CSCI 4220 Lab 3

Lab 3: TCP Message Board with `select()`

In this lab you will practice using **TCP sockets** and `select()` by implementing a simple multi-client **Message Board** application. You may want to review the code we discussed in class, such as tcpservselect01.c and strcliselect01.c. But this exercise focuses on building a small real-world style service where clients can post and retrieve messages.

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. In this assignment, you **cannot** use `fork()`.

Requirements

Server

- Listens on port `9877 + <offset>` (e.g., offset 1 → port 9878).
- Accepts up to **5 simultaneous clients**.
  - If a new connection arrives when the server already has 5 active clients, the server should:
    - Reject the connection (close the new socket).
    - Print:
      `Too many clients, rejecting connection.`
- Maintains a shared board of the **last 10 messages**.
- Supports two commands from clients:
  - `POST <message>` → adds message to the board, **broadcasts it to all other clients**. The posting client won't see its own message echoed back.
  - `GET` → sends the current contents of the board.
- Uses `select()` to multiplex input from all sockets and server stdin.
- On EOF (`^D`) from stdin, closes all sockets and terminates.
  - Print:
    `Shutting down server due to EOF.`
  - Close all client connections and the listening socket.
  - Terminate the program cleanly.

Client

- Connects to the server at `127.0.0.1` on a given port.
- Uses `select()` to multiplex stdin and the socket.
- Allows user to enter commands (`POST ...` or `GET`) interactively.
- Prints any messages received from the server.
- If the server closes the connection, prints `Server closed connection` and exits.

Some Output Examples

Server: (a client is terminated)

```
$ ./server.out 1
TCP server listening on port 9878
New client connected.
Client disconnected.
Shutting down server due to EOF.
```

Client: (the server gets EOF)

```
$ ./client.out 9878
Connected to server on port 9878
POST Hello world!
GET
Hello world!
Server closed connection
```

TCP Protocol Analysis

In addition to your implementation, you must capture and analyze the **TCP Three-Way Handshake** when a client connects.
We will discuss this in class on the lab day.

1. Run Wireshark or `tcpdump -i lo port <server_port> -w handshake.pcap` during a client connection attempt. If you use tcpdump,

- Replace lo with your actual loopback interface name (use ifconfig to find it out).
- Replace `<server_port>` with your actual port (e.g., 9878).
- This will produce a raw trace file (`handshake.pcap`).

2. Identify the three packets.
(For tcpdump, you can check https://amits-notes.readthedocs.io/en/latest/networking/tcpdump.html#flags):

- `SYN` (from client)
- `SYN-ACK` (from server)
- `ACK` (from client)

3. For each of these packets, record the **absolute sequence number** and **acknowledgment number** shown in the trace.

- `tcpdump -r handshake.pcap -S` The -S option is used to show absolute numbers.
- If you use **Wireshark**, you must disable relative numbering:
  - Go to **Preferences → Protocols → TCP**.
  - Uncheck **"Relative sequence numbers and window scaling"**.
- We will not accept relative sequence numbers starting from 0.

4. Submit **two things**:

- The raw trace file (`handshake.pcap` or `.pcapng`) containing the handshake packets.
- A `README.txt` listing the three handshake packets with their **flags, sequence numbers, and acknowledgment numbers** in the following format:

```
SYN     Seq=# Ack=#
SYN-ACK Seq=#  Ack=#
ACK     Seq=# Ack=#
```

Submission:

- `server.c` and `client.c`
- `Makefile` with targets `server` and `client`
- `README.txt` listing the handshake packets
- Raw packet capture file (`handshake.pcap` or `.pcapng`)