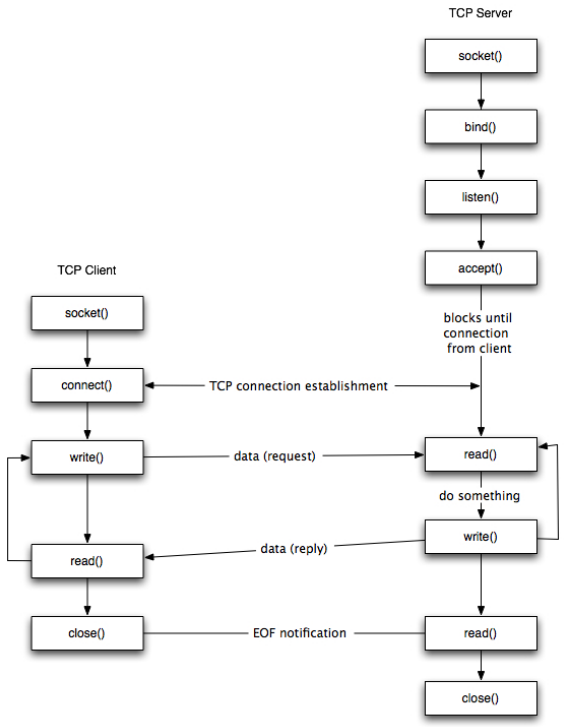
CMPE150 Final Project

For this project you will be building a remote shell application using the C programming language and its socket API. You will be implementing a client that will connect to a server running on a remote host to access its shell and run commands. The Socket API is used very frequently to build network applications for most operating systems and low level hardware. To read more about the orgin of the API and its uses please see: (http://en.wikipedia.org/wiki/Berkeley_sockets)

Socket Overview:

First you may find it very helpful to refresh your memory on how memory allocation and pointers work in C. There is plenty of references online for these topics but here's one to start with:
http://pfacka.binaryparadise.com/articles/guide-to-advanced-programming-in-C.html

You will have to learn how to setup a TCP connection with the sockets API. This diagram details the APIs you will need:



You will need to learn how to run the commands at the server and then retrieve the output and send it back to the client. This will require you to allocate memory to hold

the output strings.

Remember that all Socket API calls have return values for success and errors; you will need to capture return values and output it to the terminal if an error occurs.

For more on the Socket API please use these references:

http://www.beej.us/guide/bgnet/
https://www.youtube.com/watch?v=V6CohFrRNTo

Running your program:

You will need to create a makefile to compile your apps and separate out your files into their correct directories (e.g., src/, bin/, doc/).
For more on makefiles: http://mrbook.org/tutorials/make/

The Client program will be started using the following command:

./<ClientProgramName> PORT IPADDRESS

where PORT is the correct TCP port and IPADDRESS is a specified IPv4 address.

The Server program will be started similarly:

./<ServerProgramName> PORT

Team Rules:

You make work in teams for this project however you are only allowed to have a maximum of 2 people in a group. Each person must make a separate submission of the same project. Additionally, **both team members must be present to demonstrate the project.** One way to split up the work is to have each team member work on the client and the other work on the server. However both team members will need to interact on their design and implementation such that the client and the server communicate and work correctly.

Rubric and Grading:

200 POINTS TOTAL
Source Code (40%):

- Source code passes through checksource without any errors.
- Client can send a sequence of commands and receive output correctly.
- Minimally, the following commands should be implemented: ‘ date, ls, pwd, shutdown’
- The server can handle multiple client connections(not necessarily simultaneous), one after another.

- Client program stops when a shutdown command has been issued.

Documentation (10%)

- README, containing how to run and compile your code
- Explanation of how your client and server communicates including the handshake process and closing the connection. This can be included in a separate document which will be part of your submission.
- Well commented code.

Error handling (20%):

- Program handles unspecified client exits or server closures without crashing.
- Program reports all errors and warnings to stderr.
- Source code compiles without warnings (in gcc -Wall).

Project Demo (30%):

- Client can connect to server.
- Run at least the following commands: ‘ date, ls, pwd, shutdown’.
- If the client or the server terminates its connection early, the server or the client does not crash.
- The server can handle unspecified input.

*****Extra credit for 25 POINTS each*****

1. Implementing commands requiring the spawning of other processes: ‘ | , & , etc.’
2. Allowing simultaneous client connections using threads. For more info on Pthreads see: https://computing.llnl.gov/tutorials/pthreads/

Help & Reminders:

- Start early.
- Make sure that your compiled binary is able to run on the machines in the lab.
- Write a single code section at a time; use “dummy” code to allow you to focus on only one piece of code at a time.
- The instructor and TAs are not responsible for getting your code to compile and run on the lab machines; it is your responsibility to ensure your program runs on the machines you will use to demo it and compile on PC1 of each POD.
- Partial credit will be assigned based on the rubric and grading criteria specified above.
- **Late projects will not be accepted.**
- Cite your sources if you're basing your code structure of someone elses.