# Offline Messenger

Proiect Retele de calculatoare
Morarasu Catalin Dimitrie 2B4

## 1 Introducere

Offline Messenger este o aplicatie client/server care permite schimbul de mesaje intre utilizatori care sunt conectati. Totodata ofera functionalitatea trimiterii mesajelor catre utilizatorii offline, acestora aparandu-le mesajele atunci cand se vor conecta la server. De asemenea, utilizatorii mai au posibilitatile de a trimite un raspuns (reply) in mod specific la anumite mesaje primite si de as vizualiza istoricul conversatiilor cu fiecare utilizator in parte.

## 2 Tehnologii utilizate

Aplicatia Offline Messenger este realizata in limbajul de programare C si foloseste un server de tip TCP. Transmission Control Protocol (TCP) este un protocol folosit de obicei de aplicatii care au nevoie de confirmare de primire a datelor. Efectueaza o conectare virtuala full duplex între două puncte terminale, fiecare punct fiind definit de catre o adresa IP si de catre un port TCP. Am ales protocolul TCP deoarece este un protocol orientat către conexiune si ofera un serviciu fiabil. Acesta asigura transmiterea ordonata a datelor, detectarea erorilor si retransmiterea automata a pachetelor pierdute, in schimb protocolul UDP nu stabileste conexiunea inaintea transmiterea datelor si nu oferă garantii privind livrarea datelor sau ordinea in care acestea sunt livrate. Totodata in protocolul TCP asigura transmiterea ordonata si fiabila a datelor. Pachetele sunt numerotate si retransmise in caz de pierdere, iar in protocolul UDP nu, pachetele fiind livrate in orice ordine si pot fi pierdute fara a fi retransmise. Pentru stocarea datelor legate de conturi si mesaje folosesc sqlite3.

## 3 Structura aplicatiei

Diagrama de mai jos arata cum functioneaza aplicatia Offline Messenger. Serverul gazduieste clientii la un "IP":"PORT" pentru a face legatura intre clienti si server prin protocolul TCP. Clientul trimite la server o comanda, iar aceasta o prelucreaza cu ajutorul functiilor definite in server. Mai intai clientul are posibilitatea de a apela help, pentru a vizualiza comenzile disponibilie, si login pentru a avea acces la toate functionalitatile aplicatiei.

