Project 1. Socket programming practice

Yongun Seong 2017-19937

For this project, I implemented a basic HTTP/1.0-like client and server. Both components heavily depend on my buffered I/O module, which made reading CRLF-delimited messages and fixed-length response bodies straightforward.

My server is able to handle multiple concurrent connections by using nonblocking io and `epoll(7)`. In particular, I used a single process with level-triggered `epoll` to handle multiple (up to 1000, as tested) concurrent clients. I believe my implementation could be be trivally extended to use multiple processes, given that it is already setting `SO_REUSEPORT` on the listening socket, though I did not test this. Once the number of concurrent clients reaches 1000, in order to stay below the default limits on open file descriptors, further connections are delayed until existing connections are closed. The client, on the other hand, uses blocking I/O, as it only needs to manage a single connection.

While implementing each component, I tested them using ad-hoc commands at first, like `echo -ne 'message...' | nc -p 8080 & ./client -s localhost -p 8080`. Once I was finished implementing the basic functionality, I wrote extensive tests for various edge cases and invalid messages for both components. As I was already familiar with Go, and given that Go made writing concurrent, networked applications relatively easy, I implemented by test suite in Go.

Using my test suite, I noticed a few discrepancies between my code and the provided reference servers:

First, the reference servers seem to have a bug where the server may not write any response when the client writes extraneous data after the body; that is, `echo -ne 'POST message SIMPLE/1.0\r\nhost: localhost\r\ncontent-length:1\r\n\r\naa' | nc -v localhost 8080` prints nothing at all. My server implementation differs from the reference in that it returns a response after reading the correct request length, and ignores the extraneous data.

Secondly, the reference server hangs when a header includes the NULL (`\0`) character. My server differs from the reference by returning a `400 Bad Request` response in such cases.

Third, I noticed that the reference server returns `400 Bad Request` responses on zero-length requests. My server returns `200 OK` with a zero-length body. My server does implement other validation on the content length, and ensures that the the request is between 0 and 10 MB long.

There are likely many other cases where my client or server behave differently from the reference servers, as the HTTP/1.0 spec allows for much room for interpretation.

Collaborators

- Minhyeok Jeon (2018-10727): Discussion on testing and implementation strategies.