Assignment 1: Introduction to socket programming

Your first assignment is to build a TCP server capable of receiving messages from clients. The server should print these messages on standard output, but should not print any other messages such as debug information. From the server perspective, a message corresponds to the data received from a particular client during a communication session with that client. The server should be listening for messages to a port known to the clients. The server should be able to receive messages from multiple clients. When multiple clients simultaneously try to send messages to the server, the server should print them one at a time (in any order). Each message should be separated by ENTER whether those are sent from one client or multiple clients. Note that you don't have to implement an event-driven or multi-threaded server. Use fork() to accept multiple clients.

You should also implement a client to test the server. The client should receive the message from standard input. The client should send the message when you Hit ENTER. Hitting >ENTER< twice is a control sequence for the client to exit. This control sequence should not be transmitted. Additionally, a single ENTER should not be transmitted which means that when you hit ENTER without any message, it does not send anything. Also, if your client reads the message from a file via pipes, and reaches EOF without seeing the control sequence, then the client should still transmit the message and exit.

This assignment should be completed in ANSI C (or C++). It should compile and run without errors in the EE323 Lab cluster producing two binaries called server and client. The server should take "-p" and the port to listen on. The client should take "-h" and the host that the server is running and "-p" and the port that the server is listening on (in any order: -h host -p port or -p port -h host). If the server cannot bind to the port that you specify, a message should be printed on standard error and the program should exit. You shouldn't assume that your server will be running on a particular IP address, or that clients will be coming from a predetermined IP address. Both the client and server should generate an appropriate error message and terminate when given invalid arguments.

Recommendation

You should read uploaded Beej's guide to network programming carefully before start this assignment. We do not give you any skeleton codes for this assignment. To do this assignment easily, you may need to understand the example stream server and client codes in uploaded guide.

You should test your code with long messages (of size at least 300KB), not just short ones. You can use pipes to redirect the standard input of the client and standard output of the server. You should also test your code with multiple clients. (at least up to 5 simultaneous clients)

Readme

Use emacs to create a text file named readme that contains (not readme.txt, or README, or Readme, etc.)

Your name, student ID, and the assignment number.

A description of whatever help (if any) you received from others while doing the assignment, and the names of any individuals with whom you collaborated, as prescribed by the course Policy web page.

(Optionally) An indication of how much time you spent doing the assignment.

(Optionally) Your assessment of the assignment: Did it help you to learn? What did it help you to learn? Do you have any suggestions form improvement? Etc.

(Optionally) Any information that will help us to grade your work in the most favorable light. In particular, you should describe all known bugs.

Descriptions of your code should not be in the readme file. Instead they should be integrated into your code as comments. Your readme file should be a plain text file. Don't create your readme file using Microsoft Word, Hangul (HWP) or any other word processor. Your readme file should be written in English.

Grading

The deliverable is one tar file which contains "readme", "server.c", "client.c", and "Makefile". 'make' should produce server and client from the source code.

Submission: 1 point

0.5 pt: Submission should include a correct set of files: client.c server.c readme Makefile.

0.5 pt: Makefile should work without an error

Basic functionality: 7 points

1 pt: Correct command-line parameter handling

server argument: -p listening port

client argument: -h server host name, -p server listening port

3 pts: Correct client behavior

3 pts: Correct server behavior

Advanced functionality: 2 points

1 pt: Handling simultaneous messages from multiple clients

1 pt: Handling long message