CMPUT 379 Assignment 3

name: Waridh 'Bach' Wongwandanee
ccid: waridh
sid: 1603722

Objectives

From my point of view, this assignment was assigned in order to improve our understanding of sockets, TCP protocols, sending structs through low level I/O, and structuring a good header/message format. Some topics from the previous assignments were also covered, such as I/O multiplexing.

Design Overview

- Used unix socket(), bind(), listen(), accept(), connect(), and associate data types to establish connection between servers and clients.
- Used poll for I/O multiplexing, checking each file descriptors for activity and then dealing with them appropriately.
- Created and used a structure to contain and send messages that have informative headers. The struct has the client id, amount of lines in the message, the type of message, and the name of the object that data was sourced from.
- Parsed the input string such that instead of sending the whole line, our implementation will just send the content that is stored after the name of the object. This way, we don't have white spaces in front of the content, nor the name of the file.
- More robust error handling for failed functions.
- Stores objects that were put, and can differentiate between owners.
- Follows the requirements set out by the assignment
- Some extra error handling when deleting objects that don't exist. The server will return an error saying that the object doesn't exist.
- The server will immdiately close file descriptors when the client disconnects from it. This way, a new client with the same id can connect. There is also a little error handling that rejects clients with the same id from connecting at the same time, forcing the other client to reconnect at a different time. It's a design choice.
- At the time of quit for the server, we close connection to all clients. The server doesn't really say which file descrtiptors gets closed, since I think it looks ugly.
- The makefile and the provided test file are created by me. The makefile should follow all command requirements plus some irrelavent testing targets. The testing file is extremly similar to the given example, but with extra commands.
- If an object being put into the server has more than 3 lines of content, the client will terminate itself.

Project Status

The project is complete. Everything works as intented by my testing. I made sure that both clients can connect concurrently and interact with each other. I checked and compared the output with the example. I tried breaking the program by connecting with the same client id. I sent objects with no content lines. Everything seems to work as intended.

The difficulty that was encountered during programming was when I was still using strings as packets. In that implementation, the issue was that sending multiple strings causes some strings to not get received. To solve this issue, I moved to using structs, like given in the example code.

Testing and Results

As said above, I tested using a modified version of the original example testing file. This version gets all errors to trigger if you run it twice on the same client. It sees when there isn't something to delete, deleting something that doesn't belong to the client, getting something that doesn't exist, putting something that is already there, putting successfully, getting successfully, and we run list and quit to check the server. Everything seems to work as intended, so we are happy about that.

Acknowledgments

I'd like the thank the example file and geeksforgeeks, as I did this entire thing using that. No collaboration, and no textbook (unless you count the examples that shows up on slides and previous assignments).