Password Manager

Cires Rares

An universitar: 2020-2021
Grupa: 2E3

1 Introducere

Proiectul ales se numeste Password Manager. La baza este o aplicatie server/client, unde serverul stocheaza intr-o baza de date toate informatiile de conectare pentru site-urile web pe care clientul le introduce. Baza de date este criptata cu o parola master - parola master este singura parola pe care trebuie sa o retina clientul.

2 Tehnologiile utilizate

In ceea ce consta tehnologiile utilizate, pentru aceasta aplicatie, am ajuns la concluzia ca va fi nevoie de un server TCP concurent deoarece este mult mai sigur decat un server UDP, aplicatia trebuie sa garanteze ca toate datele se trimit in intregime si in ordine, evitand riscul de corupere acestora sau primirea unor informatii incomplete.

Este nevoie de o implementare concurenta a modelului server/client si nu una iterativa deoarece, pentru aceasta aplicatie, este nevoie sa asiguram cererile simultane a mai multor clienti, ceea ce metoda iterativa nu ne poate oferi, astfel am ales implementarea serverului cu ajutorul thread-urilor. Am ales sa implementez serverul folosind thread-uri deoarece sunt mai rapide, ocupa mai putina memorie deoarece impart memoria din procesul care le-a creat, avand memorie proprie doar pentru operatiile efectuate in corpul acestora. Deci nu este nevoie de comunicare inter-procese pentru a modifica datele din procesul initial,toate thread-urile avand "drepturi" de modificare a datelor.

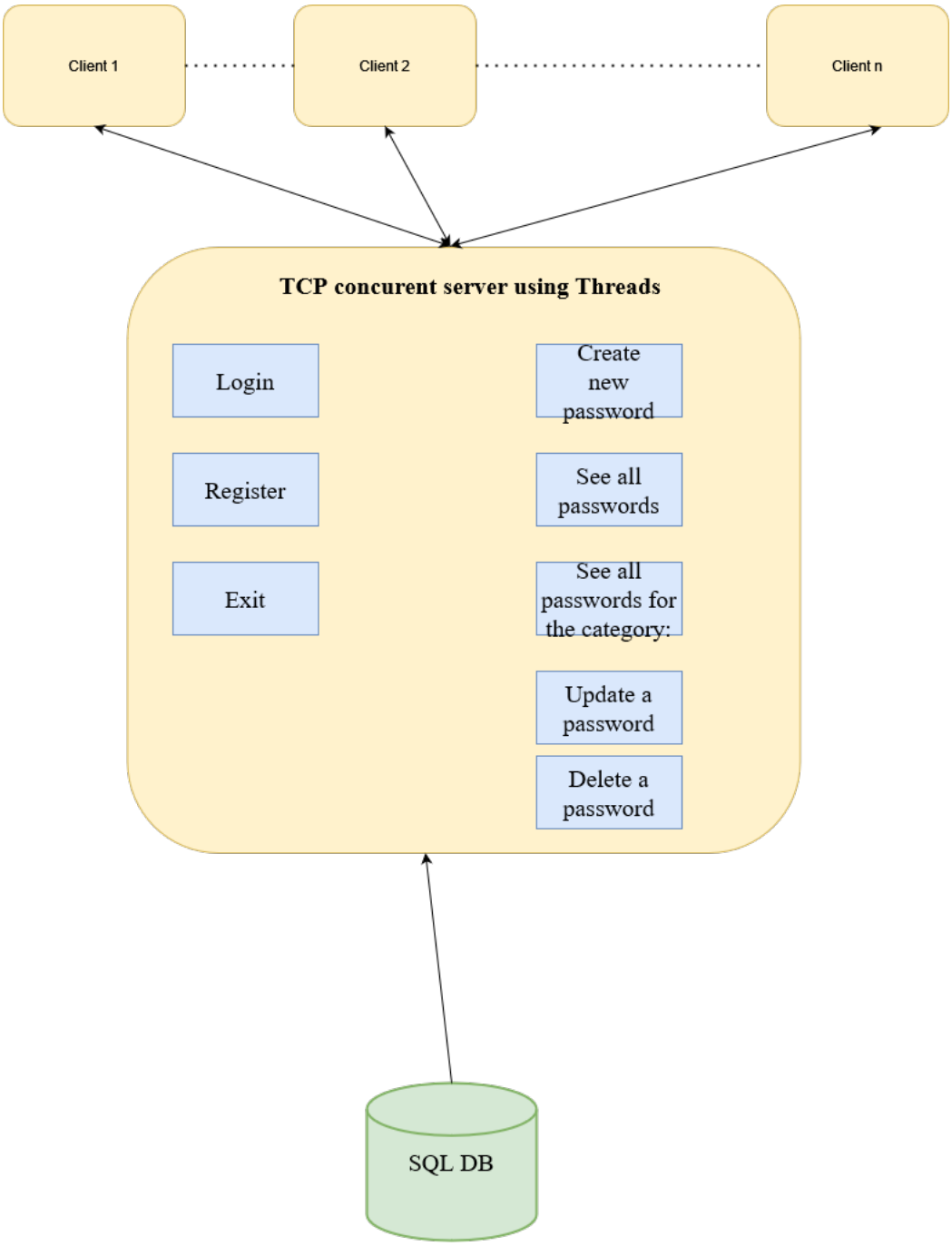
Pentru a gestiona datele necesare aplicatiei am implementat baze de date SQL cu ajutorul bibliotecii sqlite3. O baza de date ca aceasta face mult mai usoara manipularea dateleor decat un fisier .txt care ar retine datele.

