Writing a Custom TCP Application with Sockets

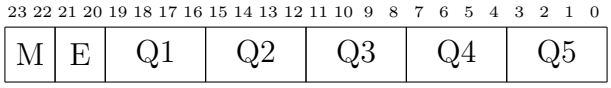
For this assignment, you will be writing a custom TCP application similar to HTTP. This application will be an Application Programming Interface (API) for the same fictitious bakery we saw in Assignment 1. We will be using the `asio` library to handle all the networking code for us. This is significantly easier than trying to do it all on our own! However, it does add an extra dependency to the project (in the `third_party` folder), which will increase compile time. I have tested this on both Mac and Ubuntu. If you run into issues compiling the default example, please let me know as soon as possible. Please go to https://github.com/BradMcDanel/cps373-assignments/tree/main/02-sockets to download (or git pull) the starter code for this assignment.

You will need to implement two programs: a client (`src/client.cpp`) and a server (`src/server.cpp`). Each of these programs will have their own `main` function and be run in separate terminals. The server is responsible for storing all the bakery data and handling client requests. The server application is **always running** much like in HTTP. Clients can query the server at any time to ask specific questions about the bakery data or add new orders.

Request and Response Message Formats

The API defines what the client is allowed to ask the server to do. Since the application is simple, it has a small 3-byte fixed-size request message shown below:

Request Message Format



M is a 2-bit message type, corresponding to 4 possible messages given shortly, E is a 2-bit employee index (we only have 3 employees), Q1 is a 4-bit quantity corresponding to item 0 (Biscuit). Q2 through Q5 are similar to Q1 and represent item quantities for Bun, Brownie, White Loaf, and Wheat Loaf, respectively. The four message types in the API are:

- `M=00`: Get total number of orders.
- `M=01`: Get number of orders for employee E.
- `M=10`: Add an order using E, Q1, Q2, Q3, Q4, and Q5.
- `M=11`: Unused. Do nothing.

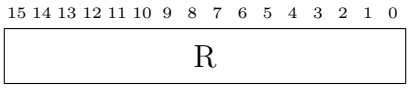
The employee mapping is as follows:

- `E=00`: Brad
- `E=01`: Claudia

- `E=10`: Simone
- `E=11`: Unused. Invalid!

You will also need to implement a 2-byte response message to send answers to the questions back to the client. It's format is even simpler:

Response Message Format



Here, R is a unsigned 16-bit integer (unsigned short), which stores the number of orders for `M=00` or `M=01`. In the case of `M=10`, the bits of R should all be 0.

Server Application

Beyond implementing the request and response message formats, you will also need to implement logic on the server to extract the number of orders from the Bakery object or add a new order to the Bakery object. For instance, if `M=00` in a request, the server would use the current size of the `orders` vector as the total number of orders in the Bakery object. Thus, a large portion of the work for this assignment relates to extracting bits from the request message format. For instance, I would start by taking the 23rd and 22nd bit from a message to determine the message type.

Start the server by reading from a serialized file (either text or binary - up to you) to populate the Bakery object. In the provided code, I used the text deserializer from assignment 1. In a real application, this Bakery object would likely be generated from a database query. After this, the server waits in an infinite while loop for client requests to come in.

Client Application

The client application sends requests to the server (following the response message format) and outputs response messages to the terminal. You do not need to build up a terminal user interface to create custom messages. Instead, you will just hard-code a series of requests/responses into client.cpp. Your finished client application should send the following messages below to the server in order. Please print each response message to the terminal.

- Get the number of orders for `E=01` (Claudia)
- Add an order: `E=01`, `Q1=2`, `Q2=5`, `Q3=0`, `Q4=1`, `Q5=0`
- Get the number of orders for `E=01` (should have increased by one!)
- Get the total number of orders

Submission

When you are finished, zip up your project (excluding the `build/` directory) into `assignment2.zip` and submit via Canvas.