Offline Messenger

Enache Mara-Georgiana

Universitatea Alexandru Ioan Cuza, Iași, România, Facultatea de Informatică
mara.enache@info.uaic.ro

1 Introducere

Această fișă de raport are rolul de a prezenta mai multe detalii despre proiectul pe care doresc să îl realizez în cadrul disciplinei Rețele de calculatoare. Proiectul ales, Offline Messenger, implică dezvoltarea unei aplicații de tipul client/server, ce permite schimbul de mesaje între utilizatorii conectați.

Funcționalitatea aplicației constă în conectarea clienților la server și executarea comenzilor de care vor dispune odată cu logarea în aplicație (onlineUsers, chat username, getInbox help etc.). Pentru a putea trimite un mesaj, utilizatorul trebuie să fie logat, dar utilizatorul căruia ii trimite mesajul poate fi online sau offline.

Fiecare utilizator ce primește un mesaj poate răspunde mesajului (reply) sau îl poate ignora. Caz special: dacă un utilizator neconectat primește un mesaj, acesta îl va putea vizualiza doar la logarea sa. Serverul îi va permite clientului să răspunde unor anumite mesaje primite (selectate prin identificatori). Pentru acest lucru trebuie sa intre in converstie cu un utilizator (comanda chat username) si astfel se vor afisa mesajele si identificatorii. lor. Dupa cu ajutorul comenzii replyto id mesaj se va trimite mesajul raspuns in conversatie. Comanda refresh permite actualizarea conversatiei.

2 Tehnologii utilizate

TCP: Proiectul va avea la bază protocolul TCP orientat-conexiune, care permite realizarea unei comunicații full duplex sigure între 2 puncte terminale. Protocolul corespunde nivelului transport din stiva TCP/IP, ce oferă mecanisme de control al fluxului, astfel asigurându-se că datele vor fi transmise în ordine.

Principalele concepte utilizate în cadrul proiectului:

- **socket:** mecanism bidirecțional, utilizat pentru a asigura comunicarea în rețea. În proiect conexiunea dintre clienți și server se va realiza folosind socket. Fiecare conexiune va fi definită de o adresă IP și de un port.
- **thread:** am optat pentru utilizarea threadurilor și nu a creării unor procese copil cu scopul de a eficientiza implementarea. Astfel, nu se creează procese noi pentru fiecare utilizator conectat, întrucât threadurile asociate fiecărui client și procesul principal împart aceeași memorie și nu se va aloca memorie suplimentară, pid. Totodată, se vor evita erorile cauzate de caracterul blocant al funcțiilor accept(), getchar() etc.

Motivele utilizării tehnologiei TCP:

• Stabilirea unei conexiuni full-duplex între client și server: Este necesară în program la conectarea utilizatorului. Odată ce un client se va conecta la serverul aplicației, acesta va stabili un port pentru conexiune și se va crea un thread cu un id unic ce va prelua datele introduse de utilizator și le va transmite la server. Serverul le prelucrează și oferă înapoi un răspuns pe baza comenzii introduse de utilizator.

• Transmiterea sigură și în ordine a pachetelor între clienți și server: Un alt aspect important îl constituie siguranța că utilizatorul va primi în întregime răspunsul de la server, respectiv de la ceilalți utilizatori, având confirmare de primire a datelor.

• Servirea concurentă și rapidă a clienților: Serverul TCP va crea câte un thread pentru fiecare client în așa fel încât va exista posibilitatea servirii mai multor clienți în mod simultan. Proprietate utilizată în cazul schimbului de mesaje între utilizatorii conectați.

Limbajul ales pentru realizarea proiectului este C.

3 Arhitectura aplicației

3.1 Proiectarea modelului CLIENT/SERVER TCP

Aplicația va folosi o arhitectură de tipul client/server TCP concurent. În această arhitectură serverul are unul sau mai mulți clienți conectați printr-o adresă și un port. Când clientul îi trimite serverului o cerere de informație, serverul acceptă cererea, o procesează și îi livrează înapoi pachetele cerute. Aceasta comunicare este de tipul cerere-răspuns (Fig.1). Una dintre caracteristicile cele mai importante este ca serverul se poate ocupa de mai multi clienti în acelasi timp.

