CS60, Lab 1: A Custom TCP Protocol Client via Berkeley Sockets

Sergey Bratus, Spring 2017

Date due: Solutions to this lab will be due on Thursday April 13 before class.

Platform: Your solutions must compile and run on CS Unix systems.

Submission: Your solutions must be submitted by checking them into the CS department LabGit system at `https://gitlab.cs.dartmouth.edu/`. Create an account for yourself by navigating to that page and following instructions; then create a project called `cs60`. Create a directory in it called `lab1` and work there. Give access to your project to your section's grader and TAs (they must be able to see your code to grade it!)

There is a server running at the host `cs60.cs.dartmouth.edu`, on TCP port 5050. When queried according to the protocol described below, it returns information about a US state by its US postal two-letter code.

Write a C program that sends properly formatted queries and displays the returned information.

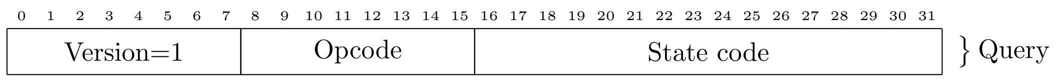
Example output (for the meaning of constants, see protocol specification below):

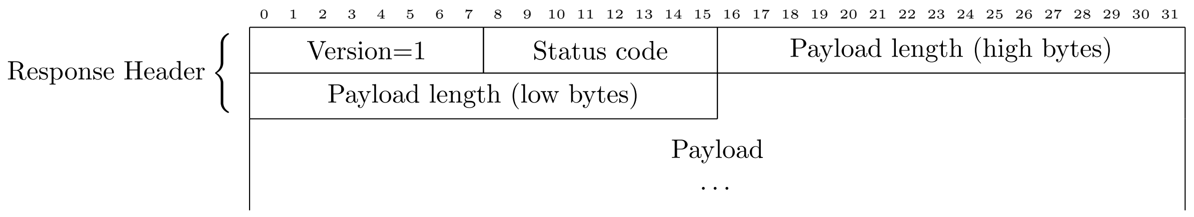
```
$ ./tcpcli nh 1
New Hampshire
$ ./tcpcli nh 2
Concord
$ ./tcpcli nh 4
Live free or die
$ ./tcpcli tx 1
Texas
$ ./tcpcli tx 2
Austin
$ ./tcpcli tx 3
December 29, 1845
$ ./tcpcli ab 3
Error received: No such state: ab
$ ./tcpcli tx 6
Error received: No such opcode: 6
```

(In this example, the address and port of the server are hard-coded in the program. This cannot be allowed in a real-world tool, but will save you some coding around C strings.)

The protocol: The following opcodes are implemented: 1 for the name of the state, 2 for its capital, 3 for the date when it officially became a state, 4 for the state motto, and 5 for the state flag (as a GIF).

The protocol (version 1) consists of queries and responses, one per connection:





Important: Your code should not crash no matter what response(s) it receives, even illegal ones! The server may be evil, or its response(s) could be spoofed. We will test your code with evil invalid responses.

Notes:

- You will find some captured exchanges between a client and a server in `sample-queries.pcap`. Find out from this capture which status codes are legal and what they mean. Handle the responses accordingly.
- The byte order of the 4-byte *Payload length* field is big-endian, the standard network order for multi-byte integer fields.
- Note that the 4-byte *Payload length* field is not aligned at a 4-byte boundary. This is very rare in actual protocols, and is not good design practice!

  Most compilers, seeing two `char`s followed by an `unsigned int` in a struct definition, will quietly insert two unused padding bytes when compiling this struct, so that the 4-byte integer would be aligned properly, at 4-byte boundary, counting from the struct's beginning. This will break processing packets, which have no extra filler bytes.

  So in order to parse this protocol correctly, you need to tell the compiler to not insert the padding. If you want to use an `int` in your structure definition, you'll have to use GCC's `__attribute__((packed))`.
- Although most opcodes bring back a short response, the state flag GIF can be quite large. You will need to *malloc* the space for it. Be careful: many protocol implementations introduced bugs at that point!

Terms and conditions: You are allowed to use any externals materials, printed or electronic. You are allowed to discuss problems and technical/C tricks, but the code you submit must be your own: you are not allowed to copy solutions from other students. Abide by the Honor Code; if in doubt, ask.

Late submissions: Throughout the course, you will get *two* free extensions for a late submission, each of 48 hours after a deadline. Once you've used these up, you will lose 10% points of credit for each late assignment, for each late day.