Offline Messenger

Ivănușcă D. Alexandra-Paula[1]

Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza" Iași

1 Introducere

În cele ce urmează, voi prezenta o metodă personală de implementare a proiectului "Offline Messenger". În cadrul său, se va realiza trimiterea de mesaje între utilizatori, simulând o aplicație de mesagerie online. Aceștia vor fi identificați printr-un username și vor putea trimite mesaje indiferent de stadiul actual al destinatarului - logat sau nu. În cazul în care destinatarul este conectat, mesajul va fi primit instant. Altfel, mesajul va fi primit la momentul următoarei conectări. Utilizatorii vor avea și alte facilități de vizualizare și manipulare a mesajelor, respectiv de logare și deconectare.

2 Tehnologii Aplicate

Implementarea utilizează un server concurent TCP cu multiplexare în care toate informațiile sunt reținute într-o bază de date.

Alegerea TCP este justificată de siguranța pe care aceasta o garantează în trimiterea intactă a mesajelor între utilizatori. Nu este necesară o distribuire live perfectă (asigurată de o eventuală implementare alternativă cu un server UDP), cu atât mai mult cu cât, dacă destinatarul nu este conectat, mesajele nu trebuie transmise prin canal de la server la destinatar.

Alegerea multiplexării este justificată de necesitatea identificării utilizatorului căruia îi este trimis mesajului. În acest sens, pentru fiecare client este realizată o corelație între *file descriptor* și *username-ul* specificat la logare.

Alegerea utilizării unei baze de date este justificată întocmai de necesitatea reținerii acestei corelări și a mesajelor transmise între utilizatori pentru notificare ulterioară, vizualizare și manipulare a mesajelor trimise anterior.

3 Structura Aplicației

La momentul deschiderii aplicației de către un nou client, vor fi afișate pe ecran următoarele comenzi posibile:

- *login : username* - un client se poate loga cu un username specificat
- *logout* - un utilizator poate ieși din cont
- *quit* - un client poate încheia sesiunea de lucru
- *send username : text* - un utilizator poate trimite un mesaj unui alt utilizator (contul trebuie să existe)

- *history : username* - un utilizator poate vizualiza istoricul conversației cu un alt utilizator
- *reply username nrMsg : text* - un utilizator poate răspunde la un mesaj anume primit de la alt utilizator, identificat prin indexul din cadrul conversației
- *delete : username nrMsg* - un utilizator poate șterge textul oricărui mesaj trimis de către el, identificat prin indexul din cadrul conversației

Baza de date din spate are următoarea structură:

Table 1: messenger.db

(a) users

Name	NULL?	Type
username	NOT NULL	TEXT
fd	NULL	INTEGER

(b) conversatii

Name	NULL?	Type
id	NOT NULL	INTEGER
expeditor	NOT NULL	TEXT
destinatar	NOT NULL	TEXT
text	NOT NULL	TEXT
data	NOT NULL	TEXT
ora	NOT NULL	TEXT
seen	NOT NULL	INTEGER
reply	NOT NULL	INTEGER

Mesajele trimise sunt salvate în tabela *conversații*, unde expeditorul și destinatarul sunt variabile de tipul username din tabela *users*.

Coloana *reply* este o variabilă ce reprezintă numărul mesajului din conversație la care s-a dat reply sau are valoarea 0 pentru ca, la afișarea istoricului, să putem specifica la ce mesaj a fost dat reply. Valorile din coloana *reply* nu coincid cu valorile din coloana *id*, întrucât în baza de date mesajele din mai multe conversații sunt amestecate.

Coloana *seen* are valoarea 1 în cazul în care mesajul a fost citit și 0 în caz contrat. Utilitatea acestei coloane este justificată de necesitatea afișării mesajelor primite în timpul în care user-ul a fost deconectat. Ne vom folosi de ea și în afișarea istoricului, pentru a arăta dacă interlocutorul a văzut sau nu mesajele trimite de user-ul care a solicitat vizualizarea conversației.

