Project 1: Implement a simplified version of netcat

Overview

For this project, you will implement a simplified version of the netcat (or nc) utility. Netcat is a simple Unix utility which reads and writes data across network connections, using TCP or UDP protocol. In the simplest usage, "nc host port" creates a TCP connection to the given port on the given target host. Your standard input is then sent to the host, and anything that comes back across the connection is sent to your standard output.

Internally, netcat uses the BSD socket interface to listen for connections, establish connections, send and receive data, and terminate connections. The implementation details of TCP/UDP, IP, and Ethernet---for example, constructing packet headers, performing route table lookups, and translating IP addresses to MAC addresses---are handled by the operating system.

If you have any other questions, please ask them on Piazza.

Submission Requirements

Your submission should include the following programs and files

Source Code
Makefile
README

You should write your implementation in C and use the functions in the BSD socket interface to establish/terminate connections and send/receive data. You should call your compiled program `snc` (which stands for simplified netcat) i.e., you should provide a Makefile that compiles your source code to produce a binary called `snc`

Your README file should contain a short header containing your full name, CNetId. The README should additionally contain a short description of your code, tasks accomplished, any issues you encountered, and how to compile, execute, and interpret the output of your programs.

Put the README, the Makefile, and all source files in a directory named with your CNetID, then tar the directory by running:

```
tar czvf CNetID.tgz CNetID
```

Submit your file to dropbox upload using the following URL.

https://www.dropbox.com/request/mpbvOzusHIhLjSdXbkOh?oref=e

Implementing Simplified netcat

You should write your implementation in C and use the functions in the BSD socket interface to establish/terminate connections and send/receive data.

Options:

The command line syntax for your simplified netcat is:

```
snc [-l] [-u] [hostname] port
```

Your simplified version of netcat should appropriately handle the following arguments:

`-l`

Used to specify that netcat should listen for an incoming connection rather than initiate a connection. In other words, it specifies that netcat should behalf as a server rather than a client.

`-u`

Indicates that UDP should be used instead of TCP.

`hostname`

If the `-l` option is given, hostname specifies the IPv4 address in dotted decimal notation (e.g., `128.105.112.1`) or the symbolic hostname (e.g., `sandlab.cs.uchicago.edu`) on which the server instance should listen for packets/connections.
If the `-l` option is **not** given, then hostname specifies the IPv4 address or the symbolic hostname that should be used by the client instance to contact the server instance.
If the `-l` option is **not** given, the hostname argument is required; otherwise it is optional.
The hostname argument must be the second to last argument, if it is included.

`port`

If the `-l` option is given, port specifies the port on which the server instance should listen for packets/connections.
If the `-l` option is not given, then port specifies the numeric port number (e.g., 9999) on which the client instance should contact the server instance.
The port argument is always required; it should be the last argument.
Note: Port numbers less than 1024 are generally reserved for common applications, so you should pick any port number between 1025 and 65535 (assuming no other application on the machine is already using it).

Behavior

Your implementation of netcat should behave as described above.

> If the `-l` option is specified, you should wait for incoming connections on the specified port. Otherwise, you should initiate a connection on the specified port.
> Once connected, your client should read characters from stdin and, when the enter key is pressed, send the text buffer over the connection. When data is received to the server, you should immediately output the text on stdout. **Bonus credit: for simultaneous reads/writes you will need two threads to achieve this: one thread to read from stdin and send data and one thread to read from the socket and output received data.**
> When you are using TCP and Ctrl+D is pressed, you should close the connection and exit.
> If the opposite side closes the connection, then you should exit.
> When you are using UDP and Ctrl+D is pressed, you should stop reading input from stdin and sending over the socket. You should still continue to receive from the socket and output (on stdout). You do not need to exit the program when Ctrl+D is pressed and UDP is being used.

Errors

You program should be able to gracefully handle errors encountered.
If the user specifies an option that does not exist or a required option is not provided, then output:

```
invalid or missing options
usage: snc [-l] [-u] [hostname] port
```

If you encounter any other error conditions, then output:

```
internal error
```