Rețele de calculatoare
RAPORT TEHNIC - Offline Messenger
Tema 2

Moisă Iulia-Elena (2A6)

Universitatea Alexandru Ioan Cuza, Iași
Facultatea de Informatică
iulia.moisa@info.uaic.ro

Offline Messenger

Să se dezvolte o aplicație client/server care să permită schimbul de mesaje între utilizatori care sunt conectați și să ofere funcționalitatea trimiterii mesajelor și către utilizatorii offline, acestora din urmă apărându-le mesajele atunci când se vor conecta la server. De asemenea, utilizatorii vor avea posibilitatea de a trimite un răspuns (reply) în mod specific la anumite mesaje primite. Aplicația va oferi și istoricul conversațiilor pentru și cu fiecare utilizator in parte.

1 Introducere

Proiectul *Offline Messenger* permite transmiterea mesajelor între utilizatorii înregistrați, aplicația având ca fundament comunicarea de tip client/server. Utilizatorii conectați vor putea comunica direct unii cu ceilalți, spre deosebire de cei offline care vor primi mesajele abia în momentul conectării la server.

Mesajele sunt stocate într-o bază de date, existând posibilitatea de a accesa istoricul conversațiilor pentru fiecare utilizator. Totodată, anumite mesaje primite pot avea atașate răspunsuri predefinite, utilizatorul putând răspunde în mod specific la acestea.

Astfel, utilizatorii se pot înregistra pe server cu un username unic și o parolă, apoi se pot loga, ulterior putând să acceseze toate funcționalitățile.

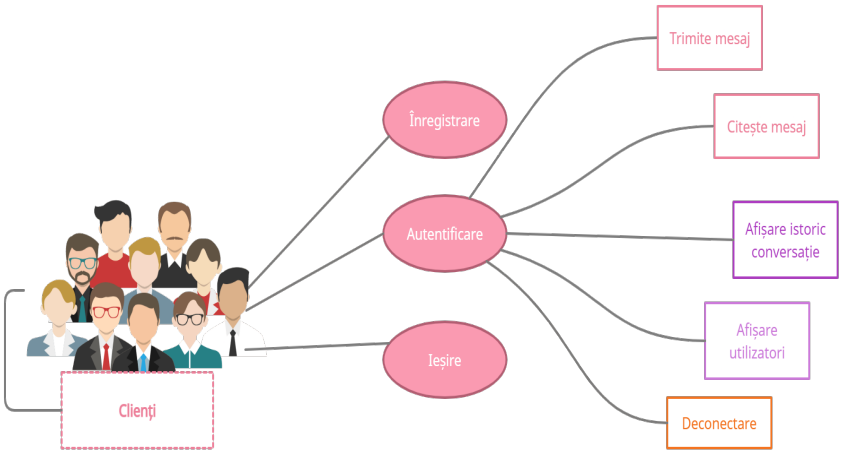
2 Tehnologii utilizate

- La baza proiectului există modelul de tip server-client. Acest model asigură paratajarea de resurse. Clientul inițiază conexiunea către server.Serverul se ocupă de coordonarea clienților, asigurând serviciile necesare pentru fiecare dintre aceștia. Comunicarea dintre server și client se realizează prin socketuri, pentru a se trimite corespunzător fluxul de mesaje și informații.
- Drept protocol de comunicare am ales TCP (Transmission Control Protocol), deoarece scopul principal al acestuia este de a controla transferul de date în

așa fel încât acesta să fie de încredere; fiind vorba de o aplicație de comunicare, mă interesează în primul rând ca mesajele să ajungă la destinație fără să se piardă informații, iar calitatea să fie maximă.
Acest protocol este orientat conexiune si bidirecțional: putem scrie și citi în ambele direcții ale conexiunii concomitent (full-duplex).

- Serverul folosit este unul de tip concurent, creându-se câte un thread pentru fiecare client în parte.Pentru a crea si manipula threadurile, programul utilizeaza standardul Pthreads (Posix Threads).
  Un thread (fir de executie) reprezinta cea mai mica sectiune de cod ce poate fi administrata de SO. Am ales threadurile datorită avantajelor din punct de vedere al timpului și al resurselor; acestora nu li se aloca memorie, PID, etc., fiind numite si *lightweight processes*.
  Threadurile si procesul principal impart aceeasi memorie, astfel ca variabilele de la nivelul procesului sunt accesibile oricarui thread.
- Datele vor fi stocate într-o bază de date SQLite care va cuprinde mai multe tabele, printre care *UtilizatoriÎnregistrați, UtilizatoriAutentificați, Mesaje, MesajeNoi*. Am ales această bază de date deoarece interacționează facil cu limbajul ales, fiind totodată potrivită pentru aplicații de acest gen.
  Serverul este cel care are legătură cu baza de date, el comunicând cu aceasta în urma procesării comenzilor primite de la clienți. Prin intermediul interogărilor (studiate la materia *Baze de date*) corespunzătoare comenzilor introduse în terminal, putem manipula datele în modul dorit.
- Limbaj utilizat: C.