O reprezentarea vizuală a modului de funcționare a aplicației ar fi după cum urmează:

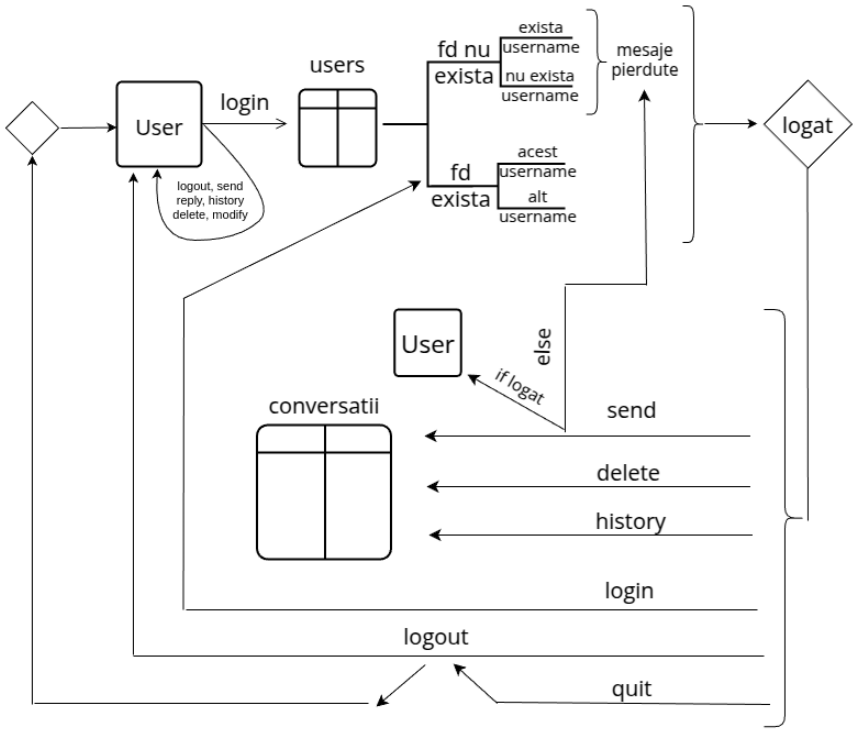

Fig. 1: Diagrama aplicației

4 Aspecte de implementare

Clientul are o structură destul de simplă. În el este verificată corectitudinea argumentelor primite de la tastatură, sunt afișate comenzile posibile, este realizată conexiunea cu server-ul printr-un *socket* și se apelează comanda *fork()*.

Necesitatea unui socket pentru comunicarea între server și client provine din schimbul bidirecțional de mesaje dintre aceștia, care stă în spatele comunicării dintre 2 clienți.

Socket-ul este identificat printr-un *port* cu o valoarea dată în server, pe care clientul trebuie să o introducă de la tastatură pentru a se conecta la aplicație.

```
int creare_socket()
{
    int sd = socket(AF_INET, SOCK_STREAM, 0);
    if (sd == -1)
    {
        perror("Eroare la socket()");
        exit(errno);
    }
    return sd;
}

struct sockaddr_in initializare_server(int port, int addr)
{
    struct sockaddr_in server;
    bzero(&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(addr);
    server.sin_port = htons(port);
    return server;
}

void conectare_server(struct sockaddr_in *server)
{
    if (connect(sd, (struct sockaddr *)server, sizeof(struct
        sockaddr)) == -1)
    {
        perror("Eroare la connect()");
        exit(errno);
    }
}
```

Necesitatea apelului *fork()* este justificată de cei doi pași din comunicarea între clienți:

- clientul citește de la tastatură și transmite textul serverului (proces tată)
- clientul primește un mesaj de la server și îl afișează pe ecran (proces fiu)

```
void proces_tata(int sd)
{
    char msg[LMAX] = "";
    signal(SIGINT, handle_quit);
    signal(SIGTERM, handle_parinte);
    while (1)
    {
        bzero(msg, LMAX);
```

```
9           apelul_read(0, msg);
10          apelul_write(msg);
11          if (strcmp(msg, "quit\n") == 0)
12          {
13              kill(pid, SIGTERM);
14              shutdown(sd, SHUT_RDWR);
15              close(sd);
16              break;
17          }
18      }
19  }
20
21  void proces_fiu(int sd)
22  {
23      char msg[LMAX] = {};
24      signal(SIGTERM, handle_fiu);
25      while (1)
26      {
27          bzero(msg, LMAX);
28          apelul_read(sd, msg);
29          if (strcmp(msg, "quit") == 0)
30          {
31              kill(getppid(), SIGTERM);
32              shutdown(sd, SHUT_RDWR);
33              close(sd);
34              break;
35          }
36          printf("%s", msg);
37      }
38  }
```

Apelul *kill* din procesul tată este pentru cazul în care clientul citește de la tastatură comanda *quit* și transmite un semnal procesului fiu pentru a fi omorât prin funcția *handle fiu* ca să nu rămână orfan. Aceeași idee este aplicată și în cazul semnalului transmis prin CTRL+C, funcția *handle quit* trimițând comanda *quit* către server pentru a fi deconectat.

