PasswordManager
A Computer-Networks Project

Bita Alexandru

Faculty of Computer-Science, University of Alexandru Ioan Cuza
bita.alexandru24@gmail.com

Abstract. An application which implements a concurrent server for users to easily store and manage their credentials.

Keywords: Passwords · Manager · SaaS · Networking · Concurrency

1 Introduction

A password manager is a service that keeps track of one's credentials through the interaction of the user.

This project[1] resembles a server/client application with such purpose and is avaliable to anyone with access to the Internet.

In order to make use of this service, the client has at their disposal a set of commands, each of them guiding the client at every step. For example, to register an account the client types *register* and then they're required to simply fill the fields that follow. The same goes for *login* and the commands which become available immediately after a succesfull login, such as *add* (to save a new password) and *view* (to view the saved passwords). Naturally, the client types *exit* to quit the application.

2 Used technologies

It is well known that TCP is a very reliable protocol. No wonder it's the most commonly used on the Internet. However, its reliability and safety come at the cost of speed. It's much slower than UDP, it's heavy-weight and doesn't support broadcasting. Even so, despite the increased performance in the case of UDP, I chose to use TCP for my project as it provides many advantages. It guarantees the delivery of data to the destination; provides extensive error checking mechanisms and has the possibility of retransmissing lost packets as opposed to the UDP protocol[2]. As we deal with sensitive information, it is better to use a more reliable protocol and this is why I use TCP.

The server stores all of the information in a database. For this I'm using SQLite, a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.[3]

3 Application architecture

The application uses sockets to estabilish the communication link between server and client and unique child processes are assigned to each new connection in order to mantain concurrency.

Once a client disconnects from the server, the child process assigned to that client is deallocated so no zombie tasks are created. In case the server terminates unexpectedly, all of its child processes are deallocated so no orphan tasks are created.

The following diagram[4] illustrates the flow of the application:

