Project 1: Building a Phone Directory Service System

1. Description

   - Server

     The server should accept following commands from the clients:

     (a) `connect` "textituser" "textitpassword"
     The server will validate the user with the provided password. If it is not a valid user, the server will close the connection. A special user called *administrator* has the permission to modify the related databases.

     (b) `search` "*name*"
     The server will search the database for the phone number of *name*. If the phone number is found, that phone number is sent back to the client in ASCII format, otherwise string *NOT FOUND* will be sent back instead. This request will be only processed after the client has been validated.
     Only valid users can send this request to the server.

     (c) `store [-f]` "textitname" "textitphone"
     The server will try to save *name*, *phone* pair into the database. If the value is stored successfully the server will send string *OK* to the client, otherwise an error message will be sent back.
     When the *name* already exists in the database, if option `-f` occurs in the command, the server will replace the phone number for *name* with *phone* and string *OK* will be sent back; otherwise, an error message will be returned.
     Only the *administrator* is allowed to execute this command.

     (d) `delete` "textitname"
     The server will try to remove the phone number for *name*. If the request is executed successfully, string *OK* will be sent to the client, otherwise an error message will be sent back.
     Only the *administrator* is allowed to execute this command.

     (e) `adduser` "*user*" "*password*"
     The server will add a new user "*user*" and the "*password*" into the user database.
     Only the *administrator* is allowed to execute this command.

     (f) `removeuser` "*user*"
     The server will remove the a user specified in the request.
     Only the *administrator* is allowed to execute this command.

     (g) `bye`
     The server will close the connection.

   - Client

     The client program displays a command menu for the end user.

     ```
     1. Create a connection
     2. Look up a phone number
     3. Add a new name with a phone number
     ```

```
4. Remove a name from the phone book
5. Add a new user
6. Remove a new user
7. Close the connection
```

Some comments:

(a) Create a connection
Prompt the user to enter the domain name of the server, user name, password and the optional port number. If the port number is not provided, a default port number will be used. To avoid port number conflicts, you are recommended use 5000 plus the last 4 digits of your SSN as your default port number.
This will try to create a TCP connection to the server.

(b) Look up a phone number
Prompt the user to enter the name of a person whose phone number will be requested. Only those requests from valid users will be processed.

(c) Add a new name/phone number
Prompt the user to enter the a person's name and his/her phone number. This name/phone pair will be saved in the database. This can only be done by the administrator.

(d) Adding and removing a user can only be done by the administrator.

- Some Requirements on the Server
  - The server should use I/O multiplexing (`select`) to handle multiple clients.
  - The server should handle as many errors as possible in the client's requests. For example, the client may tell the server it have sent the server a 20-byte command string. However, only 16 bytes are sent. The server should not be suspended for the remaining 4 bytes.
  - The server should close the connection in one of the following conditions:
    * If the client has been idle for some time. You may specify the time in the server command line with the new option `-t time`. If no time is specified, the server will the default time which is 5 minutes;
    * The time that the client sent bad requests to the server exceeds a specified number. You may specify this number in the server command line with the new option `-e enum`. If no such number is specified, use 5 as its default.
  - All the blocking system calls for data transfer should be checked for interrupts.

2. Data Transfer

- Request/Result Format

```
---------------------------------------------------
| size  |  request/result strings                 |
---------------------------------------------------
size: 32 bit integer stored in Network Byte Order, the size of
      the request/result string
```

- Sending the Request

```
len = htonl (request_length);
write (sock, (void *) &len, 4);
write (sock, request, len);
```

- Getting the Request

```
read (sock, &len, 4); /* read the size the of result string */
len = ntohl (len);
size = 0;
while (len > size) {
    n = read (sock, result_string + size, len - size);
    size += n;
}
/* now result string contains the result from the server */
```

3. Error Handling
   Most system calls will return -1 as an error indicator. The server and client programs should handle as many error conditions as possible.

