Rețele de calculatoare. Raportul tehnic.
VirtualSoc

Bursuc E. Eduarda

Universitatea A.I.Cuza

1 Introducere

Proiectul ales, VirtualSoc, propune simularea unei rețele sociale cu următoarele funcții implementate: înregistrarea unui cont nou public sau privat de tip user (client) sau admin (administrator), autentificarea prin nume de utilizator și parolă, adăugarea prietenilor în grupuri speciale (prieten sau prieten apropiat), publicarea noutăților atât public cât și pentru grupurile speciale și opțiunea de chat ce permite comunicarea între doi sau mai mulți utilizatori activi.

2 Tehnologii Aplicate

2.1 TCP

Rețelele sociale implică un schimb constant de date între client și server, unde integritatea și ordinea acestora este foarte importantă, motiv pentru care am optat pentru un protocolul de comunicare TCP (Transmission Control Protocol). Acesta spre deosebire de UDP se ocupă bine de gestionarea conexiunilor, asigurând că datele utilizatorilor sunt transferate în mod corect și complet. De asemenea, pentru implementarea concurenței ce asigură utilizarea paralelă a rețelei de către mai mulți clienți, am folosit un server TCP cu multiplexare I/O prin primitiva select() care permite gestionarea și prelucrarea clienților simultană și interacționarea în timp real.

2.2 SQLite

Pentru stocarea datelor despre utilizatori, profiluri, postări și relații, am optat pentru sistemul SQLite3 ce oferă o soluție ușor de implementat și fără server pentru gestionarea bazelor de date în cadrul aplicațiilor, fiind portabil și eficient în citire.

2.3 Socket

Utilizarea socket-urilor în cadrul TCP oferă o soluție robustă și eficientă pentru dezvoltarea aplicațiilor de rețea care necesită o comunicare fiabilă și bidirecțională. Unele din beneficiile socket-ului importante pentru proiectul dat: comunicarea bidirecțională, fiabilitatea, reasamblarea pachetelor, reconectarea automată, adresarea și menținerea unor conexiuni persistente.

2.4 PThreads

PThreads sau Posix Threads, reprezintă o interfață standard pentru crearea și gestionarea firelor de execuție specific sistemelor de operare de tip Unix. Aceasta furnizează un set de funcții ce permit crearea și sincronizarea firelor, lucru ce asigură folosirea concurentă și paralelă a rețelei sociale de către mai mulți utilizatori. De asemenea, față de un TCP implementat cu fork(), firele de conexiunea sunt create în cadrul unui singur proces, ceea ce are ca rezultat folosirea mai eficientă a resurselor de sistem.

2.5 Mutex

Tehnologia Mutex asigură accesarea exclusivă a resurselor partajate în cadrul programării cu mai multe fire de execuție. În cazul aplicației elaborate, a fost necesară utilizarea tehnologiei respective pentru prevenirea modificarii și adăugarii înregistrărilor simultane în baza de date, lucru ce putea decurge în înregistrarea a doi utilizatori cu același nume sau deformarea unei postări.

2.6 select()

Primitiva select() pentru multiplexare este o soluție pentru implementarea chatului ce permite primirea mesajelor în mod neblocant de la fiecare utilizator într-o sesiune și de asemenea, la inmplementarea aplicației client pentru scrierea în socket și citirea din stdin. De asemenea, în cazul unor mesaje ce depășesc lungimea cerută, aceasta primitivă va citi din socket atât timp cât va fi activitate în acesta.

3 Structura Aplicației

Aplicația utilizează o arhitectură client-server, unde clientul transmite cereri către server prin intermediul unui set de comenzi bine definit: registration, login, logout, show-users, show-profile, add-friend, new-post și chat cu parametri potriviți. În urma primirii unei cereri, serverul analizează comanda și cere, dacă este nevoie, informații suplimentare, după care se ocupă de prelucrarea datelor: verificarea acestora în baza de date, înregistrarea unor date noi, colectarea conținutului dorit etc. În urma procesării cererii, serverul trimite clientului un mesaj de succes sau conținutul cerut de acesta în conformitate cu specificații și datele deja cunoscute din baza de date.

Diagrama detaliată prezintă o construcție aproximată a programului incluzând comenzile specifice și tabelele din baza de date necesare.