Serverul va fi implementat folosind tehnologia TCP – prethreaded, cu blocare pentru protecția accept(). Serverul creează un număr de thread-uri când este pornit și aceastea vor servi clienții. Pentru ca doar un singur thread sa apeleze accept la un moment dat, se va folosi mutex lock pe apelul primitivei accept. Serverul va ține evidenta numărului de thread-uri active și va creea noi thread-uri când numărul clienților se apropie de numărul total al thread-uilor create. În cazul proiectului este necesară generarea mai multor threaduri cu scopul de a accepta mai multe cereri de la mai multi utilizatori în același timp. (Fig.2, Fig.3)

În momentul când un astfel de thread nou creat termină de servit clientul, el își va termina execuția, anunțând procesul principal.

Clientul va fi implementat folosind tehnologia TCP. Aplicația-client ii cere utilizatorului sa se logheze și apoi, în cazul unei acceptari din partea serverului, clientul va putea introduce comenzile la care are acces.(Fig.4, Fig.5, Fig.6) Utilizatorul va putea introduce altă comandă după primirea răspunsului de la server. Încheierea sesiunii se va face prin comanda exit.(Fig.7)

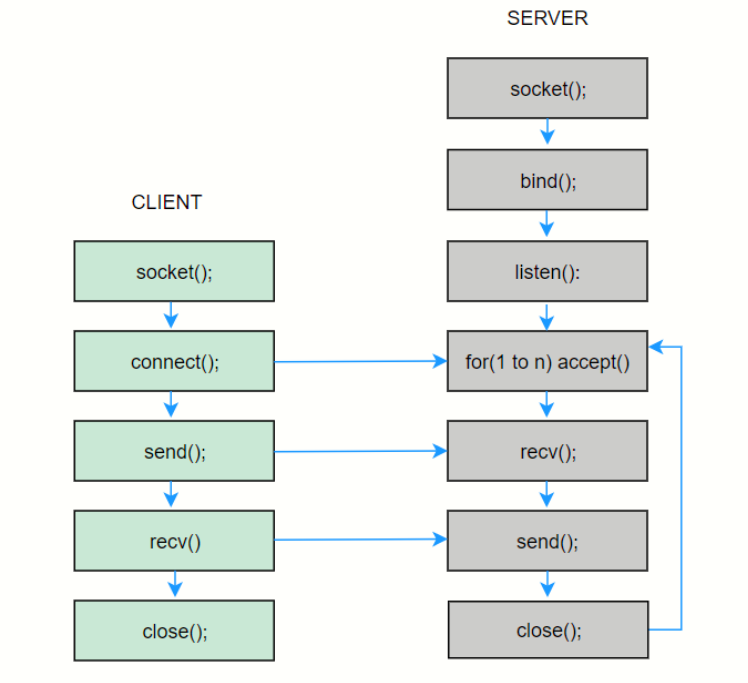


Fig. 1. Modelul client/server TCP concurent

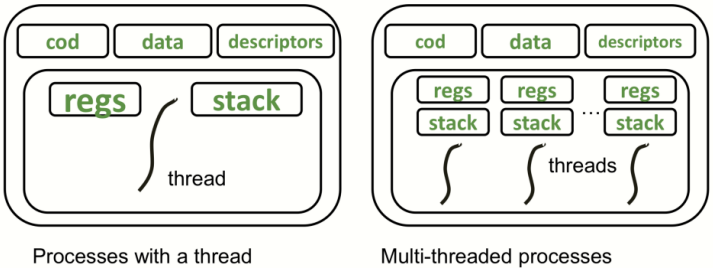


Fig. 2. thread

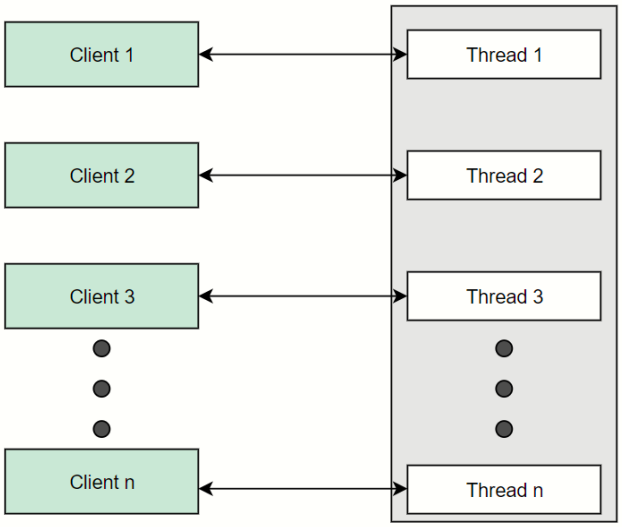


Fig. 3. Multithreaded Server

3.2 Stocarea datelor. Schema modulelor funcționale

O versiune îmbunătită a codului (pentru o implementare mai eficienta) va crea o legătură între limbajul C și baza de date SQLITE.Informatiile precum ar fi utilizatorii existenti, si mesajele trimise de acestia vor fi memorate intr-o baza de date . Acesta are rolul de a simplifica căutările. În proiect baza de date va conține tabele:

- tabela cu utilizatori, ce va avea câmpurile username, password, status
- tabela cu mesaje, ce va avea câmpurile utilizator,mesaj,id, destinatar

Scenariu de utilizare: Un utilizator cu username-ul A dorește să se conecteze la aplicație. Acesta când va utiliza comanda de login, aplicația va căuta în baza de date, în tabela cu toți utilizatorii inregistrați numele acestuia. Dacă il găsește, îl va lăsa să folosească și celelalte comenzi; În caz contrar, se va returna un mesaj de tipul: "username greșit sau username neînregistrat".

Mai jos se pot vedea mai multe diagrame în care se prezintă arhitectura aplicației.

