Brent Yelle

CSCI 3240

Project 3 Documentation

Problem-Solving Approach

Using the echo server that was provided in class and on the lecture slides as a base, I first wrote the server-side program in `server.c`, beginning with the prompt that asks the user for input. I tried to "delegate" as many repeated tasks as possible to sub-programs for readability and writability, which I found very helpful.

I considered having the input processing lead into a switch statement, but I argued with myself over the right way to have the client-server interaction loop, and I ultimately decided on a do-while loop:

```
do {
	userchoice = PromptUserChoice(connfd);
	if (userchoice == 1) {		//adding a new record
		AddRecordService(connfd);
	}
	else if (userchoice == 2) {	//reading for a record
		ReadRecordService(connfd);
	}
} while (userchoice != 3);		// closing the connection
```

This allowed the closing-connection process (triggered by `userchoice == 3`) to be placed simply after the conclusion of this loop.

As I was unable to figure out how to send an end-of-file signal to use `rio_readnb()` properly, I instead had to rely on multiple calls to `rio_readlineb()` instead, one for each newline in the messages sent over. This is extremely "janky," as the kids would say these days, and definitely not my preferred way of doing things, but in the interest of turning in a working project, I bit the bullet and did it this way.

Data Structures

I defined one struct type for use in this project, titled `Student`, which is used server-side to more neatly access and pass around the data for the students:

```
struct Student {
    char firstname[NAMELEN];
    char lastname[NAMELEN];
    int age;
    char major[NAMELEN];
};
```

…where `NAMELEN` is defined in `server.c` to be 50.

Aside from that, I made use of the `rio_t` buffer for reading messages sent between the server and client.

Algorithms

All searching through of student data is done by linear search, as the data is unsorted, and I did not want to have to deal with sorting it. Given the small number of students (less than 10), this is not a problem for efficiency.

User-Defined Functions

The server.c code defines 12 functions:

PromptUserChoice(…)

PrintOptions(…)

RePrintOptions(…)

ReadUserChoice(…)

AddRecordService(…)

GetRecordToAdd(…)

AcknowledgeAdd(…)

ReadRecordService(…)

PrintRecord(…)

PrintNoRecord(…)

GetRecordToFind(…)

PrintGoodbye(…)