The goal of this project is to gain some basic network programming experience, in particular in the use of TCP sockets. You will develop a simple client-server application that runs on the Linux computers in the lab. One of the machines in the pod will act as the server and the other three may be clients. You will need to develop two separate software modules: the server module and the client module. You will be allowed to work in teams of 2: one group member should work on the client and the other on the server.

Network Configuration
You will configure your pod as shown in the figure below. One of the PCs will run the server module and the other(s) 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 Tic-Tac-Toe gaming system that works as follows. The client module will allow a player to play a game against the server. The server module will have an AI that responds to player moves.

1 Note that you should be able to develop your program on any Linux machine; however, the program must run correctly on the lab machines as you will use them to demo your project.
Server Operation

Usage: ./ttts

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 game ends). When contacted by a client (when the client opens a connection to the server’s listening port), the server must be able to handle at least two types of client requests:

- Move – the server will receive a move from the client. If the game is not over, it should respond with a move of its own. If there are no more moves available or if one side won, it should record whether the game ended in a win, a loss, or a draw for the server.
- Results – the server will receive a request for the current AI results. It should return to the client the current AI stats (wins-losses-draws).

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: ./tttc [-r] [-s serverIP]

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.
- -r
  Results – the client will send a request for the current results. The client should display the server AI’s wins, losses, and draws, then exit gracefully.

If the ‘-r’ option is not used, it should start up a new game against the AI.

Implementation Notes

You may have the server store the current game state (and have the client identify itself with each move), or have the client send the entire game board each time.

How you implement the user interface at the client when it comes to playing the game is up to you.

Minimum:
- handle one client at a time

Full Credit:
- handle multiple clients at a time

Extra Credit (make sure you’ve completed the above first!):
- improved AI
- improved UI
- allow two clients connected to the server to play against each other
- insert your idea here
Communication sequence

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

**SERVER**

Create a TCP socket
socket( )

Assign a name/port number to socket
bind( )

Establish a queue for connections
listen( )

Extract a connection from the queue
accept( )

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.)

**CLIENT**

Create a TCP socket
socket( )

Initiate a connection
connect( )

**read( )**
**write( )**

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.

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 (up to 20% above 100).

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

Project demonstrations will be held during the regular lab sessions in the week before finals. Demonstrations are mandatory; projects not checked-out with a lab instructor will not be graded. All files must be submitted through eCommons before 11:55pm on the Friday before finals.

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 another reference I can recommend.