ECE:3540 COMMUNICATION NETWORKS
SOCKET PROGRAMMING PROJECT, FALL 2018

Due: 11:59 pm, Friday, October 26, 2018

The objective of this project is to gain a basic understanding of socket programming by developing C code for: (1) a "proxy" file server that, upon client request, retrieves a file from a "source" file server, censors it, and returns it to the client; and (2) a file retrieval client capable of making these requests. The relations among the file servers and the file retrieval client are pictured below.

The Source File Server:

The source file server is an HTTP server that hosts two files, `alice.txt` and `war.txt`, containing the text of *Alice's Adventures in Wonderland*, and *The War of the Worlds*, respectively. These files are accessible at http://user.engineering.uiowa.edu/~jwryan/Communication_Networks/ on port 80.

The Proxy File Server (`getfile_server`):

You are to implement the proxy file server as a server to your file retrieval client and a client to the source file server. Your proxy should accept TCP file requests from your client, download the requested file from the source file server, replace each occurrence of a specified word of length *n* with *n* asterisks (i.e., "**…*"), and forward censored file on to your client via TCP. It should also print to the terminal brief status messages, including: "Waiting for connection…", "Connected to client at 127.0.0.1:x", "Downloading y.txt…", "Download success", "Replacement count: z", and "Closing connection", where z denotes the number of "**…*" replacements made. If a bad (e.g., non-existent file) request is made it should additionally, print "Bad request" and return a bad request message to the client. You may bind your server to any free port between 16200 and 16300. Just be sure to close your port when you are done.

The File Retrieval Client (`getfile_client`):

Ideally, your file retrieval client would be implemented on a second host distinct from the proxy file server host. For simplicity you will instead implement the client on the same host as your proxy server. Therefore, regardless of what host you start your proxy server on, your client's requests should be addressed to the host's loopback address 127.0.0.1. The file retrieval client communicates with the source file server through the proxy file server. The client may request either `alice.txt` or `war.txt`. The request is then sent to the proxy server for it to download, censor, and return via TCP. The client should print a "Bad request" message when the client receives a bad request message from the proxy server.

Examples of correct client and proxy server interaction:

```
$ getfile_server <word to replace> <server port>
Waiting for connection…
Connected to client at <server host>:<server port>
Downloading war.txt…
Download success
Replacement count: <number of replacements made>
Closing connection
```

```
$ getfile_server <word to replace> <server port>
Waiting for connection…
Connected to client at <server host>:<server port>
Downloading file_does_not_exist.txt…
Bad request
Closing connection

$ getfile_client <filename> <server host> <server port>
Connected to <server host>:<server port>
Requested <filename>
Receiving file…
Transfer complete

$ getfile_client file_does_not_exist.txt <server host> <server port>
Connected to <server host>:<server port>
Requested file_does_not_exist.txt
Bad request
```

In summary, individually you need to write two C programs (one for the proxy file server and one for the file retrieval client) to demonstrate your understanding of socket programming. Submit both C files in a .zip file to the ECE:3540 ICON dropbox by the project deadline **11:59 pm, Friday, October 26, 2018**.

Note: **Your program must compile and run on ECS linux platforms using the gcc compiler**. We will not attempt to compile your code with any other compiler or run it in any other environment. Grading will be based on: your programs' structure, execution and documentation. If you have questions concerning the assignment, or cannot connect to the source server, please first contact Mr. Ryan as he is coordinating the assignment.

Sockets References:

- The UNIX/LINUX man pages list details of most socket calls
- Any of many online sockets resources, e.g.,
  - http://beej.us/guide/bgnet/
  - http://www.cis.temple.edu/~ingargio/old/cis307s96/readings/docs/sockets.html
- Any of several excellent texts, e.g.,
  - W. Richard Stevens, UNIX Network Programming: vol. 1, 3rd ed., Prentice Hall, 2004.

Additional Notes:

- This is NOT a group project. You are to develop and submit your own code.
- One approach to getting started on the project could be first get the timeserver.c and timeclient.c codes presented in LN2 (pp. 113-122) running and then use them as starting points for your file retrieval client and proxy file server development, i.e., create timeserver.c and timeclient.c, compile them using gcc (e.g., gcc -o timeserver timeserver.c) and then run them.
- You can list the in-use ports in the range 16200-16300 on your host using either of the following :
  - `$ nc –zv 127.0.0.1 16200-16300 2>&1 | grep succeeded`
  - `$ nmap -p 16200-16300 --open 127.0.0.1`

  You may safely bind your proxy server to any *unlisted* port in the 16200-16300 range. Once your socket is bound it will show up in the lists produced by these commands.
- You can determine the process ID (PID) of your server using either of the following
  - `$ lsof -Pan -i tcp`
  - `$ ps –a`

  (you will need to know the PID should you need to kill it using the command "kill <PID>").
- When testing your client and server you may find it helpful open three windows on your host: one for the server, one for the client, and one for issuing monitoring commands such as those listed above.