4 Aspecte de Implementare

4.1 Descrierea detaliată

Inițial am definit o structură User ce va diferenția utilizatorii după două componente: username (nume de utilizator) și fd (descriptor de fișier), aceasta va fi

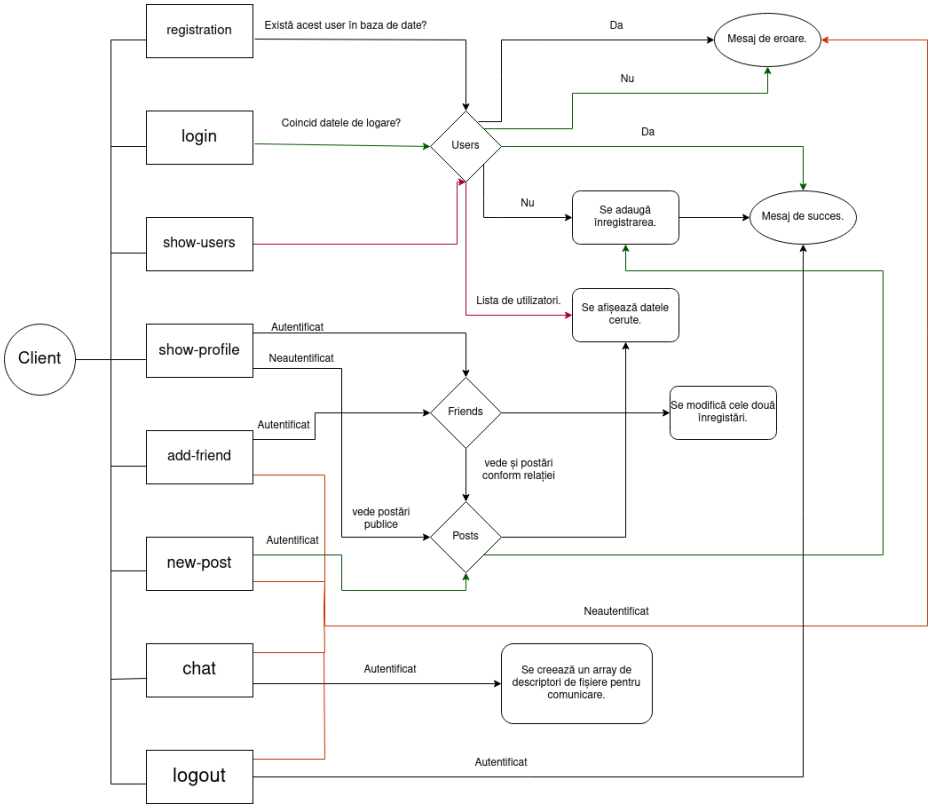

Fig. 1. Diagrama detaliată

utilizată ulterior pentru determinarea statutului de autentificare a unui utilizator și recunoașterea acestuia în lista de descriptori de fișier de citire. De asemenea, aceasta conține două variabile ce vor fi folosite pentru opțiunea de chat: chat_request și chat_accepted care vor permite selectarea corectă a utilizatorilor pentru o sesiune chat și nu vor permite încadrarea unui utilizator în mai multe conversații simultan.

Implementarea propriu-zisă constituie un set de comenzi pe care le poate introduce un utilizator pentru a introduce date sau vizualiza conținut. Toate comenzile vor fi inițial recunoscute în funcția principală și redericționate funcțiilor corespunzatoare cu parametri potriviți.

1. Comanda **registration** este preluată de o funcție specifică în cazul unui utilizator neutentificat, care primește descriptorul de fișier al utilizatorului și cere un set de informații, cum ar fi tipul de cont, tipul de utilizator, un nume de utilizator și o parolă. În momentul primirii unui nume de utilizator, funcția va accesa baza de date Users și va verifica dacă există un cont cu același nume, în caz afirmativ se afișează un mesaj de eroare, iar în caz contrar se alocă informația în baza de date și se va trimite un mesaj de succes.

2. Comanda **login** este prelucrată inițial prin verificarea componentei username a structurii User care ne indică statutul de autentificare a utilizatorului. În cazul în care utilizatorul este autentificat se va afișa un mesaj specific, în caz contrar se va cere și verifica numele de utilizator introdus în baza de date Users și în cazul existenței unui astfel de utilizator se va cere parola până când aceasta nu va coencide cu cea din baza de date. În cazul autentificării cu succes, componenta username în structura User va fi modificată.

