VirtualSoc

Caleavalea Vasile-Andrei 2B1

Universitatea Alexandru-Ioan Cuza Iasi, Facultatea de Informatica

1 Introducere

Proiectul **VirtualSoc** dezvolta o aplicație tip **client - server** care are ca scop simularea unei rețele sociale.

In aceasta aplicatie, utilizatorii au posibilitatea de a se inregistra (cu conturi obisnuite sau de administrator), de a-si seta profilurile ca fiind publice sau private, si pot trimite mesaje private catre alti utilizatori sau grupuri de utilizatori. Acestia pot, de asemenea, sa adauge persoane in lista de prieteni si sa posteze stiri catre grupuri speciale sau catre toti utilizatorii (in mod public).Postarile publice pot fi vizualizate si de utilizatori care nu sunt autentificati.

Obiectivul acestui proiect este de a realiza o retea sociala de baza cu caracteristici esentiale, implementand o aplicatie de tip **client - server**.

2 Tehnologii aplicate

Pentru realizarea proiectului, am folosit:

1. **TCP(Transmission Control Protocol)**. Acest protocol de comunicare asigura integritatea transmiterii de informatii (spre deosebire de **UDP**, care poate pierde anumite pachete de date), intre mai multi clienti si server. Astfel, avem o conexiune stabila de comunicare si putem primi de la clienti oricate solicitari (mesage, postari si alte comenzi) si sa le transmitem serverului, de unde primim ulterior raspunsuri corespunzatoare.
2. **SQLite3**. Pentru a stoca toate informatiile despre utilizatori, mesaje si alte date necesare proiectului ne folosim de baza de date Sqlite3. Aceasta este usor de folosit in C si eficienta pentru gestionarea datelor in mod local.

3 Structura aplicatiei

Proiectul este impartit in 2 componente principale: **serverul** si **clientul**.

Serverul va porni si va astepta conexiuni de la clienti, acceptându-le prin intermediul unui multiplexor. Pe masura ce accepta clienti, va primi si comenzi de la clienții existenti in mod concurent.

In functie de comanda primita, va face operatii de **select** si **insert** asupra bazei de date, toate fiind executate prin **UsersManager**. Operatiile de **create** asupra bazei de date se vor face doar de catre server, la initializarea bazei de

date, in caz ca aceasta lipeste, sau in cazul in care unele tabele nu au fost implementate inca(din lipsa datelor).

Clientul se va conecta la server in modul **neautentificat**. In acest mod poate folosi doar comenzile **get-posts** (va primi doar postarile publice), **login <user><password>** si **exit**. Daca acesta se va loga, va avea acces la mai multe comenzi:

1. logout
2. send <user> <message>
3. get-messages <user> (mesaje intre utilizatorul care trimite comanda si <user>
4. get-online-users
5. get-posts
6. change-to-private-profile (postarile devin vizibile doar pentru prieteni)
7. change-to-public-profile
8. create-post <close_friends/friends/public> <content>
9. add_to_friends <user>
10. add_to_close_friends <user>
11. quit
12. get-friends
13. get-close-friends
14. create-group <name>
15. add-to-group <name> <user>
16. send-group <name> <content>
17. get-group-messages <name>

Aceste comenzi se aplica pentru utilizatorii obisnuiti. Utilizatorii admini vor fi adaugati manual in baza de date(nu prin cadrul aplicatiei), in prezent, in baza de date exista utilizatorul *admin* cu parola *admin*. Acestia vor avea un set de comenzi in plus:

1. delete-user <username>
2. delete-post <post id>
3. delete-messages-of-user <username>
4. delete-posts-of-user <username>

Clientul va citi de la server constant mesaje, deci, in caz ca este online, va primi mesaje in timp real de la alti utilizatori online. Daca un utilizator nu este online, va primi mesajele cand se va conecta.