3 Arhitectura aplicatiei

3.1 Conceptele implicate

Ca si conceptele implicate, unul dintre ele ar fi conceptul de transmitere sigura a informatiilor canal, protocol de transport orientat conexiune, fara pierdere de informatii care vizeaza oferirea calitatii maxime a serviciilor si care integreaza mecanisme de stabilire si de eliberare a conexiunii. De asemenea, conceptul de asigurare a ordinii de transmitere a datelor prin socket intre client si server, astfel conexiunile sunt identificate prin perechi reprezentate de o adresa IP si un PORT, un socket putand fi partajat de conexiuni multiple de pe aceeasi masina.

3.2 Diagrama aplicatiei



4 Detalii de implementare

4.1 Cod relevant proiectului

Comunicarea dintre server si client se realizeaza cu ajutorul unui socket. Utilizarea unui socket pentru comunicare este mult mai avantajoasa datorita faptului ca socket-ul este bidirectional, comparativ cu pipe-urile sau fifo-urile care sunt unidirectionale, deci nu este nevoie decat de un singur socket pentru a comunica.

In structura codului am adaugat functii care sunt necesare operatiilor efectuate pe baza de date: callback, pentru create table si insert;

```
int callback(void *ans, int argc, char **argv, char **azColName)
{

    char *answer = (char *)ans;

    for (int i = 0; i < argc; i++)
    {
        strcat(answer, azColName[i]);
        strcat(answer, " = ");
        strcat(answer, argv[i] ? argv[i] : "NULL");
        strcat(answer, "\n");
    }
    strcat(answer, "\n");
    return 0;
}
```

check_user, check_account, pentru verificarea unor informatii din baza de date;

```
int check_user(const char *s, char *username)
{
    sqlite3 *db;
    char *err_msg = 0;
    int rc = sqlite3_open(s, &db);
    if (rc != SQLITE_OK)
    {

        fprintf(stderr, "Cannot open database: %s\n",
                sqlite3_errmsg(db));
        sqlite3_close(db);

        return -1;
    }
    string sql;
    char comanda[400];
    char result[10];
    bzero(result, 10);
    sprintf(comanda, "SELECT EXISTS(SELECT 1 FROM USERS WHERE username='%s');", username);
    sql = comanda;
    rc = sqlite3_exec(db, sql.c_str(), callback2, result, &err_msg);
    if (result[0] == '1')
        return 1;
    else
        return 0;
    if (rc != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", err_msg);
        sqlite3_free(err_msg);
        sqlite3_close(db);

        return -1;
    }
    return 0;
}
```

insertIn_table, insert_inpasswordsDB, update_DB, pentru inserarea sau actualizarea unor informatii din baza de date;get_ID, pentru obtinerea cheiei primare a bazei de date USERS.

```
int get_ID(const char *s, char *username)
{
    sqlite3 *db;
    char *err_msg = 0;
    int rc = sqlite3_open(s, &db);

    if (rc != SQLITE_OK)
    {

        fprintf(stderr, "Cannot open database: %s\n",
                sqlite3_errmsg(db));
        sqlite3_close(db);

        return -1;
    }

    char result[500];
    bzero(result, 500);
    char comanda[50];
    sprintf(comanda, "SELECT ID FROM USERS WHERE username='%s'", username);
    string sql = comanda;
    rc = sqlite3_exec(db, sql.c_str(), callback2, result, &err_msg);

    return atoi(result);
}
```

Pentru a asigura concurenta, in implementarea serverului am folosit thread-uri, astfel ca pentru fiecare client care se conecteaza, este creat un nou thread care se ocupa de cererile acestuia.

```
while (1)
{
    int client;
    thData *td;
    socklen_t length;
    printf("[server]Asteptam la portul %d...\n", PORT);
    fflush(stdout);
    if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0)
    {
        perror("[server]Eroare la accept().\n");
        continue;
    }
    td = (struct thData *)malloc(sizeof(struct thData));
    td->idThread = i++;
    td->cl = client;
    pthread_create(&th[i], NULL, &treat, td);
}
```