3 Arhitectura aplicației



Utilizatorii vor putea folosi următoarele comenzi:

– **Înregistrare**
Pentru ca un utilizator să acceseze aplicația, e necesar să își creeze un cont (username și parolă). Se verifică dacă username-ul introdus există; dacă nu, clientul se poate înregistra (îl adăugam în tabela *UtilizatoriÎnregistrați*).

– **Autentificare**
Utilizatorii care au cont (verificăm existența lor în tabela *UtilizatoriÎnregistrați* cu ajutorul interogărilor) trebuie să se autentifice pentru a accesa aplicația. Autentificarea se efectuează pe baza introducerii unui username și a unei parole. O autentificare reușită presupune înregistrarea în tabela *UtilizatoriAutentificați* a utilizatorului în cauză. Odata autentificat, utilizatorul are acces la toate functionalitatile aplicatiei.

– **Trimite mesaj**
Un $user_1$ poate trimite mesaj unui $user_2$, dacă $user_2$ există în baza de date.

– **Citește mesaj**
Fiecare utilizator poate verifica dacă a primit mesaje noi. Daca exista mesaje noi, se afiseaza si utilizatorul/utilizatorii de la care le-a primit.

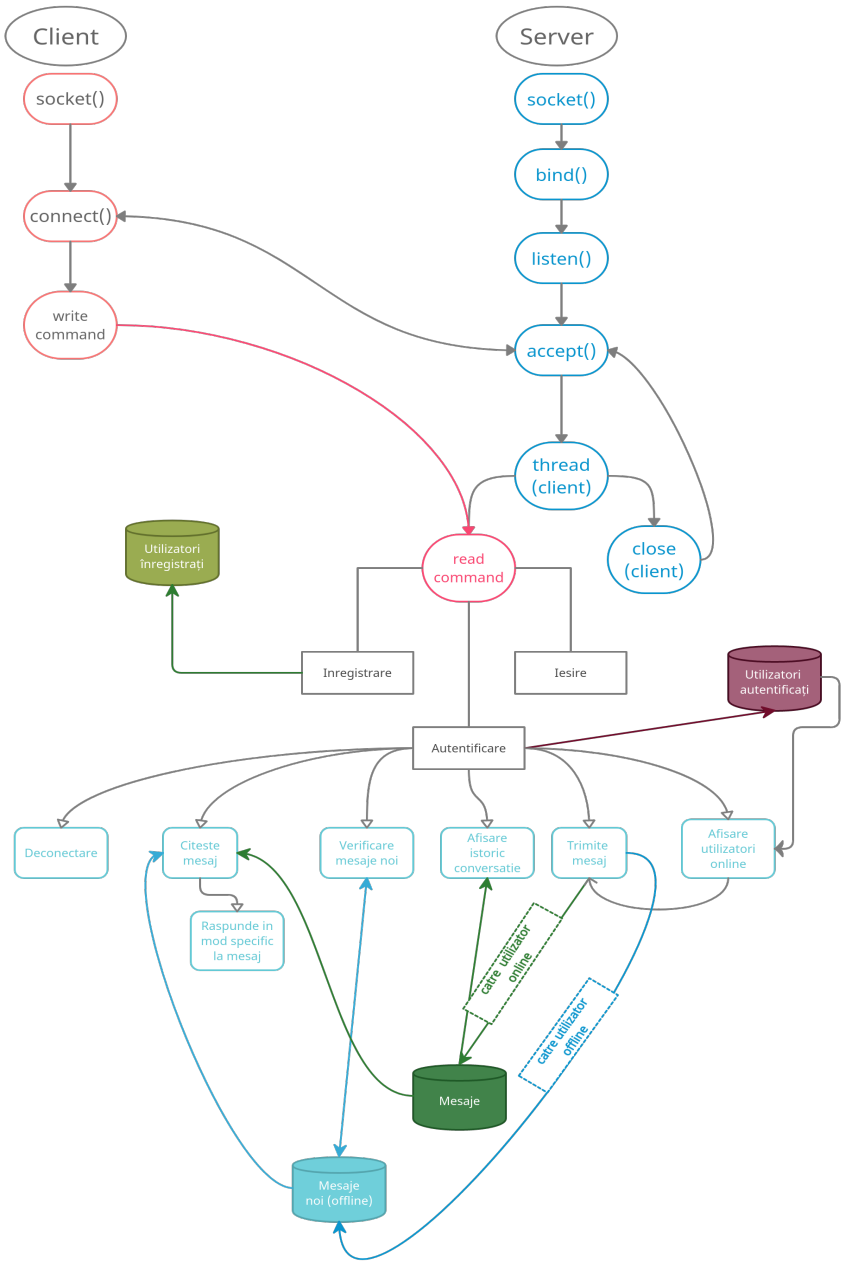
– **Răspunde în mod specific unui mesaj**
Un utilizator va avea posibilitatea de a trimite un răspuns în mod specific la anumite mesaje primite.

