

## ECE 4220 Project Report

### Remote Faircom Database Login/Editing

#### Abstract:

My project was to create a method for a person or people to be able to remotely edit their own section of a table on the Faircom Database, and to do so across a client/server socket connection.

#### Introduction:

For my project, I wanted to use a client/server socket communication in order to make changes to a remote file system. The objective was that a person could use a user space program on one TS-7250 board in order to connect to a database table on the Faircom Database that was running on a separate TS-7250 board. I believed that this was a good way to better reinforce socket communication, as well as be a good introduction to using the Faircom Database. Since I had never really had any databases experience before, it seemed like a good challenge to learn. The project idea was inspired by remote banking applications, since I was thinking of how I could do a project that wouldn't cost me much money since I don't have any, and as I was checking my banking application, I realized I could try to implement a mock money transfer system, where the user could change their own database information if I let them do so.

#### Background:

As the world advances towards non-physical money sources, such as debit and credit cards, more and more people will cease to carry cash as their primary method of spending. And in the modern age, convenience is key, so checkbook balancing is rarely practiced it seems, thus many people use online banking. Common approaches to online banking include having users log in to a secured site, and viewing their information privately. Several banking institutions also will allow their customers to wirelessly transfer money to one another, so long as they have previously approved it, and the transfer is performed by the person giving the money.

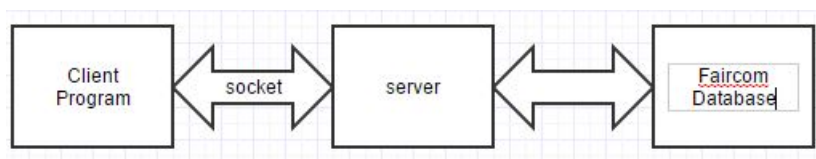
#### Proposed Method:

The method that I implemented was to first start the Faircom Database server program that was located in the file that I downloaded from Faircom. After I started running that, I started my own user space program, which was a server program that would operate as a go-between for the client and the Faircom database. When the server program was started, it initialized and defined the table that would be used from the Faircom database, and then it would wait to receive a message from a client program on the same port, which would indicate that it could start executing the bulk of its program, instead of just waiting. I used a single connectionless UDP