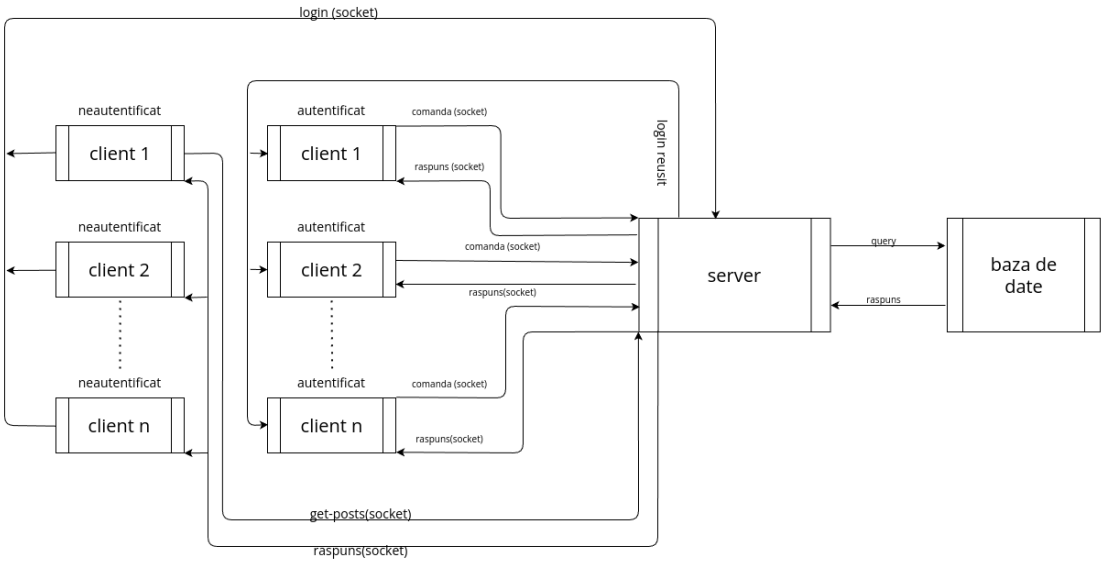

Fig. 1. Diagrama aplicatiei

4 Aspecte de implementare

Concurența serverului

Trebuie sa ne asiguram ca putem servi mai multi clienti in acelasi timp, astfel folosim un **multiplexor**. Vom retine clientii conectati la server, iar cu functia **select** ii vom putea gestiona eficient. Atunci cand un client nou se conecteaza, il vom adauga in lista noastra de clienti conectati cu ajutorul **accept** si **FD_SET**.

Cu ajutorul **while-ului** de mai jos, verificam mereu daca avem un client care doreste sa se conecteze, si daca da, il acceptam si il punem in **active_fds**. Apoi trecem prin toti utilizatorii si incercam sa primim mesaj de la ei. In caz afirmativ, luam comanda de la utilizator si o procesam corespunzator.

Cod 1.1. Cod C din server

```
while (1) {
    ...
    if (select(maxim_fd + 1, &read_fds, NULL, NULL, &tv) < 0) {
        error_message("[server] Eroare la select.\n");
    }

    if (FD_ISSET(socket_fd, &read_fds)) {
        ...
        int client = accept(socket_fd, (struct sockaddr*)&from,
```

```
                (socklen_t*)&len);
            ...
            FD_SET(client, &active_fds);
            ...
        }

        for (int fd = 0; fd <= maxim_fd; fd++) {
            if (fd != socket_fd && FD_ISSET(fd, &read_fds)) {
                run_command(&manager, fd);
            }
        }
}
```

De asemenea, serverul este responsabil si pentru comunicarea cu baza de date. Acesta, in functie de comanda primita de la user, va face query-uri asupra bazei de date prin comenzi de genul *select* si *insert*. Baza de date este salvata local in fisierul **database.db**. Operatii directe asupra bazei de date poate face doar serverul prin variabila **UsersManager**, ce are functiile ei specifice in fisierul **UsersManager.h**.

