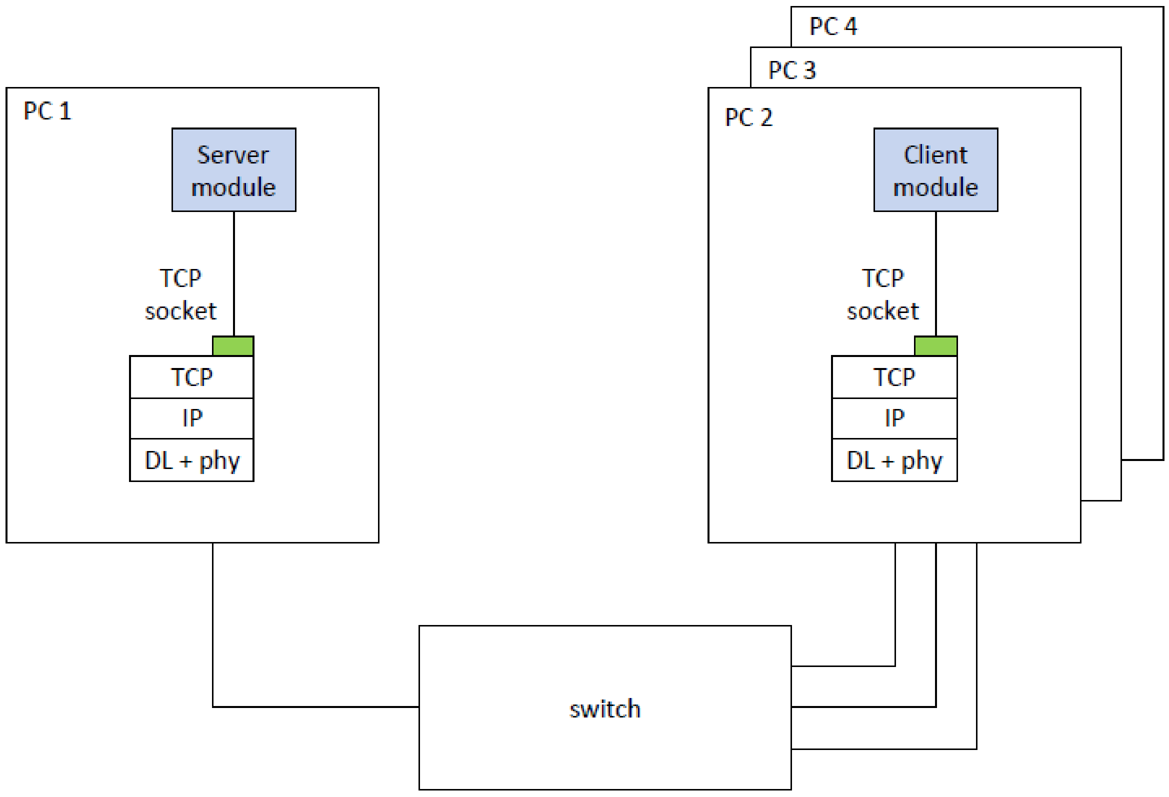
Project eVote

CMPE 150L - Winter 2012

The goal of this project is to gain some experience in socket programming. You will develop a simple client-server application on the Linux computers in the lab. One of the machines in the pod will act as the server and the other three will be clients. You will need to develop two separate software modules: the server module and the client module.

Network Configuration
You will configure your pod as shown in the figure below. One of the PCs will run the server module and the others will run the client module. You may optionally run more than one client process on the same machine, thus increasing the number of client processes beyond 3.



Application
Your server and client modules together should implement a TCP-based electronic voting system that works as follows. The client module will allow a voter to cast a “write-in” vote for the poll. It will also allow the current candidates and poll results to be displayed. The server module will keep track of write-in candidates submitted by the voters, as well as the current poll results. It should only allow each voter to submit one candidate, and should make the current results available upon request.