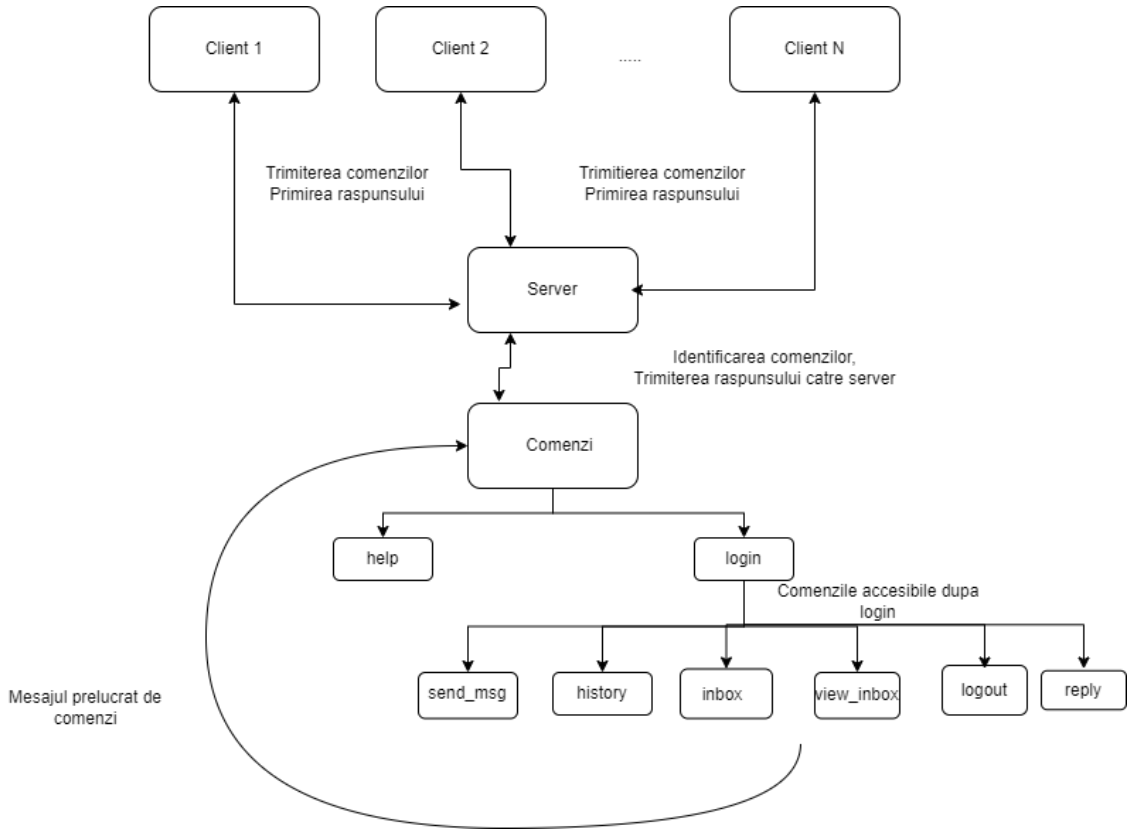


Figure 1: Arhitectura aplicatiei

Conexiunea intre Server si client este realizata prin socket-uri. Dupa acceptarea unei conexiuni si obținerea socketului asociat respectivului client (returnat de accept()), serverul creeaza un thread dedicat clientului/conexiunii respective. In cadrul thread-ului serverul realizeaza comunicarea/transferul de date catre si dinspre clienti, dupa care va finaliza conexiunea in momentul apelului quit.

## 4 Aspecte de implementare

Intre server si client conexiunea se realizeaza prin intermediul socket-urilor, iar pentru fiecare inregistrarea a unui client este creat un nou thread dedicat clientului respectiv. Am ales sa folosesc thread-uri deoarece pot comunica intre ele si ruleaza concurent. Comunicarea intre server si client se realizeaza prin trimiterea unui struct ce stocheaza datele despre clientul conectat la server(id, user, ce mesaj trimite si descriptorul). Memorarea descriptorului este folosita pentru a facilita comunicarea intre utilizatorii online. Aplicatia are aplicabilitatea in viata de zi cu zi deoarece ofera posibilitatea de a conversa in timp real cu utilizatorii conectati sau a lasa mesaje pentru cei offline.

```
int main ()
{
    struct sockaddr_in server;  // structura folosita de server
    struct sockaddr_in from;
    int sd;                     //descriptorul de socket

    /* crearea unui socket */
    if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror ("[server]Eroare la socket().\n");
        return errno;
    }

    /* pregatirea structurilor de date */
    bzero (&server, sizeof (server));
    bzero (&from, sizeof (from));

    /* umplem structura folosita de server */
    /* stabilirea familiei de socket-uri */
    server.sin_family = AF_INET;
    /* acceptam orice adresa */
    server.sin_addr.s_addr = htonl (INADDR_ANY);
    /* utilizam un port utilizator */
    server.sin_port = htons (PORT);

    /* atasam socketul */
    if (bind (sd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
    {
        perror ("[server]Eroare la bind().\n");
        return errno;
    }

    /* punem serverul sa asculte daca vin clienti sa se conecteze */
    if (listen (sd, 1) == -1)
    {
        perror ("[server]Eroare la listen().\n");
        return errno;
    }
```

Figure 2: Conexiunea server/client prin socket

```
/* servim in mod concurent clientii... */
while (1)
{
    int client;
    int length = sizeof (from);
            thData *td;
    printf("[server]Asteptam la portul %d...\n",PORT);
    fflush(stdout);

    /* acceptam un client (stare blocanta pina la realizarea conexiunii) */
    client = accept(sd, (struct sockaddr *) &from, &length);

    /* eroare la acceptarea conexiunii de la un client */
    if(client < 0)
    {
            perror ("[server]Eroare la accept().\n");
            continue;
    }
            td=(struct thData*)malloc(sizeof(struct thData));
            td->idThread=ti++;
            td->cl=client;

    pthread_create(&th[ti], NULL, &treat, td);
}
```

