Documentatie proiect "Offline Messenger"

Vreme Iulia, grupa B4

15 Decembrie 2023

Introducere

Proiectul "Offline Messenger" are scopul dezvoltarii unei aplicații server/client pentru a simula o aplicație de mesagerie cu următoarele functionalitati: schimbul de mesaje intre utilizatorii care sunt conectați, trimiterea mesajelor utilizatorilor care nu sunt conectați în aplicație, afisarea mesajelor primite (cât timp un utilizator nu a fost conectat) în momentul reconectarii în aplicație, optinuea de a răspunde la un mesaj specific, afisarea istoricului conversatiilor cu un anumit utilizator, dar și multe alte opțiuni pentru a eficientiza interacțiunea clientilor cu aplicația noastră.

Tehnologii utilizate

Protocolul de comunicare potrivit pentru aceasta aplicație este protocolul TCP (Transmission Control Protocol).

TCP este un protocol utilizat pentru organizarea datelor într-un mod care sa asigure transmisia sigură intre client și server, garantand integritatea datelor trimise prin rețea, indiferent de cantitate. Acest protocol implementeaza un mecanism de control al erorilor pentru transferul fiabil de date, fapt care ne asigura ca niciun mesaj nu se va pierde, niciun mesaj nu va ajunge segmentat la destinatar sau chiar duplicat din cauza congestiilor din rețea, iar ordinea primirii mesajelor de către destinatar este aceiași cu ordinea trimiterii mesajelor de către sursa. În plus, TCP efectueaza o conectare virtuala full-duplex intre doi clienți datorită careia datele pot fi transmise de la clientul1 la clientul2 sau invers în același timp.

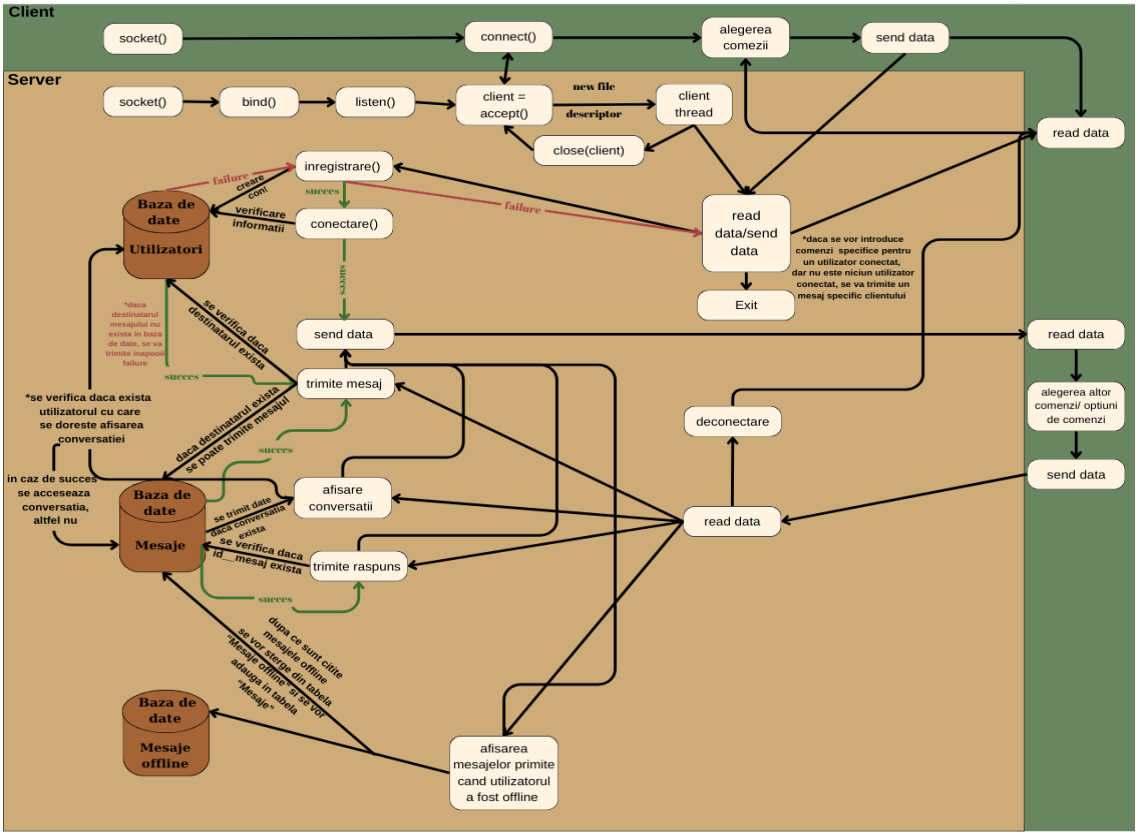
Având în vedere faptul ca aplicatia trebuie sa servească clientii în mod

concurent, tehnica optima din punct de vedere al memoriei este utilizarea thread-urilor (firelor de execuție), astfel vom crea un thread pentru fiecare client.