4. Grading Policy and Submission Requirement

   (a) Two to five students will work together on this project. No more than three CS graduate students are allowed in one group. A too small or too big number of group members, say 2 or 5, will affect your grade negatively. Each group should only submit your project once. Both hardcopy and the electronic copy are required. A cover page is required for the hard copy. The cover page should contain the following information: the course name, your names and user account names, the project name, your majors, student status (undergraduate or graduate student), and so on.

   (b) Your grade comes from three parts:
      - Executable programs (70%)
      - Program Stability (10%)
      - Documents (README, Makefile, user guide, design report) (20%)

   (c) Each group should write a design report which contains the following material:
      - The protocol design, including the definition of the request and the response (meaning and format), the physical layout of the request and the response.
      - The user interface design of the your client program. It should look like the user guide to your application program.
      - The skeletons of server and client programs, including the explanation of the working process for both the server and the client, the error handling conditions, and so on.
      - Proper in-line comments are required.

   (d) The README file should contain the following information:

- A detailed contribution list describing each student's contribution to the project. The description should be very specific. Vague accounts should be avoided.
- Instructions to build your application programs.
- Instructions to test your application programs.

give a detailed contribution list describing each student's contribution to the project. It should be specific.

5. Database Processing

   (a) Using the UNIX Database Library (libgdbm.a)
   DBM is a library of routines that manages data files that contain key/data pairs. The access provided is that of storing, retrieval, and deletion by key and a non-sorted traversal of all keys. (type '`man gdbm`' to get the man pages)

   (b) Example `db.c`

```
#include    <stdio.h>
#include    <gdbm/ndbm.h>
#include    <fcntl.h>

void error (const char *msg)
{
    fprintf (stderr, "%s\n", msg);
    exit (-1);
}

struct {
    char *name;
    char *phone;
} data [] = {
    {"D. C. Lynch", "(316)999-4444"},
    {"M. W. Joy",   "(316)999-4445"},
    {"J. Peterson", "(316)999-4446"},
};

int main (int argc, char **argv)
{
    DBM     *db;
    datum   key, value;
    int     i;

    db = dbm_open ("yellowpage", O_RDWR | O_CREAT, 0666);

    if (db == NULL)
        error ("could not open the database");

    /* store data into the database */
```

```
    for (i = 0; i < sizeof (data) / sizeof (data[0]); i++) {
        key.dptr = data[i].name;
        key.dsize = strlen (data[i].name) + 1;

        value.dptr = data[i].phone;
        value.dsize = strlen (data[i].phone) + 1;
        if (dbm_store (db, key, value, DBM_REPLACE) != 0)
            fprintf (stderr, "could not store (%s, %s)\n",
                     data[i].name, data[i].phone);
    }

    /*  traverse the whole database */
    for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db)) {
        value = dbm_fetch(db, key);
        printf ("%s\t%s\n", key.dptr, value.dptr);
    }

    /* serch the value in the database whose key is specified*/
    key.dptr = "J. Peterson";
    key.dsize = strlen ("J. Peterson") + 1;
    value = dbm_fetch (db, key);
    printf ("%s\t%s\n", key.dptr, value.dptr);

    /* serch the key in the database */
    dbm_delete (db, key);
    value = dbm_fetch (db, key);
    if (value.dptr == NULL)
        printf ("phone number for %s is unknown\n", key.dptr);

    dbm_close (db);
}
```

To compile the code, you can use the following command:

```
% gcc db.c -o db -lgdbm
```

6. Some Suggestions

- To make the project more managable you may try to finish the project in several stages. For example, in the first stage you only implement the server as a sequential server that can only server a client each time.
- You may debug your programs with some tools, such as `gdb`, `netstat`.
- Sticking to a good coding standard will make your program more readable and manageable. It's very important for developing and maintaining a group project. Here is a page of links to some coding standards:
  `http://www.google.com/search?q=c+coding+standard`