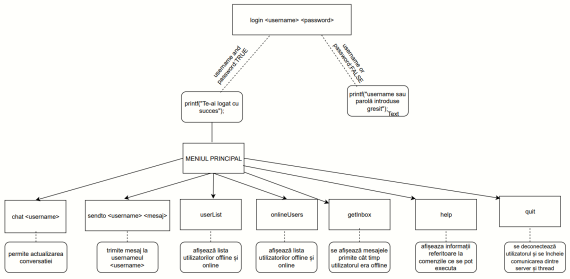


Fig. 4. Diagramă comenzilor disponibile pentru

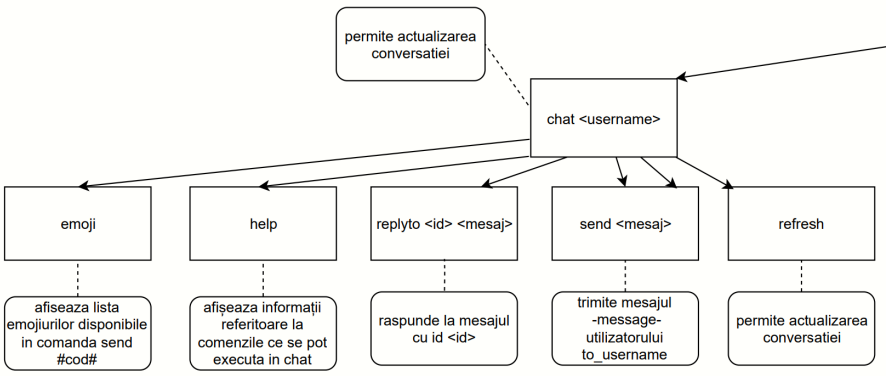


Fig. 5. Diagrama comenzilor utilizatorului aflat intr-o conversatie

4 Detalii de implementare

Detaliile de implementare sunt într-o stransă legatură cu diagramele de mai sus(Fig.4-7). Am construit în SQLITE o bază de date formată din tabele utilizatori și mesaje. Fiecare comandă ce poate fi introdusa ca input are asociată în program o funcție. Sunt mai multe detalii în subsecțiunile ce urmează.

```
void dbmessagesInitialize()
{
  int database_descriptor = sqlite3_open("database.db", &database);
  //daca nu exista
  sprintf(sql, "SELECT count(*) FROM sqlite_master WHERE type='table' AND name='messages';");
  database = sqlite3_exec(database, sql, callback, str, &error_message);
  //printf("\n STR count =====%s===\n",str);

  if(strcmp(str,"0\n")==0)
  {
    int database_descriptor = sqlite3_open("database.db", &database);
    sprintf(sql, "CREATE TABLE messages (id_mesaj INTEGER, sender TEXT, receiver TEXT, message TEXT, seen TEXT, infoplus TEXT);");
    database_descriptor = sqlite3_exec(database, sql, 0, 0, & error_message);
    if(database_descriptor!=SQLITE_OK)
        printf("Eroare la creare tabel");
  }
}
```

```c
void databaseInitialize()
{
    int database_descriptor = sqlite3_open("database.db", &database);
    sprintf(sql, "SELECT count(*) FROM sqlite_master WHERE type='table' AND name='users';");
    database = sqlite3_exec(database, sql, callback, str, &error_message);
    if(strcmp(str,"0\n")==0)
    {
      int database_descriptor = sqlite3_open("database.db", &database);
      sprintf(sql, "CREATE TABLE users (ID INTEGER,username TEXT,password TEXT,status TEXT);");
      database_descriptor = sqlite3_exec(database, sql, 0, 0, & error_message);
      if(database_descriptor!=SQLITE_OK)
        printf("Eroare la inserarea inregistrarilor");

      sprintf(sql, "INSERT INTO users (ID,username,password,status) VALUES (1,'%s','%s','%s');",