3. Comanda **logout** este executată în funcția principală și verifică inițial dacă structura User specifică utilizatorlui are un username atașat (lucru ce ne spune că utilizatorul este autentificat) și în acest caz o va șterge, iar în caz contrar - trimite un mesaj utilizatorului că nu este autentificat.

4. Comanda **show-users** transmite doar descriptorul de fișier funcției ce accesează baza de date Users și colectează toți utilizatorii existenți pentru a-i afișa utilizatorului.

5. Comanda **show-profile : username** transmite funcției structura User și numele utilizatorului ai cărui profil vrea să-l viziteze. În cazul unui utilizator neautentificat, se accesează baza de date Posts și se selectează linia utilizatorului cerut și coloana cu postări de tip public, iar în caz contrar se verifică tipul de relație a celor doi utilizatori dacă există (friends, close friends) în baza de date Friends și în funcția acesteia se cer postări din Posts.

6. Comanda **add-friend : username** este prelucrată de o funcție ce primește structura User, numele utilizatorului ce trebuie adăugat, și cere relația dorită între ei (prieteni sau prieteni apropiați) și modifcă sau adaugă o nouă înregistrare în baza de date Friends. De asemenea, trimite un mesaj de succes. În cazul unui utilizator neautenficat, se va trimite doar un mesaj de eroare.

7. Comanda **new-post** este redirecționată funcției ce primește structura User și cere tipul postării (public, friends, close friends) și conținutul acesteia, urmând ca uletrior să introducă o nouă înregistrare în baza de date Posts pe linia utilizatorului și coloana tipului cerut. În cazul unui utilizator neautenficat, se va trimite doar un mesaj de eroare.

8. Comanda **chat : username ... username** este autentificată în funcția principală unde se creează un vector cu descriptorii de fișiere corespunzatoare numelor de utilizatori activi și se transmite unei funcții ce va avea proprietatea unui broadcast. Pentru utilizatorii dați se va genera un mesaj specific pentru a înștiința aceasta cerere după care cu excepția utilizatorului ce a creat chatul, utlizatorii vor avea posibilitatea de a accepta sau nu accesarea sesiunii. La generarea unui mesaj specific stop-chat un utilizator poate ieși din conversație, iar ultimului utilizator rămas i se va trimite un mesaj specific pentru incheierea chat-ului. În cazul unui utilizator neautenficat, se va trimite doar un mesaj de eroare.

De asemenea, pentru simularea mai bună a unei aplicații de genul, am folosit funcția getpass() pentru a selecta parola introdusă de utilizator fără ca aceasta să fie afișată în momentul scrierii și ulterior.

4.2 Secvențe de cod

În următoarele figuri se pot observa secțiuni de cod specifice lucrului cu baze de date și implementarea structurii User menționată mai sus.

Fig. 2 și **3** ilustrează crearea celor trei tabele Users, Posts și Friends menționate mai sus cu coloanele bine definite:

```
const char *createUsers = "CREATE TABLE IF NOT EXISTS users ("
                          "    username TEXT PRIMARY KEY,"
                          "    password TEXT NOT NULL,"
                          "    type_of_user TEXT,"
                          "    type_of_profile TEXT"
                          ");";

const char *createFriends = "CREATE TABLE IF NOT EXISTS friends ("
                            "    host_username TEXT PRIMARY KEY,"
                            "    friend_username TEXT,"
                            "    type_of_friend TEXT"
                            ");";

const char *createPosts = "CREATE TABLE IF NOT EXISTS posts ("
                          "    username TEXT PRIMARY KEY,"
                          "    public_posts TEXT,"
                          "    close_friend_posts TEXT,"
                          "    friend_posts TEXT"
                          ");";
```