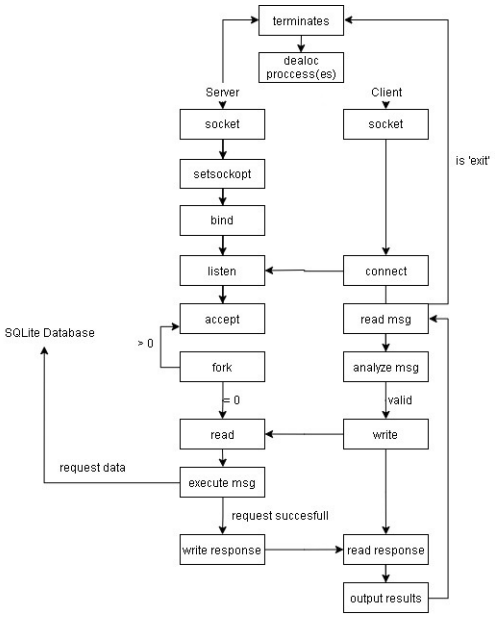

Fig. 1. Application flowchart

4 Implementation details

Here are some code snippets used to perform the database creation, the login and the addition of passwords:

```
void init_db() // create the initial database
{
    sqlite3_open("main.db",&db);

    char *content = "CREATE TABLE Accounts(" \
                    "User TEXT NOT NULL," \
                    "Pass TEXT NOT NULL);" \

                    "CREATE TABLE Passwords(" \
                    "User TEXT NOT NULL," \
                    "Username TEXT NOT NULL," \
                    "Password TEXT NOT NULL," \
                    "Title TEXT NOT NULL," \
                    "Category TEXT NOT NULL," \
                    "Url TEXT NOT NULL," \
                    "Notes TEXT NOT NULL);"; // sql query

    sqlite3_exec(db,content,0,0,0); // execute query
}
```

```
else if(strstr(msg,"login!")) // 'msg' is provided by the client
{
    char user[NRM_SIZE] = {0}, pass[NRM_SIZE] = {0};
    char *t;
    t = strtok(msg,"!"); // 'msg' format is '<command!>
    t = strtok(NULL,"_"); strcpy(user,t); // <username_>
    t = strtok(NULL,"_"); strcpy(pass,t); // <password>

    int ok = 0;
    char *content = "SELECT User,Pass FROM Accounts;";
    int ret = sqlite3_prepare_v2(db,content,-1,&stmt,0);
    do
    {
        ret = sqlite3_step(stmt);
        if(ret == SQLITE_ROW &&
          strcmp(user,sqlite3_column_text(stmt,0)) == 0 &&
          strcmp(pass,sqlite3_column_text(stmt,1)) == 0) ok = 1;
    } while(ret == SQLITE_ROW);

    bzero(msg,MAX_SIZE);
    if(!ok)
    {
        printf("login attempt failed by user %s\n",user); fflush(stdout);
        strcpy(msg,"LOG_NOK");
    }
    else
    {
        printf("%s logged in\n",user); fflush(stdout);
        strcpy(msg,"LOG_OK");
        strcpy(my_user,user);
    }
}
```

```
else if(strstr(msg,"add!")) {
    char user[NRM_SIZE]={0}, username[NRM_SIZE]={0},
    password[NRM_SIZE]={0}, title[NRM_SIZE]={0},
    category[NRM_SIZE]={0}, url[NRM_SIZE]={0}, notes[NRM_SIZE]={0};
    char *t;
    t = strtok(msg,"!"); // 'msg' format is <command!>
    t = strtok(NULL,"_"); strcpy(user,t); // <user_>
    t = strtok(NULL,"_"); strcpy(username,t); // <username_>
    t = strtok(NULL,"_"); strcpy(password,t); // ...
    t = strtok(NULL,"_"); strcpy(title,t);
    t = strtok(NULL,"_"); strcpy(category,t);
    t = strtok(NULL,"_"); strcpy(url,t);
    t = strtok(NULL,"_"); strcpy(notes,t); // <notes>
    int ok = 1;
    char *content = "SELECT Title FROM Passwords;";
    int ret = sqlite3_prepare_v2(db,content,-1,&stmt,0);
    do {
        ret = sqlite3_step(stmt); // iterate through rows
        if(ret == SQLITE_ROW &&
          strcmp(title,sqlite3_column_text(stmt,0)) == 0) ok = 0;
    } while(ret == SQLITE_ROW && ok);
    bzero(msg,MAX_SIZE);
    if(!ok) {
        printf("addition attempt failed by user %s\n",my_user);
        fflush(stdout); strcpy(msg,"ADD_NOK");
    }
    else {
        content = "INSERT INTO Passwords
        (User,Username,Password,Title,Category,Url,Notes)
        VALUES (?,?,?,?,?,?,?);";
        ret = sqlite3_prepare_v2(db,content,-1,&stmt,0); // replace '?'
        sqlite3_bind_text(stmt,1,user,-1,SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt,2,username,-1,SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt,3,password,-1,SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt,4,title,-1,SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt,5,category,-1,SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt,6,url,-1,SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt,7,notes,-1,SQLITE_TRANSIENT);
        ret = sqlite3_step(stmt);

        sqlite3_finalize(stmt);
        printf("new addition by account with username %s\n",my_user);
        fflush(stdout); strcpy(msg,"ADD_OK");
    }
}
```

5 Conclusions

In conclusion, this paper presents a project which implements a password manager service over the TCP/IP protocol and uses child processes to create concurrency. A faster implementation of such project would require using threads but since the complexity of the application is relatively small this is not an absolute necessity. Moreover, the information could be encrypted in order to provide a more secure use to the application. Overall, the application manages to perform extremely well and it's really easy to use.

References

1. Projects list
   `https://profs.info.uaic.ro/~computernetworks/ProiecteNet2019.php`
2. TCP vs UDP
   `https://www.geeksforgeeks.org/differences-between-tcp-and-udp/`
3. SQLite
   `https://www.sqlite.org/cintro.html`
4. draw.io
   `https://www.draw.io/`