       "maraenache","pass1234","online");
      ret= sqlite3_exec(database, sql, 0, 0, & error_message);
      sprintf(sql, "INSERT INTO users (ID,username,password,status) VALUES (2,'%s','%s','%s');",
      "andreea","12345678","online");
      ret = sqlite3_exec(database, sql, 0, 0, & error_message);
      sprintf(sql, "INSERT INTO users (ID,username,password,status) VALUES (3,'%s','%s','%s');",
       "maria","maria222","online");
      ret = sqlite3_exec(database, sql, 0, 0, & error_message);
      sprintf(sql, "INSERT INTO users (ID,username,password,status) VALUES (4,'%s','%s','%s');",
      "bianca","lalala","offline");
      ret = sqlite3_exec(database, sql, 0, 0, & error_message);
      sprintf(sql, "INSERT INTO users (ID,username,password,status) VALUES (5,'%s','%s','%s');",
       "darius","copac787","offline");
      ret = sqlite3_exec(database, sql, 0, 0, & error_message);
      sprintf(sql, "INSERT INTO users (ID,username,password,status) VALUES (6,'%s','%s','%s');",
      "sebi","09882138","online");
      ret = sqlite3_exec(database, sql, 0, 0, & error_message);
      if(ret!=SQLITE_OK)
          printf("Eroare la inserarea inregistrarilor");
    }
```

4.1 Comanda login

Scopul comenzii este de a i se permite unui utilizator care este deja înregistrat în baza de date să se conecteze în contul sau. Se ocupă cu logarea cu un username și o parolă la server pe portul specificat. Dacă acesta a introdus corect informațiile va avea acces la meniu, în caz contrar se permite reîncercarea logării sau opțiunea de a ieși (prin exit).

```
rdc@rdc-VirtualBox:~/nou$ ./client 127.0.0.1 2228
Bine ai venit!👋
 Introduceti username si parola pt conectare!
SINTAXA login <username> <password>
login maraenache pass1234

Ati introdus comanda:login maraenache pass1234

Te-ai logat cu succes!✅

🌟MENIUL PRINCIPAL!🌟

Aveti urmatoarele optiuni de comenzi:
        1)usersList
        2)onlineUsers
        3)sendto <username> <mesaj>
        4)getInbox
        5)chat <username>
        6)logout
        7)help
        8)quit
```