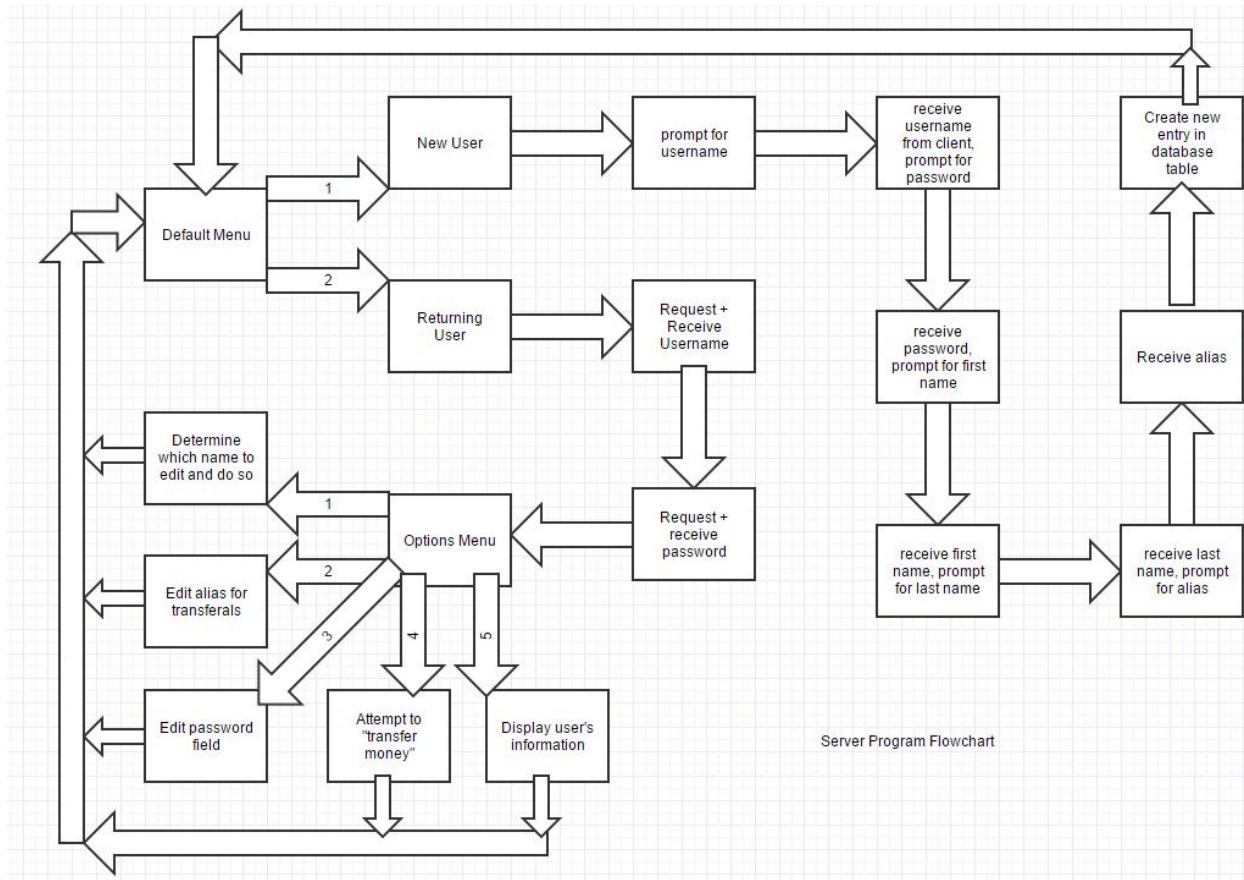
setup at first because I tried to get the TCP server working for multiple users and it didn't want to have anything connect to it, so I switched to a modification of the code that we used for labs 5 and 6 in order to make sure I could test the database functions with information received across the socket. The server program would begin by defaulting to asking the user in a message sent across the socket if they were a new user or a returning user. The client would then reply with whether the user wanted to make a new entry, or deal with an existing entry, and the server would begin to implement the proper routine to handle that case. If the user was requesting to create a new entry, the server would begin to prompt the user to enter all relevant information (username, password, first name, last name, and an alias), and then would call a function that I wrote named "Add\_Single\_Entry." Add\_Single\_Entry is a function that returns void, takes in a CTSTRING parameter for each field of the table, and then is supposed to navigate to the final entry of the table and append a new piece of data with those fields filled in. After doing this, the program would then return back to the default menu of asking the user if they were new or returning. The difference then would be that the credentials that the user had just created could then be used to access a new chunk of data. At this point, the user could elect to "log in" to the entry that they had just made. In order to do this, the server would send a message to the client asking for a relevant username. The client would then send a message back that would contain a string that the user entered, and would use a function called "find\_user" in order to determine the index that the username that was provided was located at. After finding the index of the user, the server would then use the socket to ask the client to send back the "password" for that index of the table. I then would compare the string that the client sent across the socket, to the "password" for the row. In order to do this, I simply had a field in the table that held the "password" for each of the entries in the table. This was a simple way to do it, though it isn't exactly secure by any means, other than the client program was not set up to show any information for users that the credentials weren't entered for. The client program was to be given three total chances to have the correct password input for the given username. If the user successfully managed to pass a correct username and password combination across the socket to the server program, the user would be allowed to continue. Continuing at this point means that the user is given the choice of editing their first or last name, changing their password, changing their alias that is used for "transferring money", "transferring money" to another person if they can correctly identify another alias, or for the program to print out their database entry. This was done with a secondary menu. If the user chooses to change their names, they are given the choice between changing their first or their last name, and then based upon which decision they make at that point, they will be allowed to enter in a new name, which will then be passed in to a function called "Edit\_Entry", which takes in the user's index in the table, what field they want to edit, and the change they want to make. Changing their first name will change field0, last name changes field1. Once the user successfully makes their desired change, the server program should return to the default menu. If the user had chosen to change passwords, then the server would make the client reply with a new entry for the password for that index of the table, and would use Edit\_Entry to make that change. The same idea holds true for changing the alias for the user to use when receiving transfers from other users. If the user chooses to transfer money to another

user, the server should attempt to find the index of the other user by comparing the entered alias with the known aliases of the other entrants in the table. The program would then prompt the client for an amount to transfer, and make sure that the user had sufficient funds in order to transfer money to the other person, and if they did not, the server would disallow the transfer. Otherwise, the server should then subtract the designated amount from the current user's money, and add the same amount to the person who had the alias that was designated. It was again necessary to use `Edit_Entry()` in order to change the amounts of money each person had. The program would then be finished with this part again and return to the default menu. Had the person gone with the option to print out their entry, the server would then pull out the entry from the database at the index of the user, get information from all of the separate fields, and then pass that information across the socket to the client. This would also be a point where the program was complete and the server would return to the default menu. The user could also enter "q" to close the connection to the database and shutdown the server remotely.