Server Operation
Usage: ./evs
Listening port: 1337

The server process is started before starting any client process. When started, the server waits for clients to contact it (this is achieved by opening a “listening” TCP socket and keeping it open until the voting period ends). When contacted by a client (by the client opening a connection to the server’s listening port), the server must be able to handle at least two types of client requests:

- Vote – the server will receive a voter ID number and a “write-in” candidate name from the client. A check must be completed to see if the voter with the given ID number has already voted. Then, if the given candidate is not currently on the list, it should be added. The number of votes for the given candidate should be incremented by 1. The server should indicate to the client whether the vote was successfully recorded or if it failed (perhaps because the voter already voted).
- Results – the server will receive a request for the current poll results. It should return to the client a list of candidates along with the number of votes each candidate has received.

The server should close their connection with the client after fulfilling all client requests. The server should close the listening socket and exit gracefully when Ctrl+c is pressed on the command line.

Client Operation
Usage: ./evc [-r] [-v candidateName] [-s serverIP] voterIDnumber

The client starts a TCP connection to the server with the given IP address and the default port number of 1337. The client port number should be dynamically allocated. The client must be able to handle at least these command line options:

- -s serverIP
  Server – required – specifies the IP address of the server.
- -v candidateName
  Vote – the client will send a voter ID number and a candidate name to the server. The client should display whether the vote was successfully recorded or not.
- -r
  Results – the client will send a request for the current poll results. If used in combination with the vote option, it will ask for the results after the vote is recorded. The client should display the list of candidates returned by the server, along with the number of votes each candidate has received.

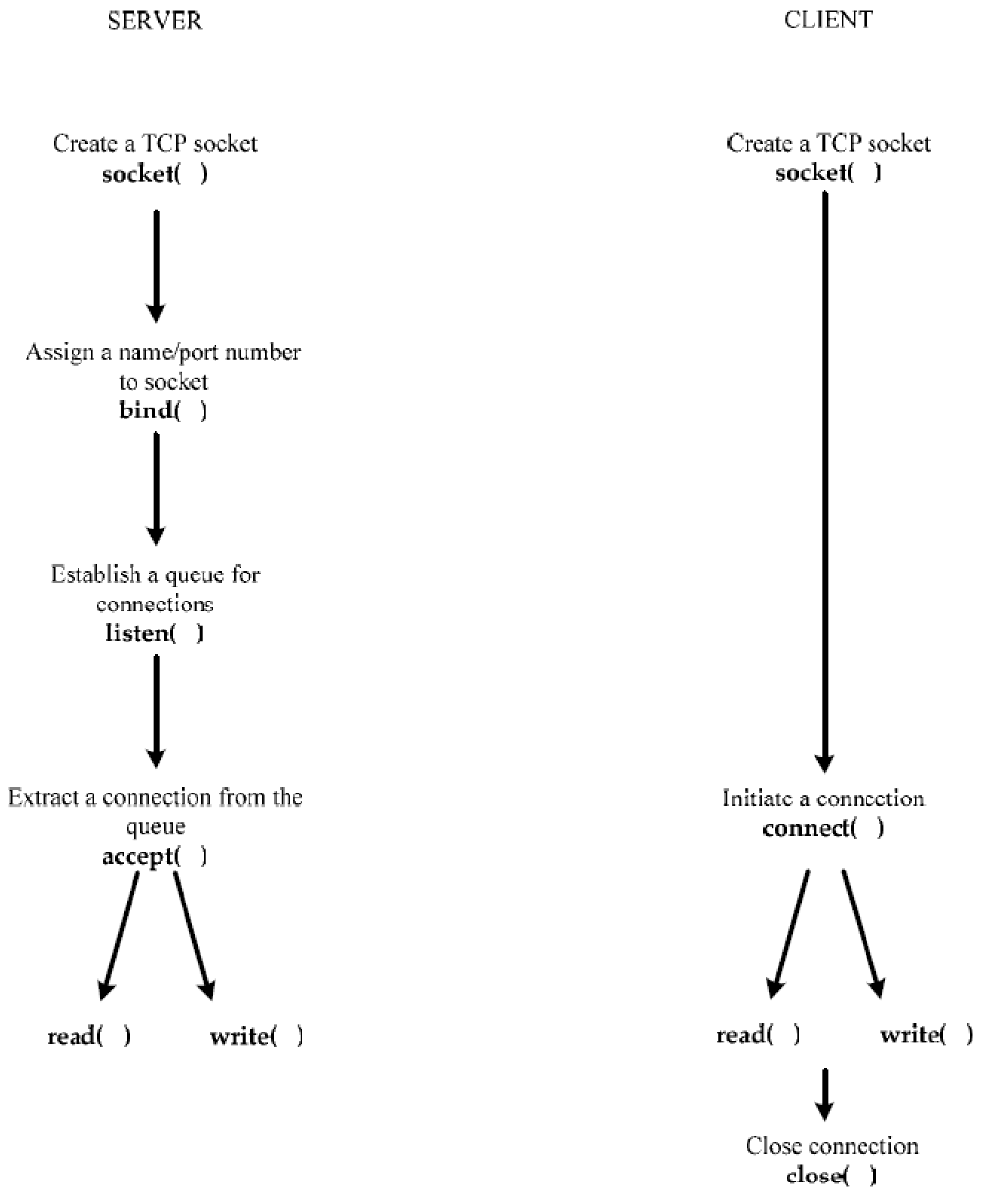
After completing the command line actions, the client should exit gracefully.