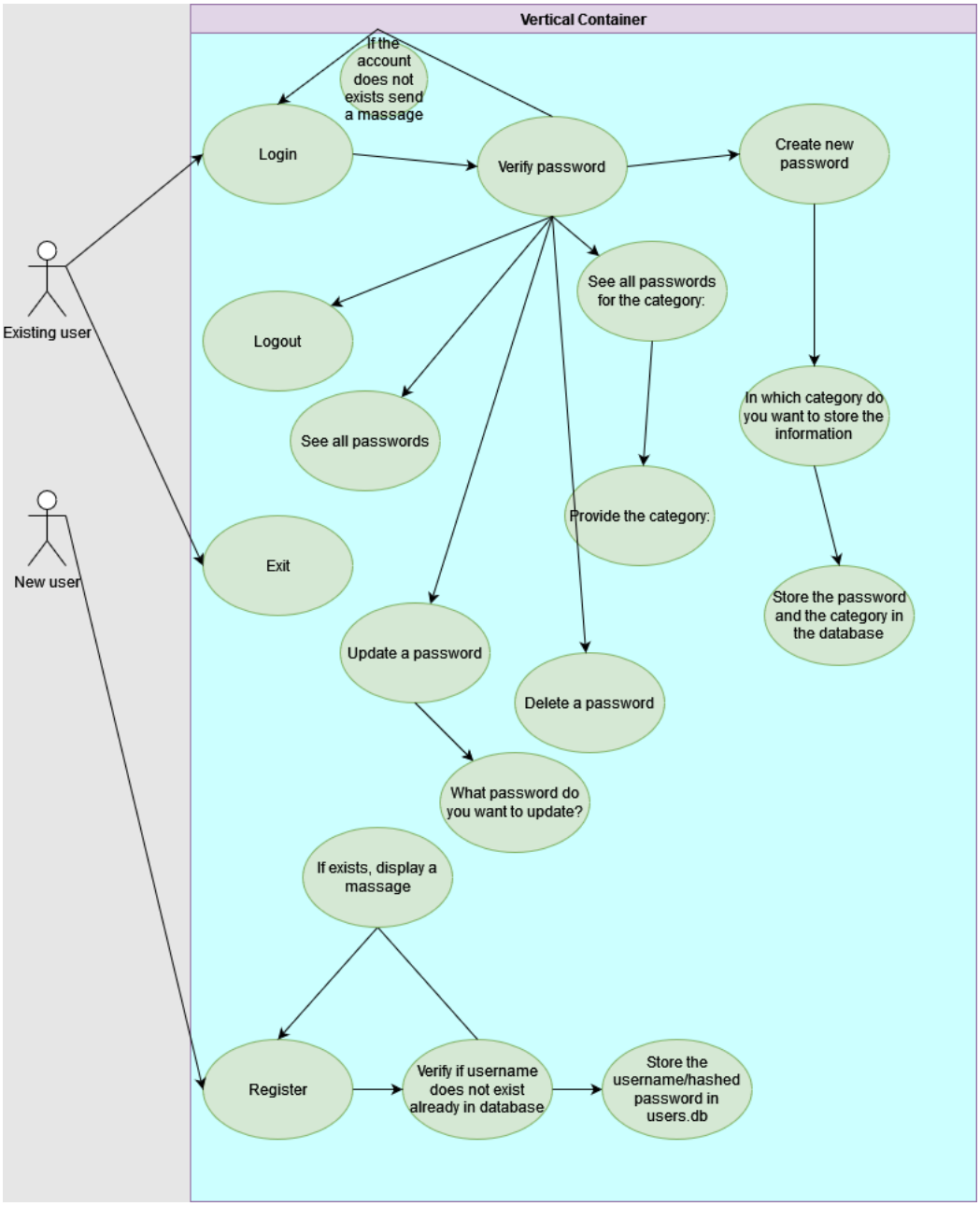
In functia *main* am creat baza de date si tabelele necesare stocarii si prelucrarii informatiilor; am creat socket-ul utilizat in comunicare, am atasat socket-ul, cu bind, si partea de ascultare a clientilor care doresc sa se conecteze, listen.

In functia *raspunde* se realizeaza interactiunea dintre client si server. Clientul se poate conecta la server, utilizand comanda *Login*, isi poate creea un cont nou utilizand comanda *Register* sau poate incheia conexiunea cu comanda *Exit*. Odata logat, utilizatorul are optiunea de a-si stoca o noua parola,

de a-si vedea toate parolele stocate, de a-si vedea parolele stocate in functie de categoria parolei, de a-si updata informatiile unei parole stocate, sau de a sterge o parola stocata in baza de date a utilizatorului.

```
================
1.Create new password
2.See all passwords
3.See all passwords for the category:
4.Update a password
5.Delete a password
6.Exit
================
```

4.2 Use cases diagram



5 Concluzii si imbunatatiri

1. Transmiterea exacta a numarului de bytes de catre server la client.
2. Acoperirea cazului in care user-ul inchide subit aplicatia in timpul scrierii unui mesaj catre server.
3. Sa existe optiunea de recuperare a parolei.
4. Sa existe optiunea de sortare a parolelor introduse in baza de date a clientilor.

6 Bibliografie

https://www.linuxjournal.com/content/accessing-sqlite-c
https://github.com/KalleHallden/pwManager
https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_node/libc_650.html
https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_node/libc_650.html
https://stackoverflow.com/questions/52146528/how-to-validate-salted-and-hashed-password-in-c-sharp
https://www.delftstack.com/howto/c/crypt-function-in-c/
https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm
https://sqlite.org/c_interface.html