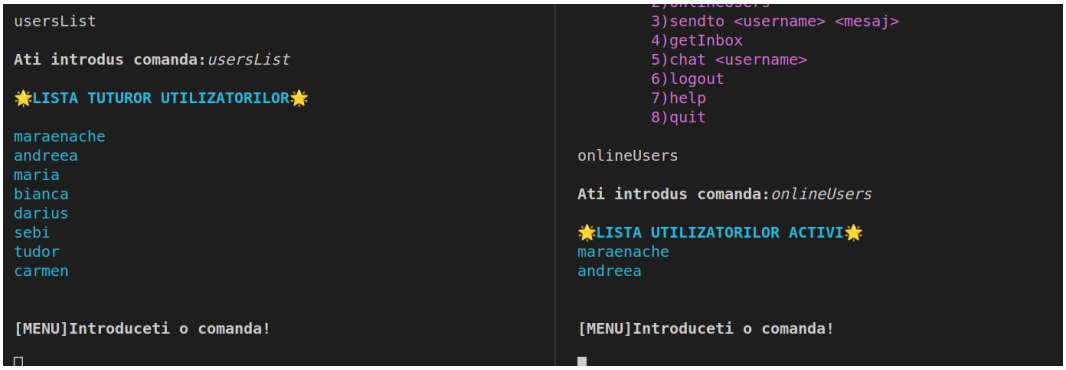
```
rdc@rdc-VirtualBox:~/nou$ ./client 127.0.0.1 2228
Bine ai venit!👋
 Introduceti username si parola pt conectare!
SINTAXA login <username> <password>
login maraenache parolagresita

Ati introdus comanda:login maraenache parolagresita

❌Nume utilizator sau parola au fost introduse gresit! I
ncercati din nou
login andreea 12345678

Ati introdus comanda:login andreea 12345678

Te-ai logat cu succes!✅

🌟MENIUL PRINCIPAL!🌟

Aveti urmatoarele optiuni de comenzi:
        1)usersList
        2)onlineUsers
        3)sendto <username> <mesaj>
        4)getInbox
```

4.2 Funcția menu()

În fucția menu, citesc mesajul de la client care este de tipul: comanda - usersList, getInbox, help sau quit-. În funcție de opțiune va fi direcționat de mesaje pentru a reuși să facă ce și-a propus. În funcție de comanda primită, se apeleaza funcțiile. Orice funcție din meniu odată executata permite întoarcerea la meniu.

4.3 Comenzile usersList si onlineUsers

Comanda usersList are scopul de a afișa lista tuturor utilizatorilor înregistrați în aplicație, indiferent de statusul lor(offline/ online) prin care se va putea alege în conversația cărui utilizator vreau să intru.Comanda usersList are scopul de a afișa lista utilizatorilor online. După execuția acestor comenzi se rămâne în meniu.

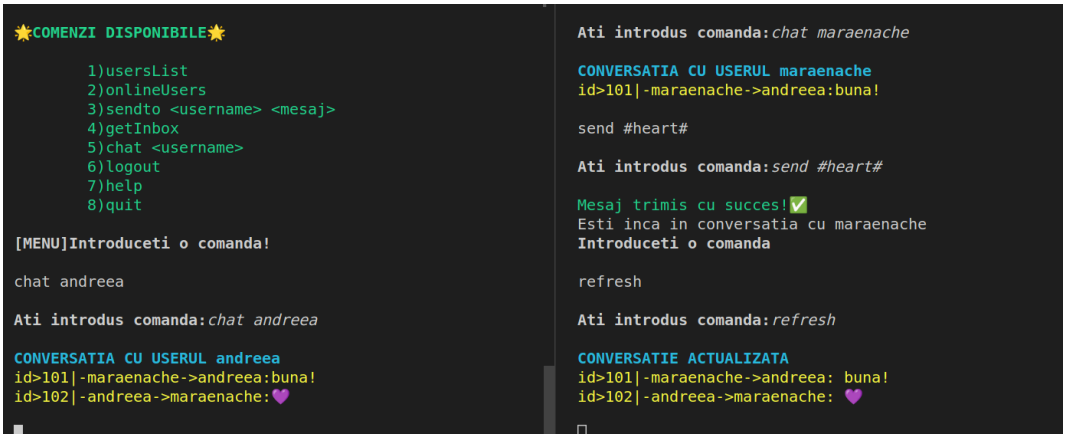


4.4 Funcția getInbox

Funcția permite vizualizarea mesajelor primite în timpul în care utilizatorul nu era conectat; după execuția acesteia se rămâne în meniu. Comanda poate fi utilizată doar în momentul în care utilizatorul se află in meniul principal.