Fig. 2. Crearea interogărilor

Iar pe măsură ce datele de inserție a clientului sunt verificate inițial în aplicația client și ulterior în unele funcții ale aplicației server, nu este necesar să condiționăm coloanele speicifice în interogarea de creare cu NOT NULL.

```
int createTables()
{
  char *errMsg = 0;

  if (sqlite3_exec(db, createUsers, 0, 0, &errMsg) != SQLITE_OK)
  {
    fprintf(stderr, "SQL error: %s\n", errMsg);
    sqlite3_free(errMsg);
    return 1;
  }

  if (sqlite3_exec(db, createFriends, 0, 0, &errMsg) != SQLITE_OK)
  {
    fprintf(stderr, "SQL error: %s\n", errMsg);
    sqlite3_free(errMsg);
    return 1;
  }

  if (sqlite3_exec(db, createPosts, 0, 0, &errMsg) != SQLITE_OK)
  {
    fprintf(stderr, "SQL error: %s\n", errMsg);
    sqlite3_free(errMsg);
    return 1;
  }

  return 0;
}
```

Fig. 3. Execuția și crearea propriu-zisă a tabelelor din baza de date

Fig. 4 reprezintă inserarea datelor specifice unui utilizator în tabela Users în urma primirii și validării acestora, folosind funcția:

sqlite3_bind_text(sqlite3_stmt*, int, const char*, int, void(*)(void*)) care este utilizată în SQLite pentru a lega o valoare de tip text la o instrucțiune SQL pregătită. Această funcție va fi întâlnită de asemenea la interogări select, update etc.

```
const char *insertDataSQL = "insert into users values (?, ?, ?, ?);";
if (sqlite3_prepare_v2(db, insertDataSQL, -1, &statement, 0) == SQLITE_OK)
{
  sqlite3_bind_text(statement, 1, username, -1, SQLITE_STATIC);
  sqlite3_bind_text(statement, 2, passwd, -1, SQLITE_STATIC);
  sqlite3_bind_text(statement, 3, type_usr, -1, SQLITE_STATIC);
  sqlite3_bind_text(statement, 4, type_acc, -1, SQLITE_STATIC);

  if (sqlite3_step(statement) != SQLITE_DONE)
  {
    strcpy(answ, "Datele introduse sunt invalide.");
    write(fd, answ, strlen(answ));
  }
  else
  {
    strcpy(answ, "Inregistrare cu succes.");
    write(fd, answ, strlen(answ));
  }
}
```

Fig. 4. Înserarea datelor specifice în baza de date

Fig. 5 prezintă definirea structurii User despre care am vorbit în partea de implementare, iar și utilizarea componentei chat_accepted care este inițializată anterior în cazul în care există o cerere de tip chat și comanda primită de la utilizator este accept (adică denotă acceptarea cererii de chat).

```
struct User
{
  int fd;
  char username[101];
  int chat_request;
  int chat_accepted;
} users[10001];
```

```
for (int i = 0; i < n; i++)
{
  if ( userByFd(fds[i])->chat_accepted )
    {
      FD_SET(fds[i], &readfds);
      if (nfds < fds[i])
        nfds = fds[i];
      accepted ++;
    }

}
```

Fig. 5. Structura User definirea și utilizarea acesteia

În **Fig. 6** observăm definirea, inițializarea și lucrul cu tehnologia mutex:

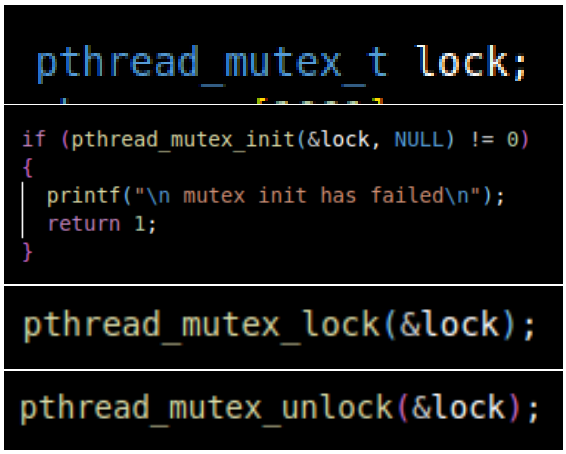

Fig. 6. Folosirea tehnologiei mutex

5 Concluzii

În concluzie, aplicația conține un set esențial de servicii și cereri astfel încât utilizatorii să poată interacționa într-un mod direct și eficient și să beneficieze de raspunsuri cât mai rapide.

Referințe bibliografice

1. www.sqlite.org/c3ref/intro.html