CSCI 4220 Lab 3

Lab 3: select and Reverse Echo Server/Client

In this lab you will practice using `select()` and will be reversing the roles of server and client from the echo server application we have previously explored. You may want to make use of **unp.h** since it has many common includes. You may also want to use some of the book code as a starting point for your solution. **We strongly encourage working in pairs for labs going forward.**

This lab will have you reverse the usual roles of the echo "client" and "server" - the client will still connect to the server, but it will be the server that reads from stdin and sends the message to the client, and the client will send the message back to the server. Please carefully read the description below before beginning.

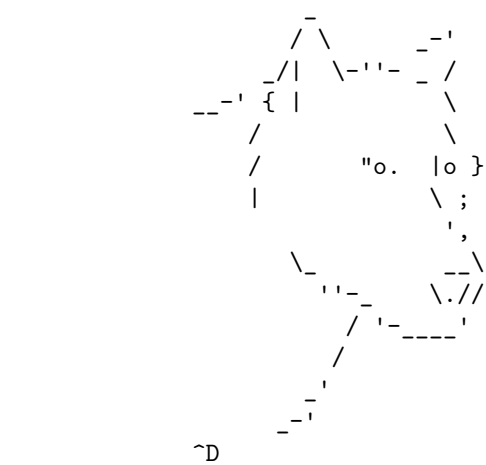
Write a pair of C programs **lab3_server.c** and **lab3_client.c** which are an echo server and client respectively. The server can support up to 1 client at once. The server should read from standard input, and whenever a string is provided, it should send that string to the connected client. If the server reads an end-of-file, it should close all connections and terminate. The server should take 1 argument from standard input, a number. This number should be added to 9877 and used as the port number for the server.

The client should support connecting up to 5 copies of the server on different ports. Whenever a message is received from a server, the port number and the message should be printed to standard output. Your client should use `select()` instead of blocking on `read()`. If the user enters a port number via `stdin`, and the client has less than 5 connections in use, it should attempt to connect to `127.0.0.1` on that port. Any input provided from `stdin` while all connections are in use should be ignored. If a server closes the connection, the client should print a message stating the connection was closed and include the port number of the server. The client should not exit if a server closes.

On the following two pages are an example of what one server followed by another server might look like, with the same client being used for both. Due to how complicated displaying interleaved output is, there is not an example of the client simultaneously connected to multiple servers, but you should be prepared to demonstrate your code works in such a situation.

Example server input (lines starting with > are output):

```
???@???:~/Teaching/NetProgF18/longlabs$ ./server.out 1
>Accepted connection
The wolf says,
    "Awooooooo~"
-A wolf
```



```
^D
>Shutting down due to EOF
???@???:~/Teaching/NetProgF18/longlabs$ ./server.out 3
>Accepted connection
This is another connection
>str_cli: client disconnected
```

Example client output (lines with > are output):

```
???@???:~/Teaching/NetProgF18/longlabs$ ./client.out
9878
>9878 The wolf says,
>9878     "Awooooooo~"
>9878 -A wolf
>9878          _
>9878        / \      _-'
>9878      _/|  \-''- _ /
>9878 __-' { |          \
>9878     /             \
>9878     /       "o.  |o }
>9878     |            \ ;
>9878                   ',
>9878        \_          __\
>9878          ''-_    \.//
>9878            / '-____'
>9878           /
>9878         _'
>9878       _-'
>Server on 9878 closed
9880
>9880 This is another connection
^C
???@???:~/Teaching/NetProgF18/longlabs$
```