4.5 Comanda help

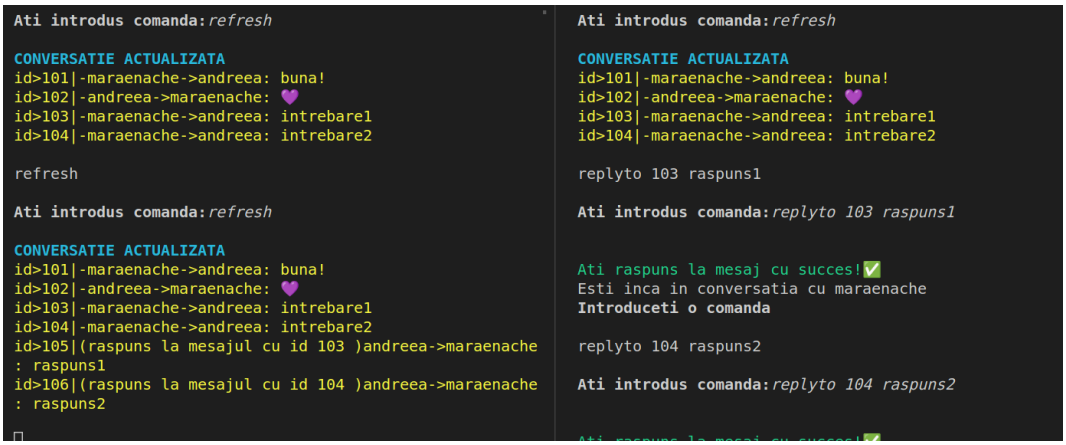
Scopul acestei comenzi este de a afișa informații referitoare la comenzile care pot fi introduse de un utilizator conectat. Această comandă poate fi utilizată doar în momentul în care utilizatorul se află in meniul principal. După executarea ei, după afișarea opțiunilor disponibile, utilizatorul rămâne în meniul principal.



4.7 Funcția send si replyto

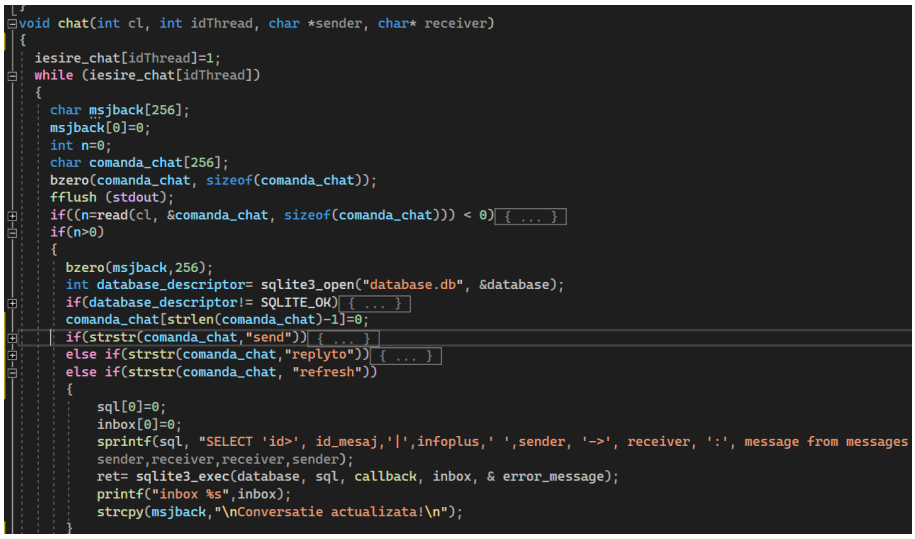
Clientul va trimite la server un mesaj. Cum functia chat a fost apelata anterior, serverul cunoaște destinatarul mesajului. Serverul va insera noi înregistrări în tabelul **messages** din baza de date, astfel: pentru câmpul mesaj se va introduce

mesajul, pentru câmpul destinatar se va insera numele utilizatorului selectat la comanda chat user, si pentru id va fi generat un număr unic. Fiecare utilizator ce primește un mesaj poate răspunde mesajului(replyto) sau îl poate ignora. Serverul îi va permite clientului posibilitatea răspunde unor anumite mesaje primite (selectate prin câmpul id din tabelul mesaje).



