Program 1 {October 27, 2014} 37 points

A Location Client and Server
Due: Tuesday, November 11, 2014 at 11:59 p.m.

This assignment emphasizes client-server programming using the TCP protocol. The assignment is to write both a TCP client and a TCP server in C or C++ using Linux socket commands. The Location Client and Location Server must execute on different CCC machines and communicate with each other using TCP.

The Location Client

The Location Client should be designed to take input either as a series of single line commands from standard input or from an input file, ***LClient.txt***. The Location Client provides an interface to the Location Server that sends proper commands and receives all responses from the Location Server and prints them out to the standard output or writes them to a file, ***LClient.log***.

The Location Client should run on any arbitrary CCC Linux machine. The command line for initiating the client is:

./**my_*LClient*** ***LServermachine*** ***LClient.txt***

where

> ***LServermachine*** indicates the logical name for the server machine (e.g., CCCWORK4.wpi.edu).

and

> ***LClient.txt*** indicates that data is to be read from this text file. If this field is not specified, the Location Client reads command line input from standard input (the keyboard).

The Location Client communicates with the Location Server assuming knowledge of a unique "well-known" port. The Location Client accepts and relays to the Location Server the following five commands:

login *name*

Upon receiving **login**, the Location Client establishes a TCP connection for **name** to the Location Server. ***name*** is a non-blank ASCII string with maximum length of 10 characters.

add *id_number first_name last_name location*

where

id_number is a 9-digit identification number.
first_name is a non-blank ASCII string with maximum length of 20 characters.
last_name is a non-blank ASCII string with maximum length of 25 characters.
location is a non-blank character string (30 character max) indicating the person's current location.

remove *id_number*

where

id_number is a 9-digit identification number.

list *start finish*

where

start is a one or two-character ASCII alphabetic character string.
finish is a two-character ASCII alphabetic character string .

quit *end-of-file*

Upon receiving **quit**, the Location Client indicates to the Location Server to close **name**'s connection.

where

end-of-file is an optional argument indicating whether another client script follows in the input stream.

When ***end-of-file*** is specified via the string "**EOF**", once the Location Client receives a response from the Location Server, the Client closes the log file and terminates. When ***end-of-file*** is not specified, another client script follows ***quit***.

The Location Server

The Location Server is started first and waits for a connection request from a **single** Location Client stream. The Location Server maintains an in-memory location database that keeps track of the locations of all the people added to the database by the Location Client. The database is maintained in alphabetical order by ***last_name***. All name strings in the legal commands are **case-insensitive**. **{Note – the data structure implemented is the student's choice, but the suggestion for this assignment is to keep it simple!}**

Location Server Responses

The following define the response actions of the Location Server to each of the valid commands sent via TCP messages by the Location Client:

login

Upon receipt of login, the Location Server returns a **Hello *name*!** message back to the client process. ***name*** is the specified login name.

add

Upon receipt of the **add** command, the server adds the four items as an entry into the location database in the proper location.

The Location Server checks for duplicates. Namely, if a duplicate ***id_number*** is received, the server sends an error message back to the client that indicates the name of the **id_number** already stored in the database. The goal of the Location Server is to maintain the location database in a manner that facilitates listing the locations of people in ***alphabetical*** order by last name. Note, **add** commands with identical first and last names are NOT duplicates if they have unique ***id_numbers***.

The Location Server sends back a copy of ALL the information as an indicator of a successful entry into the Location database.

For simplicity of design assume that the maximum number of entries in the Location database is **100**.

remove

Upon receipt of the **remove** command, the Location Server searches the database for a match on ***id_number***. If the ***id_number*** entry exists in the database for a person, that entry is removed from the Location database and a success message that contains the last and first name of the person removed is sent back to the Location Client. If there is not a match in the database, the server does not modify the database and sends an appropriate error message back to the Location Client.

list

Upon receipt of the **list** command, the Location Server sends back to the Location Client all location entries in the database currently within the range of the **list** limits. Each Location database entry is sent as a **separate** TCP message back to the Location Client. The entries sent back contain all of the entry information for those entries in the database where the ***last_name*** begins with the ***start character string*** and is less than or equal to the ***finish character string***. If the ***start character string*** is a single letter, the finish character string should **NOT** be provided. In this case, the Location Server sends back all entries in the database where the last name matches the single letter string. If neither

start nor ***finish*** are specified, the Location Server returns the complete database in alphabetical order. If ***finish*** is lower in the alphabet than ***start,*** the server returns an error message. If the database currently holds no entries satisfying the range of the specific list request, the Location Server sends back an indication that there are no entries satisfying the list request.

quit

Upon receipt of **quit**, the Location Server sends a response back to the location Client indicating that the TCP connection will be closed for ***name*** and include a count of the number of commands that ***name*** issued to the server. The Location Server prints out a count of the number of TCP packets it sent to ***name***. The Location Server then returns to wait for a new connection triggered by a subsequent login request from the Location Client. The ***end-of-file*** field is optional. If this field contains the text "**EOF**", the Location Server additionally writes out the complete database to the file ***LDatabase.txt*** and sends back to the Location Client a count of the number of clients processed and the total number of TCP packets that the Location Server sent . The server then terminates.

Do not wait for the official test data to work on this assignment. Work with your own test data initially. The Location Client needs to be able to read directly from a test file ***LClient.txt.*** Your client must also write out Location Server responses out to the ***LClient.log*** file.

What to turn in for Program 1

An official test file will be made available a couple of days before the due date. Turn in your assignment using the ***turnin*** program. **You should turn in a tarred file that includes:** all the source programs for ***LClient*** and ***LServer***, a **README** file and a **make** file. **You can optionally also turn in the LClient.log in the tarred file.** The **README** file should include any special directions needed to execute your Location Client and Server on separate CCC machines. **README** should also contain a **clear** indication of the state of your program when you turn it in. This includes current limitations and parts of the assignment that are not working correctly.

Programming Hints

Below is a list of possibly useful C function calls for Program 1.
Note - you will only need a subset of these calls. See also Help Session 1 on the course web page.

For File operations:

fopen(), fgets(), fclose(), feof(), fscanf(),fprintf(), fputs()

For String operations:

strcmp()/strncmp(), strcpy(), strtok(), sprintf(), strstr(),
strcasecmp()/strncasecmp()... /*case insensitive version of strcmp */
toupper() ..... /* converts a character to uppercase */
isupper() ..... /* checks for an uppercase letter. */

For Memory operations:

malloc()/free(), memcpy()/memncpy(), bzero(), memset().

If your program is written in C++ you will obviously need a different set of system calls for your program.