Implementation Notes
To avoid issues with case sensitivity for candidate names, the client or server should convert all names received to upper or lower case, so that ‘James’ and ‘JAMES’ are treated as the same person.

You may assume the voter ID is an integer.

Communication sequence

The basic Linux system calls to be used to implement the client and server modules are outlined below.



The server process first creates a socket using the socket system call. The bind call assigns it a name, and also allows you to assign a listening port number. The listen initializes a queue for the incoming connection requests. You can then wait for an incoming connection request from a client using the accept call. This system call will perform a blocking wait and return with the parameters of the first connection request received. You can then send and receive data to/from this client using the write and read system calls (Note: Meanwhile, you can continue to listen for new requests using the accept call.)

The client process also uses the socket system call to create a TCP socket, but then uses the connect call to set up a TCP connection to the server. It can then send and receive data using read and write, and close the connection using close when it is done.

Logistics
The project can be done either individually or by teams of two students. In a two-student team, one of the students must develop the server side of the application and the other the client side. As you will both need to collaborate to develop a protocol or message format for communication between the client and the server, it is suggested that you work together on the design document.

The description above is meant only as a starting point. Use your creativity to enhance it, add more features, better UI, increased robustness against failures, etc. Extra credit will be granted for projects going beyond the minimum.

You must write your code in C or C++.

Deliverables

Project deliverables include a project report and a project demonstration. *Demonstrations are mandatory*; code not checked-out with a lab instructor *will not be graded.*

The project report should include the following:

- Quick start guide, with extra emphasis on any additional features
- Design document, including:
    - Design considerations
    - System architecture
    - User interface
    - Client/server interface
- Source code (attached)
- If working on a team, the report should indicate the division of labor.

Project demonstrations will be held during the regular lab sessions in the week of March 12.
The report and the source code must be submitted through eCommons before 11:55pm on Friday, March 16. If working in a team, both partners should submit the same report/source code.

Grading (out of 100 points):
50 – Project Report
50 – Demonstration/Code
30 – Extra credit

Resources
There are a large number of books and online sites that show examples of the use of Linux sockets. A popular online guide to the use of sockets can be found at
http://www.beej.us/guide/bgnet/

If you really want to delve deep into network programming, the book *TCP/IP Illustrated* by Stevens is a great reference.