Cod 1.2. Cod C din server

```
char* query = "SELECT id FROM users WHERE username = ?;";
if (prepare(database, query, &stmt) == -1) {
    return -1;
}
sqlite3_bind_text(stmt, 1, sender, -1, SQLITE_STATIC);
if (sqlite3_step(stmt) == SQLITE_ROW) {
    s = sqlite3_column_int(stmt, 0);
}
```

Clientul se conecteaza la server, apoi face 2 lucruri: citeste comenzi de la tastatura pentru a le trimite catre server si citeste si raspunsuri de la server in acelasi timp(mesaje de la alti utilizatori online, etc.). Pentru ca acest lucru sa fie posibil avem un fork ce creaza 2 procese in paralel. Parintele citeste de la tastatura si trimite catre server, iar fiul se ocupa cu receptionarea mesajelor de la server.

Utilizatorii pot face urmatoarele lucruri: citirea istoricului mesajelor cu alti utilizatori, trimiterea si primirea de mesaje in timp real, crearea de stiri, adaugarea utilizatorilor in lista de prieteni si lista de prieteni apropriati, aflarea utilizatorilor online.

Baza de date este reprezentata in diagrama de mai sus. Pentru preluarea mesajelor/postarilor si altor informatii din baza de date facem integorari asupra acesteia prin **DatabaseManager** (implementat in **DatabaseManager.h**. Avand in vedere ca mesajele si stirile au id-ul utilizatorilor, putem cu usurinta sa primim mesaje cu numele utilizatorilor, la fel si pentru postari si listele de prieteni/prieteni apropriati. In baza de date mai exista si tabelele *groups*, *groups_messages*, *group_members*.

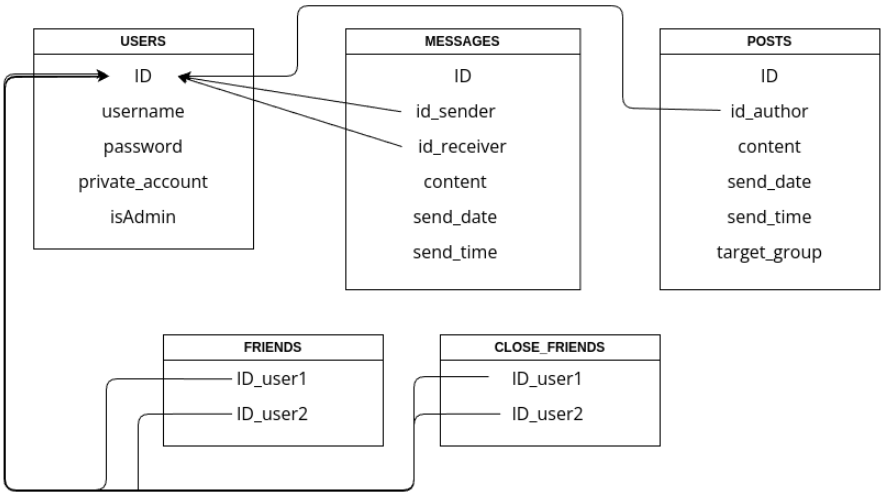

Fig. 2. Diagrama baza de date

5 Concluzii

Aceasta aplicatie poate avea si imbunatatiri precum o interfata grafica, posibilitatea de a sti daca un utilizator ti-a citit mesajul sau nu, optiunea de a da reply la un mesaj, posibilitatea de a insera imagini in stiri/mesaje, posibilitatea de a-ti pune o poza de profil.

Aceasta aplicatie ilustreaza tipul de aplicatie **server - client** clasica, ce foloseste, de asemenea, si o baza de date,

References

1. Wikipedia, https://en.wikipedia.org/wiki/Transmission_Control_Protocol,
2. Sqlite, https://www.sqlite.org/cintro.html,
3. Port70, https://port70.net/ nsz/c/c11/n1570.html,
4. Cursuri UAIC, https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php