Only
You do not need to support Windows with this assignment, nor do you need to support cross-platform compilation (though you may if you wish). Submit only a Linux project.

Testing
There are two sides of testing your program: the client connection to it, and its connection to the target server.

The latter is easy to test. As with assignment 3, your program implements the HTTP standard and can be tested against any web server. The packet-logging page at http://echo.cs260.net/packets continues to work as a log of HTTP packets. In any case, as long as your assignment 3 worked, the outgoing request from your program should also work, because that part is not really any different.

The former is a little trickier. Remember that standard web browsers—including your assignment 3 client—work by parsing the URL to get the hostname, then doing a DNS lookup, then opening a socket to that IP address. What you want the client to do in this case is replace that IP address with the address (including port number) of *your* program, but otherwise compose the same request (including the correct server name in the Host header). Browsers won't do this by default. There are two ways you can test your program:

1. Connect to it by telnet and type the request in by hand.
2. Create a modified version of your assignment 3 client that skips the DNS lookup step and instead always connects to your assignment 4 proxy.

Submission
Your Linux project must have a makefile and compile with nothing installed except build-essential. Once finished, upload your project folder and upload it to Moodle as a zip or tar. You should clear out any build detritus (executable files, object files, symbol tables, etc.).

Commented [RW1]: Clarified that zip and tar are both accepted submission formats.

Grading
From a base score of 100:

- If your program can't be compiled with make on Linux Mint, with nothing but build-essential installed, you lose 100 points.
- If your program doesn't open a listening TCP socket on the port specified by argv[1], you lose 70 points.
- If your program doesn't parse the incoming HTTP request and connect to the intended target server, you lose 70 points.
- If your program does not correctly and completely pass along the HTTP request, you lose up to 40 points, depending on how close to correct formatting you came.
- If your program does not correctly and completely pass along the HTTP response, you lose up to 40 points, depending on how close to correct formatting you came.
- If your program closes either the outbound server socket or the inbound server socket abnormally, you lose 30 points.
- If your program does not allow three clients to make separate requests simultaneously, without any of the requests blocking the others, you lose up to 30 points.
- If your program has memory leaks, you lose 10 points.
- If your submission has build detritus, you lose 10 points.

If you score 89 or fewer points, you may correct errors and resubmit for the usual 10-point penalty. ***However***, because this is the last assignment of the semester, resubmission is *only* possible if you deliver your assignment at least 3 days early, so that it can be graded ahead of time. Assignments delivered on or after the due date cannot be resubmitted. **No assignment or resubmission will be accepted after 11:55pm on the Wednesday of the last week of class.**