Apelul *kill* din procesul fiu este pentru cazul în care serverul citește de la tastatură comanda *quit* printr-un apel non-blocant sau combinația de taste CTRL+C, care are următorul efect:

- se transmite mesajul *quit* către toții clienții și apoi sunt șterși din lista de comunicare a serverului
- mesajul *quit* este citit de către clienți în procesele fii
- procesele fii transmit un semnal *kill* către părinte prin funcția *handle parinte* și apoi închid *socket*-ul
- este închis serverul

Serverul este complex. Crearea socket-ului este identică, însă el trebuie să verifice continuu dacă primește vreun mesaj de la oricare dintre clienți, identificați printr-un file descriptor.

Tot în server este creată baza de date care cuprinde cele două tabele anterior menționate.

```
void creare_baza_date()
{
    char *error = 0;
    rc = sqlite3_open("messenger.db", &db); vf_rc_ok();

    const char *sql_create_table_users =
            "CREATE TABLE IF NOT EXISTS users (" \
            "username TEXT PRIMARY KEY," \
            "fd INTEGER);";
    rc = sqlite3_exec(db, sql_create_table_users, 0, 0, &
        error); vf_sql_ok(error);

    const char *sql_create_table_conv =
            "CREATE TABLE IF NOT EXISTS conversatii (" \
            "id INTEGER PRIMARY KEY AUTOINCREMENT," \
            "expeditor TEXT NOT NULL," \
            "destinatar TEXT NOT NULL," \
            "text TEXT NOT NULL," \
            "data TEXT NOT NULL," \
            "ora TEXT NOT NULL," \
            "seen INTEGER NOT NULL," \
            "reply INTEGER NOT NULL);";
    rc = sqlite3_exec(db, sql_create_table_conv, 0, 0, &error
        ); vf_sql_ok(error);
}
```

Lucrul cu baza de date se realizează prin funcțiile din librăria *sqlite3.h* pentru comenzi de tipul SELECT, INSERT, UPDATE.

Dintre toate funcțiile, poate cea mai importantă este cea pentru introducerea unui mesaj în baza de date, întrucât este vorba despre o aplicație de mesagerie.

```
void salvare_mesaj(char *expeditor, char *destinatar, char *
    text, int seen, int reply)
{
    sqlite3_stmt *stmt;
    const char *sql_insert =
            "INSERT INTO conversatii" \
            "(expeditor, destinatar, text," \
            " data, ora, seen, reply)" \
            "VALUES (?, ?, ?," \
            "STRFTIME('%d-%m-%Y', 'now', 'localtime')," \
            "STRFTIME('%H:%M:%S', 'now', 'localtime')," \
            "?, ?);";
    rc = sqlite3_prepare_v2(db, sql_insert, -1, &stmt, 0);
    vf_rc_ok();
    sqlite3_bind_text(stmt, 1, expeditor, -1, SQLITE_STATIC);
    vf_rc_ok();
```

```
16      sqlite3_bind_text(stmt, 2, destinatar, -1, SQLITE_STATIC)
            ;
17      vf_rc_ok();
18      sqlite3_bind_text(stmt, 3, text, -1, SQLITE_STATIC);
19      vf_rc_ok();
20      sqlite3_bind_int(stmt, 4, seen);
21      vf_rc_ok();
22      sqlite3_bind_int(stmt, 5, reply);
23      vf_rc_ok();
24      rc = sqlite3_step(stmt);
25      vf_rc_done();
26      sqlite3_finalize(stmt);
27  }
```

Aplicația necesită următoarele apeluri din terminal pentru a fi folosită:

- *gcc -Wall server.c -o server -lsqlite3*
  - *./server*
- *gcc -Wall client.c -o client*
  - *./client 0 2727*
  - *./client 0 2727*
  - ...

Pentru aplicarea oricărei comenzi, este necesar ca clientul să fie logat la un cont. Comanda *login* realizează o multitudine de verificări pentru care primește răspuns de la server:

- clientul este deja logat, dar cu alt username
- clientul este deja logat la username-ul specificat
- clientul nu este logat, dar un alt client este logat la username-ul specificat
- clientul nu este logat și niciun alt client nu este logat la username-ul specificat
  - dacă niciun client nu s-a mai conectat până acum cu acest username, îl adăugam în baza de date, creându-se astfel un cont nou de utilizator
  - dacă utilizatorul a mai fost conectat (i.e. se află deja în baza de date), atunci programul verifică dacă a primit mesaje în timpul în care a fost deconectat, în caz afirmativ afișându-le pe ecran

Comenzile *logout* și *quit* manipulează, de asemenea, proprietatea de conectivitate a clienților la server. Ele sunt strâns legate între ele, a doua cuprinzând-o pe prima.

