Offline Messenger (B)

Bojescu Bianca-Daniela, Grupa X

Facultatea de informatică, Iași

Abstract. Raportul tehnic al aplicației de tip server-client va descrie modul în care am gandit eu realizarea proiectului Offline Messenger.

Keywords: Mesaje · Offline · Sever · Client · TCP · Socket.

1 Introducere:

Proiectul presupune dezvoltatrea unei aplicații de tip client/server ce permite schimbul de mesaje între doi sau mai mulți utilizatori logați.

De asemnea, software-ul are implementat funcționalitatea de a transmite mesaje și utilizatorilor offline, putând să le citească ulterior la conectare. Aplicația va oferi si posibilitatea de înregistrare în baza de date, schimbare a parolei, cât si vizualizarea istoricului conversațiilor cu fiecare utilizator în parte.

2 Tehnologii utilizat:

Acest program este implementat cu ajutorul limbajului C.

Un protocol reprezintă un set de reguli ce asigură modul în care sunt transmise datele. Protocolul utilizat este TCP (Transmission Control Protocol). TCP/IP e o stiva de protocoale de comunicare ce permite interconectarea la distanță a mai multor dispozitive. Am ales utilizarea acestuia, deoarece asigură siguranță, cât și transmiterea în ordine a datelor, fără pierderea acestora, aspect fundamental în crearea aplicațiilor de transmitere a mesajelor.

Gestionarea proceselor se face prin thread-uri, acestea fiind dependente unele de altele. Thread-urile sunt mult mai rapide decât procesele deoarece:

- Se crează mult mai repede
- De asemenea, informația se schimbă între ele mai rapid
- Thread-urile pot fi terminate ușor

Baza de date utilizată este MySQL (biblioteca mysql/mysql.h). In baza de date avem doua tabele: mesaje și users. Fiecare user si mesaj au un id unic, ca fiind cheie primara. Prin intermediul interogarilor putem afla pentru utilizatori: username-ul, parola și starea autentificării acestora. Prin metode de update și

insert putem actualiza informații despre aceștia. Pentru mesaje avem o tabela ce conține id-ul unui expeditor și id-ul unui destinatar, ambele fiind chei străine din tabela de users, cât și mesajul text trimis și o coloana de seen, ce are o valuare default 0 pentru mesaje trimise, dar necitite de destinatar. Comenzile utilizate pentru crearea bazei de date offlineMessenger:

```
CREATE TABLE authentificated (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL,
    authenticated TINYINT DEFAULT 0
);

CREATE TABLE mesaje (
    id INT PRIMARY KEY AUTO_INCREMENT,
    expeditorId INT,
    destinatarId INT,
    mesaj TEXT,
    seen TINYINT DEFAULT 0,
    FOREIGN KEY (expeditorId) REFERENCES authentificated(id),
    FOREIGN KEY (destinatarId) REFERENCES authentificated(id)
);
```

3 Arhitectura aplicației:

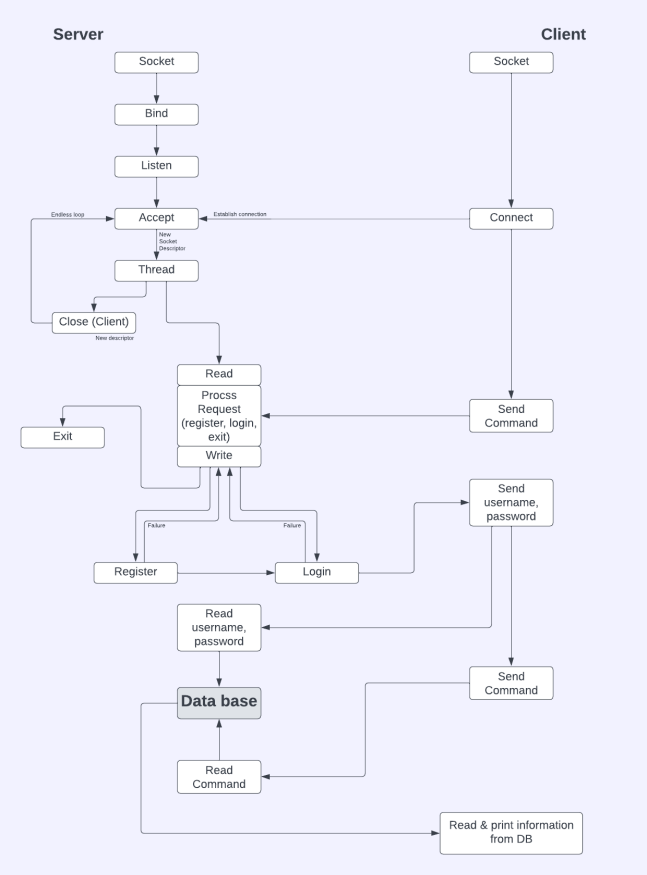
Aplicația "Offline Messenger" se bazează pe principiul unui server concurent astfel încât se pot deservi mai mulți clienți. Se crează câte un proces fiu pentru fiecare client, iar comunicarea dintre server si client se realizează prin intermediul socket-urilor.

