## CS3516 Computer Networks B10

## Program 1 30 points

### A Location Client and Server
### Due: Monday, November 8, 2010 at 11:59 p.m.

This assignment introduces 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 Unix socket commands. The Client and Server must execute on different CCC machines and communicate with each other using TCP.

## The Client

The Client should be designed to take input either as single line commands from standard input or from an input file, ***LClient.txt***. The 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 Client should be written to be run on any arbitrary CCC Linux machine. The command line for initiating the client is:

***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 Client reads command line input from standard input.

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

### `login` *name*

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

### `add` *id_number last_name first_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) identifying the current location of the person.

### `remove` *id_number*

where

***id_number*** is a 9-digit identification number.

### `list` *start finish*

where

***start*** is a single ASCII capital letter
***finish*** is a single ASCII capital letter**.**

### `quit` *end-of-file*

Upon receiving quit, the Client indicates to the Location Server to close the connection.

where

***end-of-file*** is an optional indicator to determine whether another client script follows in the input stream.

When ***end-of-file*** is specified via the text '**EOF**', once the 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** 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 client. The database is maintained in alphabetical order by ***last_name***. **{Note – the data structure implemented is the student's choice, but the suggestion for this assignment is to keep it simple!}**

The following define the response actions of the Location Server to each of the valid commands sent as TCP messages by the 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 add, 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. 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.

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 remove, the 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 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 Client.

### list

Upon receipt of list, the Location Server sends back to the Client all location entries within the range of the list limits. Each entry is sent as a **separate** TCP message back to the 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*** and is less than or equal to the ***finish character***. If the database currently holds no entries within this range, the Location Server sends back an indication that there are no entries satisfying the list request. If neither ***start*** nor ***finish*** are specified the Location Server returns the complete list. If ***finish*** is lower in the alphabet than ***start,*** the server returns an error message.

### quit

Upon receipt of quit, the Location Server sends a response back to the Client indicating that the connection will be closed for ***name*** and include a count of the number commands that ***name*** issued to the server. The Location Server then returns to wait for a new connection triggered by a subsequent login request. 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 a count of the number of clients processed. 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 Client needs to be able to read directly from a test file ***LClient.txt.*** Your client must also write out server responses out to the ***LClient.log*** file.

## What to turn in for Program 1

The TA will make an official test file 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:** the two source programs ***Client.c*** and ***LServer.c***, a README file and a ***make*** file. **You can optionally also turn in the LClient.log in the tarred file.** Contact the TA if you need assistance with make files. The README file should include any special directions needed to execute your client and LServer on a ccc machine and 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 function calls for Program 1.
Note - you will only need a subset of these calls.

***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().