CS3516 Computer Networks A15

Program 1 {September 4, 2015} 42 points

Prototype Test for Medical Examiner Clients and Server
Due: Friday, September 11, 2014 at 11:59 p.m.

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

Medical Examiner Disaster Identification Database (DID) Concept

FEMA needs a database and a client/server paradigm to create and access a repository for information about bodies collected during and after a natural disaster. Authorized FEMA employee clients need the ability to input information such as photos, fingerprints, dental records and tattoos that could facilitate identification of a body recovered after events such as a flood, earthquake or tornado. Acting on behalf of a qualified medical examiner, the FEMA employee can input that a body has been positively identified (i.e., full name is input) or that the identification is unknown (i.e., UNK for first and last name). Each body will have a unique ID code assigned and a location where emergency personnel recovered the body. Family members seeking information about missing loved ones ONLY have the ability to query the database.

Assume FEMA has advertised for contract bids to implement a simplified prototype for this network-based system that includes a simple Disaster identification Database (DID) with a maximum of 100 body entries in a simplified in-memory data structure. Your program submission should convince FEMA to hire your company to implement the full Medical Examiner network Application and database.

Preliminary Medical Examiner (ME) Clients

The prototype network application supports two types of clients (**authorized** and **query**) where authorized clients handle FEMA employee functions for the DID and query clients handle family queries into the DID.

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

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

```
./my_MEClient   MEServermachine   MEClient.txt
```

where

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

and

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

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

ME Client commands

The following two commands identify the beginning and end of a simulated, prototype client:

login *name*

Upon receiving **login**, the ME Client determines first whether the user is authorized to input changes (adds, removes or updates) to the DID or that the user can only issue query requests.

For this prototype assignment, the only two legal names are: "**FEMA"** for an authorized user or "**Query"** to indicate that this user that can only issue query commands. The ME Client is responsible for assuring that each command issued to the ME Server satisfies the proper authorization level. The ME Client establishes a TCP connection for **name** to the ME Server.

quit *end-of-file*

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

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 ME Client receives a response from the ME Server, the ME Client closes the log file and terminates. When ***end-of-file*** is not specified, the ME Client and the ME Server both assume that another client script follows ***quit***. Note – authorization does not run across quit commands.

The following are three commands that only a FEMA user can issue:

add *id_number first_name last_name gender 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.
gender	is a single character string. 'M' indicates a male and 'F' indicates a female.
location	is a non-blank character string (30 character max) indicating the person's current location.

Add inserts a new record into the DID. If the body has been identified both the ***first_name*** and the ***last_name*** must be included in the add command. If the ID of the body is currently unknown then both name fields will contain the string "UNK". All fields are required.

update *id_number first_name last_name gender 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.
gender	is a single character string. 'M' indicates a male and 'F' indicates a female.
location	is a non-blank character string (30 character max) indicating the person's current location.

Update changes information for an existing ***id_number***. All fields are required. Note ID numbers are unique while names are not in the DID.

remove *id_number*

where

id_number	is a 9-digit identification number.

Remove eliminates the record associated with the ***id_number*** from the DID.

The following are the query commands that any user can issue:

find *first_name last_name*

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.

Find is a query command that searches for only one name in the DID.

list *start finish*

where

start	is a one character ASCII alphabetic character string.
finish	is a one character ASCII alphabetic character string .

List is a query command that lists records within a name range.

locate *location*

where

location	is a non-blank character string (30 character max) indicating a possible location in the disaster region.

Locate is a query command to obtain information about all bodies at one location.

The ME Server

The ME Server starts first and waits for a connection request from a **single** ME Client stream. The ME Server maintains an in-memory location database that keeps track of the locations of all the people added to the database by the ME Client. The database is maintained in alphabetical order by ***last_name***. Unidentified bodies are located at the back of the database behind all the identified bodies. 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!}**

ME Server Responses

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

login

Upon receipt of login, the ME 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 ME Server adds the five items as an entry into the location database in the proper location.

The ME Server checks for duplicates. Namely, if a duplicate ***id_number*** is received via an **add** command, the server sends an error message back to the client that indicates the ***id_number*** is already stored in the database. The goal of the ME 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***. Identical names should be stored in the chronological order of the add function.

The Me Server sends back a copy of ALL the information sent as an indicator of a successful entry into the DID.

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

update

Upon receipt of the **update** command, the ME Server replaces the four items associated with an ***id_number*** that already exists in the DID. The ME server returns an error message if the ***id_number*** is not found in the DID.

remove

Upon receipt of the **remove** command, the ME Server searches the database for a match on ***id_number***. If the ***id_number*** entry exists in the database, that entry is removed from the DID and a success message that contains the last and first name of the person removed is sent back to the ME 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 ME Client.

find

Upon receipt of a **find** command, the ME Server searches the database for all entries that match the first and last name. The server returns all the information associated with any matches. Each database entry is sent as a **separate** TCP message back to the ME Client.

list

Upon receipt of the **list** command, the ME Server sends back to the ME Client all DID entries in the database currently within the range of the **list** limits. Each database entry is sent as a **separate** TCP message back to the ME 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***. The ***finish character*** is optional. If only a ***start character*** is specified, the ME Server sends back all entries in the database where the first character of the last name matches the single letter string. If neither ***start*** nor ***finish*** are specified, the ME 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 ME Server sends back an indication that there are no entries satisfying the list request. **List** does not return information for any of the unidentified bodies (UNK).

locate

Upon receipt of the **locate** command, the ME Server sends back to the ME Client all DID entries in the database that currently match the specified ***location*** in alphabetical order. Each database entry is sent as a **separate** TCP message back to the ME Client. This query request does include all the unidentified body information found at the ***location*** indicated. Note, ***location*** is case-insensitive.

quit

Upon receipt of **quit**, the ME Server sends a response back to the ME 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 ME Server prints out a count of the number of TCP packets it sent to ***name***. After closing the connection, the ME Server then returns to wait for a new connection triggered by a subsequent login request from the ME Client. The ***end-of-file*** field is optional. If this field contains the text "**EOF**", the ME Server additionally writes out the complete database to the file ***MEDatabase.txt*** and sends back to the ME Client a count of the number of client scripts processed and the total number of TCP packets that the ME 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 ME Client needs to be able to read directly from a test file ***MEClient.txt.*** Your client must also write out ME Client commands issued and ME Server responses out to the ***MEClient.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 ***MEClient*** and ***MEServer***, a **README** file and a **make** file. **You can optionally also turn in**

the MEClient.log in the tarred file. The **README** file should include any special directions needed to execute your ME 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. Indicate clearly all commands that work completely, partially and not at all!

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()

Note – you can use command line redirection to simplify your file input and output.

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.