– **Afișare istoric conversație**
Fiecare utilizator are posibilitatea de a verifica istoricul conversațiilor cu orice alt utilizator inregistrat.

– **Afișare utilizatori online**
Fiecare utilizator poate verifica ce alti utilizatori sunt conectați, prin accesarea tabelului *UtilizatoriAutentificati*.

– **Deconectare**
Utilizatorul se poate deconecta prin introducerea acestei comenzi. Astfel, el este eliminat din tabela *UtilizatoriAutentificați*.

– **Ieșire**
Utilizatorul poate închide aplicația prin introducerea acestei comenzi.



4 Detalii de implementare

Primitiva socket facilitează comunicarea între sever și client. Un socket este un canal bidirecțional, astfel informația poate fi transmisă în ambele părți ale conex-

iunii. Acesta poate fi asociat unuia sau mai multor procese. Clientul conectează descriptorul de socket la adresa serverului (*connect*), care la randul ei este asignată socket-ului, pe partea de server(*bind*). Socket-ul acceptă conexiuni prin ascultare (*listen*). Apelarea *accept* reprezintă conectarea propriu-zisa cu clientii.

În server se găsește o buclă infinită prin intermediul căreia se acceptă clienți. Odată ce a sosit un client, se creează un thread pentru acesta.

```
typedef struct thData{
  int idThread; //id-ul thread-ului tinut in evidenta de acest
     program
  int cl; //descriptorul intors de accept
}thData;
static void *treat(void *); /* functia executata de fiecare
    thread ce realizeaza comunicarea cu clientii */
```

```
 pthread_t th[100];    //Identificatorii thread-urilor
 int i=0;
  while (1)
    {
      int client;
      thData * td; //parametru functia executata de thread
      int length = sizeof (from);

      printf ("[server]Asteptam la portul %d...\n",PORT);
      fflush (stdout);

      // acceptam un client-stare blocanta
      if ( (client = accept (sd, (struct sockaddr *) &from, &
    length)) < 0)
      {
        perror ("[server]Eroare la accept().\n");
        continue;
      }
    /* s-a realizat conexiunea, se astepta mesajul */
  td=(struct thData*)malloc(sizeof(struct thData));
  td->idThread=i++;
  td->cl=client;//cl = descriptorul intors de accept

  pthread_create(&th[i], NULL, &treat, td);//creare fir de
    executie nou

  }//while
//din curs
```

