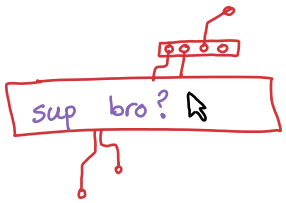
Lab 5 - Let's chat



CC3501

Task Description

Build a network chat application to enable you to send text messages to your classmates. Your system will operate by sending UDP broadcast packets on the local network. Therefore, you will be able to chat with other users on the same wifi network.

Your software must be a command line Linux application that prints messages to standard output (the terminal). The user will type their chat message into the terminal. Once your program receives a line of keyboard input, it will prepare the UDP data packet and transmit it to the network. Similarly, once your program receives a UDP data packet, it will interpret it as an incoming chat message and print it to the terminal.

dont clone Bronsons repo. wont need all the exe files

X TCP
X HTTP
edit CMAKE lists

Outline of the program

An overview of the program you need to write is as follows:

```
Use getaddrinfo() to produce the address data structures for a UDP socket.
Set ai_socktype = SOCK_DGRAM. The IP address is the broadcast address of your
local Wifi network.

Use socket() to create the socket.
Use setsockopt() to set SO_BROADCAST = 1.
Use setsockopt() to set SO_REUSEPORT = 1.
Bind socket to the address returned by getaddrinfo().
Prepare a 'pollfd' array containing standard input (STDIN_FILENO) and the socket.

Loop forever:
    Use poll() to wait for events on stdin or the socket.

    If there was an event on stdin:
        Read from stdin.
        Prepare the message to transmit.
        Send the message on the socket using the sendto() command.

    If there was an event on the socket:
        Read the packet into a buffer.
        Print the received username and message to the terminal.
```

my_socket

already in the structure

Data format

Messages on the wire are transmitted as regular ASCII/UTF8 text.

The data format is as follows:

Field	Remarks
Username	Variable length string of maximum length 32 bytes
Null separator	Single byte of 0
Message	Variable length string of maximum length 240 bytes

To prepare this packet, allocate a buffer of appropriate size (e.g. the maximum possible length). Then use functions like 'memcpy' and 'strncpy' to copy strings into the buffer. Note that you can use pointer arithmetic to get a pointer to the start of the message, i.e. buffer location + length of username + length of null.

Your program should be robust against malformed messages being received over the network. It is likely that at least one person in the room will make a buggy version of their software that transmits something in violation of this format. Therefore you must take care to handle potentially corrupted incoming messages.

Hints

- You can copy the project structure (including CMakeLists.txt) from a different project and modify it accordingly. For example, consider starting with the code in

  `https://github.com/bronsonp/SocketsDemo`

  In particular, the udp_receive.cpp example uses the poll API to monitor the socket. It can be extended to monitor both the socket and the standard input ("STDIN_FILENO").

- The username (your name) should be either hardcoded into your source code or specified by the user when they launch your program.

- The message is what the user types into the terminal.

- By default, the terminal operates in "buffered mode", where characters are not delivered to your application until the user has typed a complete line and pressed enter. Therefore, when you read characters from standard input, you will read an entire line at a time.

- To save you from doing bitwise calculation to determine the broadcast address, you can read it from the "ip addr" command. It is labelled "brd" in the row where the Wifi interface's IPv4 address is shown.

Assessment

To complete this lab, demonstrate a working chat application.

Extension exercises

- Other people may include newlines and other special characters in their usernames or messages. Consider stripping out newlines for a nicer display.
- Allow the username to be specified on start-up.