Figure 3: Servirea in mod concurent a clientilor prin thread-uri

Pentru verificarea datelor legate de conturi, stocarea datelor si a mesajelor folosec sqlite3, unde utilizez 3 tabele: users (nume utilizator, parola), newMsg(stocarea mesajelor primite de utilizatorii offline) si history pentru stocarea conversatiei cu un anumit user. Comenzile disponibile sunt login, logout, send msg, reply, inbox sau view inbox, history, register, quit. Fiecare comanda este trimisa la server sub forma de sir de caractere prin struct-ul mentionat anterior. Modelarea datelor nu se face in cadrul bazei de date, aceasta are doar rolul de a stoca sau trimite informatii din cadrul tabelelor. In istoricul conversatiei se pot obersva anumite id-uri in fata mesajelor primite, aceste id-uri sunt folosite pentru functia reply, astfel programul stie la ce mesaj dai reply pentru a-l adauga in tabelul history si pentru a-l afisa destinatarului.

```
void insertMsgOffline(char userSEND[100], char userRECV[100], char msg[100])
{
        sqlite3 *db;
        sqlite3_stmt* stmt;
        int rc;
        rc = sqlite3_open("data.db", &db);
        if(rc != SQLITE_OK)
        {
                fprintf(stderr, "Cannot open database");
                sqlite3_close(db);
        }
        sqlite3_prepare_v2(db, "INSERT INTO newMsg VALUES(?,?,?)", -1, &stmt, 0);

        sqlite3_bind_text(stmt, 1, userSEND, -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 2, userRECV, -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 3, msg, -1, SQLITE_TRANSIENT);

        sqlite3_step(stmt);
        sqlite3_finalize(stmt);
    sqlite3_close(db);
}
```

Figure 4: Exemplu de functie ce utilizeaza baza de date (insereaza mesajul in tabelul newMsg pentru utiliaztorii offline)

```
loggedUsers = mmap(NULL, 1000*sizeof(struct log), PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, -1, 0);
bzero(loggedUsers, 1000*sizeof(struct log));
```

Figure 5: Structura folosita pentru a retine date despre utilizatorii conectati la server (dupa login)

```
else if(strstr(msg.mesaj, "register") != NULL)
{
        if(msg.id == 0)
        {
                char user[100], pass[100];
                commLogin(user, pass, msg.mesaj);

                printf("[server]aceasta este comanda register\n");
                bzero(msgrasp.mesaj,1000);
                /*strcat(msgrasp,"aceasta este comanda register");*/

                if(user[0] != '\0' && pass[0] != '\0')
                {
                        if(verifyUser(user) == 1)
                        {
                                strcat(msgrasp.mesaj, "Acest nume de utilizator este deja folosit!");
                        }
                        else if(verifyUser(user) == 0)
                        {
                                insertNewUser(user, pass);
                                strcat(msgrasp.mesaj, "Utilizator inregistrat cu succes!");
                        }

                }
                else
                        strcat(msgrasp.mesaj, "Eroare, structura: register user parola");
        }
        else if(msg.id != 0)
        {
                bzero(msgrasp.mesaj, 1000);
                strcat(msgrasp.mesaj, "Nu poti inregistra un utilizator daca esti logat!");
        }

}
```

Figure 6: Exemplu de comanda din cadrul serverului(comanda de inregistrare la aplicatie)

```
else if(loggedUsers[idRECV].id != 0)
{
        struct log msgTrimis;
        msgTrimis.id = loggedUsers[idRECV].id;
        printf("status user primit:%d",loggedUsers[idRECV].id);

        bzero(msgTrimis.mesaj,1000);
        strcpy(msgTrimis.mesaj, userSEND);
        strcat(msgTrimis.mesaj, ": ");
        strcat(msgTrimis.mesaj, userMSG);
        strcpy(msgTrimis.user,userRECV);
        ///msgTrimis.client = loggedUsers[idRECV].client;
        insertMsgHistory(userSEND,userRECV, userMSG);

        write (loggedUsers[idRECV].client, &msgTrimis, sizeof(struct log));
        bzero(msgrasp.mesaj,1000);
        strcpy(msgrasp.mesaj, "Mesaj trimis cu succes!");
        /*printf("[server]aceasta este comanda send message\n");
        strcat(msgrasp.mesaj,"aceasta este comanda send message");*/
}
```

Figure 7: trimiterea mesajului pentru un destinatar online si pentru clientul ce a lansat comanda mesajul "mesaj trimis cu succes!"

## 5 Concluzii

Imbunatatirile aplicatiei pot fi legate de modul de stocare a mesajelor precum si a utilizatorilor folosind baze de date, pentru login, istoric si in-

box. Totodata pot fi implementate noi functionalitati interesante precum crearea unei liste de prieteni si vizualizarea utilizatorilor online.

## 6 Referințe Bibliografice

Site-ul cursului : https://profs.info.uaic.ro/ computernetworks/index.php
Site-ul laboratorului : https://sites.google.com/view/fii-lab-retele-de-calculatoare/home
Wikipedia TCP : https://ro.wikipedia.org/wiki/Transmission_Control _Protocol
Documentatie Sqlite3 : https://www.sqlite.org/index.html