Introduction
You will be writing the code for a client program that communicates using sockets with a server that I will be providing for you. I will also give you a skeleton of the client program that does command line argument parsing for you. You will be filling in the code that does the actual communication with the client. The tar file for the project also contains a binary of the server, the client skeleton, a file called “NetworkHeader.h” that you should include in your client, and a simple “echo server” that prints any messages it receives on standard output - a useful tool for debugging your client. (If you haven’t encountered tar archives before, check out the “File archives and compression” document on the “Tools for Computer Scientists” web page (http://www.mathcs.richmond.edu/CS/Tools.html).

The Protocol
For this program, the protocol for communication between client and server has only three types of messages: HELLO, ACK and BYE. HELLO messages are sent from a client to the server and ACK messages are sent from the server to a client. Each message is a simple character string which consists of several fields, and which must be terminated by a newline character, ‘\n’. The fields of the message must be separated by a blank character, and the characters sent cannot include the null terminator character ‘\0’ which is normally a part of all C style strings. The number of fields varies with the message type.

HELLO
The HELLO message has four fields:
• a version field, which should be set to “CS332” for this project
• a type field, which should be “HELLO”
• a firstname field (this is your first name)
• a lastname field (this is your last name)
A sample HELLO message in C string format is as follows:
CS332 HELLO BOB SMITH\n
ACK
After establishing a connection with the server and sending a HELLO message, the client waits for a reply. When the server receives a HELLO message, it sends back an appropriate ACK message. The ACK message has three fields, all separated by a single space (remember, no ‘\0’ characters), with the final field followed by a newline character. The fields are:
• a version field (“CS332” again)
• a type field, “ACK” in this case
• a cookie – this is a string that must be set by the person who starts the server.

**BYE**

Once the ACK containing the cookie is received, your client is ready to end its conversation by sending a BYE message. The fields are:
• a version field (“CS332”)
• a type field, which should be “BYE”
• the cookie sent by the server.

**The Client program**

Here is the command-line syntax for the client:

```
Project1Client [-s <hostname>:<port>] -f <firstname> -l <lastname>
```

The `<firstname>` and `<lastname>` arguments correspond to the first and last name fields in the HELLO message. Remember that they shouldn’t contain spaces! The hostname and port specifications are optional, and only need to be used if the person running the server has chosen to override the default definitions provided in NetworkHeader.h.

**The Server**

The command line syntax for the server is

```
Project1Server -s <cookie> -p <port>
```

The server responds to correctly formatted client messages by printing the content of the received message. When the server receives the BYE message, it adds that it has “received the final client message,” prints out this final message, and continues to run, waiting for additional clients that may wish to contact it. My advice is that when coding (and debugging, etc.), you pick one of the Linux boxes to run your server on (mathcs01 to mathcs10), with the port set to some random port number that you choose between 5001 and 5100. This is to avoid all of you getting error messages about a port being in use when you run the server, which is what would happen if you all just stayed with the default port number. If two of you pick the same number, you could still get this error message, but it can be fixed by choosing a different port number and re-running the server. (Note that the firewall running on the Linux systems blocks incoming connections on all but a small number of ports; the range from 5001 to 5100 has been enabled for use by our class. If you choose a port number outside this range, your client will fail with a “host unreachable” message.

**The Echo Server**

Most echo servers “echo” whatever they receive back to the sender. This particular server instead prints whatever it receives to standard output. It should prove useful for debugging your client. The command line syntax is
Notes
You will need the file cs332_proj1.tar, which you will find in my Netfiles outbox (netID lbarnett, look in the cs332 folder).

You should use TCP (SOCK_STREAM) sockets for your client. This means that you aren’t guaranteed that message boundaries will be preserved. IP, the underlying protocol, could fragment a message as it traverses the network, so even though the server might send a complete string, your client may receive the message in several pieces, so a single call to recv( ) won’t be sufficient, you’ll need to loop and test for the receipt of the newline character which terminates all messages from the server.

You should compile your source code using the gcc compiler (or g++ if you decide to use C++). In order to link the socket libraries, you must use the “-lnsl” options when you compile. You should write a makefile that includes these details so that you don’t forget them. When I test your program, I should be able to untar your files and type “make” and get the executables I need to test.

You can test code on any machines you like, though the final version must work correctly with the server running on mathcs02 (IP address 141.166.207.129) and the client on mathcs03 (IP address 141.166.207.130). Since these are Linux machines, it is probably best to write, compile and run your code on Linux machines to avoid last minute compatibility problems.

Substantially similar projects have been assigned in previous semesters in the Networks class. It is an Honor violation to use code written by another student or distributed by a faculty member for those assignments in any way in the completion of your solution to this project.

How to kill a server that won't respond: The first thing to try is typing Ctrl-c in the window where you ran the server. If that doesn’t work, pop up another terminal window and type the command “ps -ef | grep <your user name>”. This should show you all processes running on the current machine that belong to you. Pick out the line with the server name and look at the leftmost number on the line. This is the process ID for the server. Now, type “kill -9 <processID>”, which should terminate the process regardless of what it’s doing.

Turning in your program
You should create a tar archive of all of the files I will need to compile your program, including the makefile (you don’t need to send back either of the executables I’m providing for you), giving the tar archive a name that will be unique, like barnett_proj1.tar, and drop the tar archive in my Netfiles inbox (my user ID is lbarnett). You may resubmit before the deadline if you discover errors. Be sure to send me an email if you do this.