Comenziile ce vor fi disponibile pentru trimitere de la client la server înainte ca utilizatorul să fie logat:

1. Register:
   Utilizatorul îsi va crea un username care nu există deja și îi va atribui o parolă. In baza de date starea authenticated va fi stabilită default 0, urmând ulterior să se modifice la logarea acestuia.
2. Login:
   Dacă acesta are un cont existent se va conecta folosind username-ul și parola, această acțiune actualizând în baza de date flag-ul authenticated cu 1. În cazul în care username-ul sau parola sunt greșite, logarea nu se va realiza, iar persoana va fii înștiințată.
3. Quit:
   Va închide conexiunea cu clientul.

Comenziile ce vor fi disponibile pentru trimitere de la client la server dupa ce utilizatorul se va loga:

1. Change password:
   Utilizatorul își va putea schimba parola.
2. Send message:
   Un utilizator poate să trimită un mesaj către un alt utilizator, precizând cui dorește să trimită mesajul. În baza de date se va insera acesta cu valoarea seen setată by deafult 0.
3. See new messages:
   Cu utilizarea acestei comenzi, utilizatorul poate vedea ce mesaje noi a primit.
4. See a conversation:
   Un utilizator are posibilitatea să selecteze un alt user și să vadă istoricul conversațiilor acestora.
5. Online users:
   Utilizatorii vor putea să vadă alți utilizatori logați.
6. Logout:
   Un utilizator se va deloga din aplicație, aceasta memorând istoricul acțiunilor sale (mesaje trimise sau mesaje citite).
7. Quit:
   Va deloga user-ul apoi va închide conexiunea cu clientul.



4 Detalii de implementare:

Comunicarea dintre server si client se face prin socket-uri. Comunicarea realizându-se în ambele direcții (full duplex) fără a fi nevoie de mecanisme proprii de sincronizare. Un alt avantaj al acestora este și faptul că nu trebuie să închidem la final canalul de transmitere.

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

În server setăm pentru socket opțiunea SO_REUSEADDR ce specifică că regulile utilizate în validarea adreselor furnizate către bind() ar trebui să permită reutilizarea adreselor locale, dacă acest lucru este acceptat de protocol.

```
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));
```

Pregătim structurile de date cu bzero(), o funcție ce șterge datele din n octeți ai memoriei, scriind zerouri în acea zonă de memorie, iar apoi o umplem.

```
/* pregatim structurile de date */
bzero(&server, sizeof(server));

/* umplem structura folosita de server */
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(PORT);
```

Funcția bind() va atribui o adresă socket locală ce se adresează unui socket identificat printr-un descriptor.

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

Cu ajutorul funcției listen() vom pune server-ul să asculte dacă vin clienți să se conecteze.

```
if (listen(sd, 5) == -1)
{
    perror("[server] Eroare la listen().\n");
    return errno;
}
```

În continuare, în program vom servi în mod concurent clienții folosind thread-uri.

```
/* a venit un client, acceptam conexiunea */
client = accept(sd, (struct sockaddr *)&from, &len);

/* eroare la acceptarea conexiunii de la un client */
if (client < 0)
{
    perror("[server] Eroare la accept().\n");
    continue;
}
```

Conectarea de la client în server se face cu ajutorul funcției: int connect(int socket, const struct sockaddr *address,socklen_t address_len); (interleagă socket-urile din client și server)

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

În proiect se regăsesc funcții pentru fiecare comandă, unde se face legătura cu baza de date.

5 Concluzii:

O aplicație de acest tip facilitează o comunicare interumană prin mesaje. Un element important în crearea aplicației reprezintă aspectul său grafic ce oferă o înfățișare plăcută pentru utilizatori. Un alt aspect ce va atrage consumatorul sunt funcționalitățiile implementate în program ce vor face experiența acestuia cât mai ușoară, rapidă și eficientă.

References

1. https://profs.info.uaic.ro/ computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf
2. https://www.andreis.ro/teaching/computer-networks
3. https://www.geeksforgeeks.org/multithreading-c-2/
4. https://stackoverflow.com/questions/3229860/what-is-the-meaning-of-so-reuseaddr-setsockopt-option-linux
5. https://man7.org/linux/man-pages/man3/bzero.3.html
6. https://pubs.opengroup.org/onlinepubs/009695399/functions/connect.html
7. https://stackoverflow.com/questions/3453168/c-program-mysql-connection