Programming Assignment 1 30 pts.

A Location Client and Server

Due: Friday, November 4, 2005 1 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., ccc4.wpi.edu).

and

lclient.txt indicates that data is to be read from this text file. If this field is not specified, the Client will read 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

Upon receiving login, the client establishes a TCP connection to the Location Server.

add *first_name last_name id_number location*

where

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.
id_number is a 9-digit identification number.

location is a non-blank character string (with a 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 letter of the alphabet.
finish is a single letter of the alphabet such that ***finish*** is greater than or equal to ***start.***

`quit` *end-of-file*

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

where

end-of-file is an indicator to determine whether there is another client script in the input stream. When ***end-of-file*** is specified, once the Client receives a response from the Location Server, the Client closes the log file and terminates.

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 alphabetically 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!** message back to the client process.

`add`

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

This simple Location Server does not check for duplicates. If a duplicate id_number is received, the server simply overwrites the last name, first name and location information associated with this id_number. The goal of the Location Server is to maintain the location database in such a manner to facilitate listing the location of people in alphabetical order based on last name.

The Location Server sends back a copy of 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 first and last 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 satisfying 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 beginning with all last names with ***start*** as the first letter up to and including all last names with ***finish*** as the first letter. 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.

quit

Upon receipt of quit, the Location Server sends a response back to the Client indicating that the connection will be closed. 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 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 Assignment 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. Turn in the two source programs ***client.c*** and ***LServer.c***, a README file and a make file. Contact the TA if you need assistance with make files.

Programming Hints

Below is a list of possibly useless 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().