The proposed setup for the client program would be a user space program that loops forever and has essentially two threads, with one thread reading in information from the socket and displaying it on screen, and the other thread sending back user input across the socket. I would like for it to basically be a multi-user TCP configuration client that would connect to the server and then after it connects would immediately begin receiving information. Due to time constraints, I was unable to make the client program at first, especially since I had difficulty getting my server program to connect with TCP anyway, even using the sample code provided to us for creating a multi-user connection. I bypassed this by utilizing the client program that was used in lab 6 in order to communicate with the server, and to indicate connection the first time by sending any generic string. The only problem with this was that after the program had started, it was impossible to connect other individual clients to the program in this way and get it to display the default menu.



Simple view of intercommunication between programs (above)



## Experiments and Results:

In order to test the database functions to ensure that they worked to some capacity, at first I would enter hard-coded strings as input whenever I would need to, and would check to make sure that the desired output had taken place. I then tested the socket communication by connecting a single client program to the same port that the server program was connected to (single client due to me having a connectionless socket) and ensuring that the two ends of the socket were receiving the correct information from the other end. I then went through each step of the program with any steps that I believed that I would want for a user to be able to implement. By this, I mean that I started out by creating a new user, and then I would attempt to make that user “login” by verifying that they entered matching usernames and passwords to what they had created. I then would select the option to print out that user’s information, so that I could tell if the database entry had been successfully inserted with the proper parameters. Once that was done, I would log them in for each of the other changes that they could make, and would change each field that I wanted the user to be able to directly modify, and then would print out the entry again, in order to ensure that the proper changes had been made. I then created another user entry so that I would have a minimum of two entries, which I knew the alias field of each one so that I could then go back through and attempt to “transfer money” from one to the other. I then would shut down my server and restart it to make sure that I could still log in with

one of those users and print out the information that they had stored, thus successfully using the database.

### Discussion and Conclusions:

The results of my project were both exciting to me as well as a little bit disappointing. I was excited that I managed to get the Faircom database functions that I wrote to work with hard-coded information, but I was disappointed in how much issue I had with trying to get the TCP configuration to work. I was also disappointed that whenever I tried to execute the function for transferring money with variables passed in for parameters instead of CTSTRINGS, it didn't want to read in the correct amounts. I didn't manage to fix either one of these issues, due to time constraints. I bypassed the multi-user TCP connection by implementing a UDP connection with a single user, which I knew would work because it was essentially the same thing as what I implemented in labs 5 and 6. I attempted to change the way that I compared the information in the transferal function, but that didn't seem to fix my problem that I was having. As far as the rest of my program, all of the rest of my results seemed encouraging to me, as it did exactly as I expected it to. I wish that I had implemented my own client program though, since that would have allowed me to have a longer buffer read in, which would allow me to print out more information at once, and make my menus actually legible to the user. I was still able to pass the correct information back and forth between the two programs, but at times the buffer I sent to the client would be too long for the lab6 client to display. I did manage to have the client create a new entry into the table, as well as to edit most of the entries, aside from the Balance entry, which I did not allow the client to have the ability to edit that field directly. Despite some of the issues that I had, I feel like I learned more about TCP and UDP connections, due to the fact that I spent some time looking at each type of connection with a hands-on approach, and had to try to determine the advantages and disadvantages of each kind. I feel as if I learned that TCP has the advantage of being able to become a single user connection that will filter out other connections and simply be able to communicate with a server by itself and not get any data crossover, due to the connection based aspect of it. I learned this even though I was unable to figure out how to successfully implement a TCP connection with my code. I learned that the UDP connection has the advantage of being able to have multiple users connect to it at any time, but there is never any indication of a connection being established, thus making it difficult for me to have automatically started my server routines. In addition to learning about TCP/UDP connections, I also feel like I learned quite a bit about the Faircom Database system. Since a large portion of my program was to be able to create new entries into a table in the Faircom system, and to be able to edit specific entries of that table that already existed, I feel as if I learned some of the inner workings of the Faircom database and how to implement more things on it, which could help me a lot in the future whenever I might need to utilize a database in other applications or for other classes. Generally, I'd say that the project for me was successful, given my experience with database systems being very minimal, and aside from wanting to implement a few more things, and attempt to get the program to work better, it worked to my expectations.