Programul cuprinde tabelele UtilizatoriInregistrati, UtilizatoriAutentificati, Mesaje si *MesajeNoi* . Crearea acestora este prezentată mai jos.

```
void creareTabele(){
  char *err_msg = 0;
  sqlite3* db;
  char *sql;
```

```
5   int rc;
6   rc = sqlite3_open("OM_BazaDeDate.db", &db); //deschidere
      baza de date
7   if(rc)
8       fprintf(stderr, "Baza de date nu poate fi deschisa\n",
      sqlite3_errmsg(db);
9     else
10        fprintf(stderr, "Baza de date a fost deschisa cu
      succes\n");
11    //creare tabele
12    sql = "CREATE TABLE IF NOT EXISTS UtilizatoriInregistrati
13    (user_ID INTEGER PRIMARY KEY AUTOINCREMENT, nume_user
      varchar(100), parola varchar(100));"
14  int rc = sqlite3_exec(db, sql, 0, 0, &err_msg );
15  if(rc != SQLITE_OK)
16  {
17    fprintf(stderr, "SQL error: %s\n", err_msg );
18    sqlite3_free(err);//The allocated message string must be
      freed
19  }
20  else fprintf(stdout, "Tabela UtilizatoriInregistrati s-a
      creat cu succes!\n");
21
22  sql = "CREATE TABLE IF NOT EXISTS UtilizatoriAutentificati
23 (user_ID INTEGER PRIMARY KEY AUTOINCREMENT, username varchar
      (100));";
24  rc = sqlite3_exec(db, sql, 0, 0, &err_msg );
25    if(rc != SQLITE_OK)
26  {
27    fprintf(stderr, "SQL error: %s\n",err_msg );
28    sqlite3_free(err_msg );
29  }
30  else fprintf(stdout, "Tabela UtilizatoriAutentificati s-a
      creat cu succes!\n");
31
32        sql = "CREATE TABLE IF NOT EXISTS Mesaje(mesaj_ID
      INTEGER PRIMARY KEY AUTOINCREMENT, expeditor varchar(100),
       destinatar varchar(100), continut_mesaj varchar(100));";
33  rc=sqlite3_exec(db, sql, 0 ,0, &err_msg);
34  if(rc != SQLITE_OK)
35  {
36    fprintf(stderr, "SQL error: %s\n",err_msg );
37    sqlite3_free(err_msg );
38  }
39  else fprintf(stdout, "Tabela Mesaje s-a creat cu succes!\n")
      ;
40
41        sql = "CREATE TABLE IF NOT EXISTS MesajeNoi(mesaj_ID
      INTEGER PRIMARY KEY AUTOINCREMENT, expeditor varchar(100),
       destinatar varchar(100), continut_mesaj varchar(100));";
```

```
42   rc = sqlite3_exec(db, sql, 0, 0, &err_msg );
43   if(rc != SQLITE_OK)
44   {
45     fprintf(stderr, "SQL error: %s\n",err_msg );
46     sqlite3_free(err_msg );
47   }
48   else fprintf(stdout, "Tabela MesajeNoi s-a creat cu succes!\
       n");
49
50 }
```

Un utilizator nou trebuie să se înregistreze în aplicație cu un username (nemaifolosit înainte) și o parolă (comanda *Înregistrare*), pentru a avea acces la functionalitatile aplicatiei. Daca username-ul este valid, inregistram utilizatorul in tabela.

```
1  void Inregistrare(char* nume_user, char* parola){
2    sqlite3 *db;
3    sqlite3_stmt * res;//a single sql statement
4    if (sqlite3_open("OM_BazaDeDate.db", &db) == SQLITE_OK)
5    {
6      char* sql = "INSERT INTO UtilizatoriInregistrati (
       nume_user, parola) VALUES (?, ?);";
7      int rc = sqlite3_prepare_v2(db, "SELECT SQLITE_VERSION()",
        -1, &res, NULL);//compiling statement into
8    //a byte code before execution
9      if (rc == SQLITE_OK)
10     { sqlite3_bind_text(res, 2, nume_user, strlen(nume_user),
       SQLITE_TRANSIENT); // store application data into
       parameters
11       sqlite3_bind_text(res, 3, parola, strlen(parola),
       SQLITE_TRANSIENT);
12       sqlite3_step(res);
13       sqlite3_finalize(res);}}}
```

Un utilizator conectat, se poate deconecta de la aplicatie in orice moment prin apelarea comezii *Deconectare*. Astfel, va fi eliminat din tabela *UtilizatoriAutentificati*.

```
1  void removeLoggedUser(char* nume_user){
2    sqlite3 *db;
3    sqlite3_stmt *res;
4    if (sqlite3_open("OM_BazaDeDate.db", &db) == SQLITE_OK)
5    {
6      char* sql = "DELETE FROM UtilizatoriAutentificati WHERE
       name=?;";
7      int rc = sqlite3_prepare(db, sql, -1, &res, NULL);
8      if (rc == SQLITE_OK)
9      { sqlite3_bind_text_v2(res,2,nume_user,-1,NULL);
10       sqlite3_step(res);
11       sqlite3_finalize(res);}}}
```

5 Concluzii

Motivul principal pentru care am ales acest proiect este faptul că aplicațiile de chatting reprezintă un instrument utilizat zilnic în zilele noastre. Astfel, mi s-a părut util să înțeleg cum funcționează ele și să încerc și eu să implementez o astfel de aplicație.

Printre îmbunătățirile care ar putea fi aduse aplicației mele se numără:

- stergerea unui anumit mesaj sau a unei conversatii intregi
- comunicarea in groupchats
- o interfata atractiva
- optiune de a trimite fisiere (poze, etc) intre utilizatori
- blocare conturi

6 Bibliografie

- https://profs.info.uaic.ro/~computernetworks/index.php
- https://profs.info.uaic.ro/~ioana.bogdan/
- https://man7.org/linux/man-pages/
- https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
- https://zetcode.com/db/sqlitec/
- https://www.sqlite.org/cintro.html https://profs.info.uaic.ro/~georgiana.calancea/Laboratorul_12.pdf
- https://www.sqlite.org/c3ref/bind_blob.html