4.8 Funcția chat

Odata ajuns in chat cu un user, se pot folosi comenzile send, pe a i trimite un mesaj, replyto pentru a raspunde, emoji pt a vedea lista emojiurilor disponibile in comanda send si backtomenu pentru a reveni in meniul principal.



```
void chat(int cl, int idThread, char *sender, char* receiver)
{
  iesire_chat[idThread]=1;
  while (iesire_chat[idThread])
  {
    char msjback[256];
    msjback[0]=0;
    int n=0;
    char comanda_chat[256];
    bzero(comanda_chat, sizeof(comanda_chat));
    fflush (stdout);
    if((n=read(cl, &comanda_chat, sizeof(comanda_chat))) < 0){ ... }
    if(n>0)
    {
      bzero(msjback,256);
      int database_descriptor= sqlite3_open("database.db", &database);
      if(database_descriptor!= SQLITE_OK){ ... }
      comanda_chat[strlen(comanda_chat)-1]=0;
      if(strstr(comanda_chat,"send")){ ... }
      else if(strstr(comanda_chat,"replyto")){ ... }
      else if(strstr(comanda_chat, "refresh"))
      {
          sql[0]=0;
          inbox[0]=0;
          sprintf(sql, "SELECT 'id>', id_mesaj,'|',infoplus,' ',sender, '->', receiver, ':', message from messages
          sender,receiver,receiver,sender);
          ret= sqlite3_exec(database, sql, callback, inbox, & error_message);
          printf("inbox %s",inbox);
          strcpy(msjback,"\nConversatie actualizata!\n");
      }
```

5 Concluzii

Proiectul ales, Offline Messenger, oferă funcționalitățile de baza ale unei aplicatii de chat. Modul de implementare a serverului oferă servirea rapidă, concurentă și dinamică a clientilor, iar folosirea tehnologiei TCP asigura siguranța transferului de date. Prin realizarea acestui proiect consider că voi înțelege mai bine conceptele ce stau la baza networkingului. Tehnologia aleasa ar fi putut fi îmbunătățită, din punctul de vedere al vitezei folosind un server UDP, dar considerând faptul siguranța trimiterii mesajelor este mai importantă, am optat pentru modelul TCP.

Idei pentru îmbunătățirea proiectului La nivelul proiectului, se pot realiza anumite optimizări astfel încât experiența de utilizare a aplicației să fie una mai apropiata aplicației originale (Messenger).

- înregistrarea în cazul în care username-ul introdus nu se află în baza de date.
- crearea de grupuri.
- trimiterea unui mesaj către mai multe persoane.
- ștergerea unui mesaj din conversație-ștergerea din tabela mesaje.
- căutarea unui mesaj
- schimbarea parolei

6 Bibliografie

Site-uri utilizate:

http://www.overleaf.com
https://www.diagrameditor.com
https://profs.info.uaic.ro/ computernetworks/cursullaboratorul. php
https://www.diagrameditor.com
https://medium.com/client-server-architecture-
https://profs.info.uaic.ro/ computernetworks/S12/cliTcpNr.c
https://profs.info.uaic.ro/ computernetworks/S12/servTcpConcTh2.c
https://www.overleaf.com/learn/latex/Inserting$_{I}mages$
https://www.lifesize.com/blog/tcp-vs-udp/
https://www.oracle.com/java/technologies/jpl2-socket-communication.html
https://math.hws.edu/javanotes-swing/c12/s4.html
https://www.w3.org/People/Frystyk/thesis/multithread.html
https://www.tutorialspoint.com/sqlite/sqlite$_{cc}pp.htm$
https://stackoverflow.com/questions/28602168/mysql-insert-statement-in-c
https://stackoverflow.com
http://www.mysqltutorial.org