- dacă un client este logat, îl deconectează (în cazul comenzii *logout*, este afișat un mesaj adecvat)
- *file descriptor*-ul este eliberat din listă
- este închis socket-ul

Alternativ, utilizatorul poate forma combinația de taste CTRL+C care transmite tot comanda *quit* către server.

Comanda *send* trimite un mesaj unui alt utilizator, făcând următoarele teste:

- dacă clientul nu este logat, se afișează un mesaj corespunzător
- dacă clientul este logat:
  - dacă username-ul către care se dorește a fi trimis mesajul nu există, se afișează un mesaj corespunzător
  - dacă username-ul există:
    - mesajul este salvat în baza de date
    - dacă utilizatorul este logat, mesajul este afișat în fereastra sa
    - altfel, pe coloana *seen* este marcat cu 0, iar mesajul va apărea la următoarea conectare a utilizatorului

Comanda *history* este foarte importantă pentru următoarele comenzi. Ea face următoarele verificări:

- dacă clientul nu este logat, se afișează un mesaj corespunzător
- dacă clientul este logat:
  - dacă nu ai început încă o conversație cu username-ul specificat, se afișează un mesaj corespunzător
  - în caz contrar, mesajele sunt afișate pe rând cu următoarea structura:
    - numărul mesajului în cadrul conversației
    - interlocutor / eu : text
    - (ne)citit (pentru mesajele trimise de user-ul care a solicitat vizualizarea conversației)
    - dacă este reply la un mesaj de mai sus

În apelul comenzii *reply* trebuie specificat numărul mesajului din cadrul conversației, cu următoarele precizări:

- dacă clientul nu este logat, se afișează un mesaj corespunzător
- dacă clientul este logat:
  - dacă nu există un cont cu username-ul specificat, se afișează un mesaj corespunzător
  - dacă există username-ul specificat în baza de date:
    - dacă mesajul specificat nu există sau nu a fost scris de tine, se afișează un mesaj corespunzător
    - în caz contrar:
      - mesajul este salvat în baza de date
      - dacă utilizatorul este logat, mesajul este afișat în fereastra sa
      - altfel, pe coloana *seen* este marcat cu 0, iar mesajul va apărea la următoarea conectare a utilizatorului

Comanda *delete* are o logică asemănătoare, însă nu este trimis un mesaj nou, ci textul unui mesaj vechi este înlocuit cu *Acest mesaj a fost șters.*

Utilizatorul poate închide aplicația într-un mod foarte simplu, scriind comanda *quit* sau formând combinația de taste CTRL+C în terminalul serverului.

5 Concluzii

Proiectul prezintă o paletă largă de îmbunătățiri, care ar beneficia experiența utilizatorilor și l-ar apropia de formatul unei aplicații de mesagerie reale.

O posibilă idee este trimiterea mesajelor către mai mulți utilizatori simultan prin extinderea comenzii *send* care să permită o astfel de structură : *send username1 username2 ... : text.*

O îmbunătățire mai complexă este crearea grupurilor printr-o comandă *group username1 username2 ... : NameOfTheGroup* și de a trimite mesaje către acestea printr-o variantă dezvoltată a comenzii *send username : text*, care să caute username-ul și în lista grupurilor (extindere și pentru comenzile *history*, *reply*, *delete*, *modify*). Ar trebui adăugată o nouă tabelă *groups* în care să reținem corelația dintre *username* și *group*. În acest context, nu putem reține în coloana *seen* dacă un user a văzut sau nu un mesaj trimis pe grup, deoarece mesajul va apărea o singură dată în tabela conversației. Pentru a rezolva această problemă, am putea reține în tabela *users*, pe două coloane noi, data și ora ultimei deconectări a fiecărui utilizator. Dacă mesajul a fost trimis după deconectare, el trebuie să apară ca notificare la prima conectare. Bineînțeles, aceste coloane trebuie actualizate la fiecare aplicare a comenzii *logout* de către utilizatori. Alte comenzi interesante pentru grupuri ar fi:

- *SeeMembers : NameOfTheGroup*
- *RenameGroup : NameOfTheGroup*
- *DeleteGroup : NameOfTheGroup*
- *DeleteFromGroup NameOfTheGroup : username*
- *AddToGroup NameOfTheGroup : username*
- *LeaveGroup : NameOfTheGroup*
- *MakeAdmin NameOfTheGroup : username*
- *RemoveAdmin NameOfTheGroup : username*

6 Referințe Bibliografice

1. Andrei Scutelnicu, Computer Networks, https://www.andreis.ro/teaching/computer-networks
2. SQLite Tutorial, https://www.tutorialspoint.com/sqlite/index.htm
3. Overleaf Documentation, https://www.overleaf.com/learn
4. Visual Paradigm, https://online.visual-paradigm.com