Pentru stocarea datelor (nume de utilizatori, parole, date despre utilizatori, conversatii, mesaje) se folosește SQLite deoarece scrierea/citirea/cautarea sunt mai eficiente într-o baza de date decât într-un fisier text (nu se acceseaza date direct de pe disk), iar modul de actualizare a datelor se simplifica prin utilizarea interogarilor SQL.

Arhitectura aplicatiei

Diagrama de mai jos ilustreaza arhitectura aplicatiei client/server:



In diagrama aplicatiei se observa ca realizarea conexiunilor dintre clienti se face cu ajutorul unui server concurent astfel :

- Serverul creaza un socket, il ataseaza la un port prestabilit pentru a putea oferii sevicii la acel port si intra intr-o stare de asteptare pasiva a conexiunilor de la potentiali clienti

- Clientul creaza un socket si ii aloca o zona de memorie, apoi incepe conectarea cu serverul

- In bucla de tratare, serverul sesizeaza cererea clientului de a se conecta cu el

- Daca serverul accepta conexiunea cu clientul si obtine socketul asociat respectivului client, creaza un fir de executie dedicat clientului respectiv/conexiunii respective

- Clientul isi alege comenzile dorite, iar in functie de acestea serverul executa instructiunile fiecarei comenzi

Lista de comenzi posibile:

1.inregistrare : utilizatorul are optiuna de a-si crea un cont, dacă nu are deja unul. Datele necesare sunt : nume de utilizator (se va verifica sa nu fie identic cu unul deja în baza de date), parola, numar de telefon (necesar pentru schimbarea parolei)

2.conectare : când utilizatorul va dori să se conecteze, serverul ii va cere numele de utilizator, dacă nu este corect conexiunea va esua, altfel i se va cere parola. Dacă aceasta este cea corecta (cea cu care s-a creat contul, sau cea schimbata in trecut) conexiunea va reusi, altfel va avea optinea de a-si schimba parola sau conexiunea va esua

3.deconectare : posibila doar dacă utilizatorul este conectat, altfel deconectarea nu ar avea sens

4.exit : iesirea din aplicatie

5.afisarea mesajelor offline : posibila doar dacă un utilizator este conectat la aplicație (clientul va fi avertizat ca are noi mesaje necitite (daca exista) chiar dupa ce conexiunea la aplicatie va reusi si va avea optinuea de a le vizualiza cand doreste)

6.trimitere mesaj : posibila doar dacă utilizatorul este conectat la aplicație. Datele necesare trimiterii unui mesaj sunt : destinatarul mesajului(se va verifica daca exista in aplicatie) , mesajul de trimis.

7.raspunde mesaj : se va putea răspunde în mod specific la un mesaj primit în timp cât utilizatorul este online, cât și mesajelor primite când uti-

lizatorul a fost offline

8.afisare conversatii : se vor afisa conversatiile unui utilizator cu un utilizator specificat (daca utilizatorul are mesaje necitite, acestea nu vor fi afisate)

9.utilizatori online : se vor afisa utilizatori conectati la aplicatie in acel moment

10.vazut : utilizatorul va avea obtinuea de a vedea daca utilizatorul caruia i-a dat mesaj a vazut mesajul

11. data/ora trimitere : utilizatorul va avea optinea de a verifica la ce ora a trimis/primit un mesaj (in functie de id mesaj)

12. schimbare parola : un utilizator conectat își poate schimba parola dacă își cunoaște parola actuala, un utilizator neconectat își poate schimba parola (în caz ca a uitat-o) dacă își cunoaște numele de utilizator și numărul de telefon

Desi in diagrama de mai sus sunt ilustrate trei constructii de tipul baze de date, aplicatia contine doar o baza de date cu trei tabele. Tabelele ce se regaesc in baza de date a server-ului, dar si un exemplu de valori pentru aceste tabele, sunt urmatoarele :

1. Tabela utilizatori

id utilizator	nume utilizator	parola	numar de telefon	status
1	nume1	parola1	+40792301294	online
2	nume2	parola2	+40732751635	offline
3	nume3	parola3	+40765357365	online
4	nume4	parola4	+40757509844	online

2. Tabela mesaje offline

id mesaj	nume utilizator sursa	nume utilizator destinatie	mesaj
1	nume1	nume4	Buna! Vrei sa iesim azi la cafea ? Vin nume2 si nume 3.

3. Tabela mesaje

id mesaj	nume utilizator sursa	nume utilizator destinatie	mesaj	seen	data/ora trimitere
1	nume1	nume2	Buna! Vrei sa iesim azi la cafea ?	1	10:08/12.12.2023
2	nume2	nume1	Buna! Sigur ca da.	1	10:10/12.12.2023
3	nume1	nume3	Buna! Vrei sa iesim azi la cafea ? Vine si nume2.	1	10:11/12.12.2023
4	nume3	nume1	Buna! Mi-ar placea foarte mult!	1	10:17/12.12.2023
5	nume1	nume4	Buna! Vrei sa iesim azi la cafea ? Vin nume2 si nume 3.	0	10:18/12.12.2023

Detalii de implementare

In urmatoarele sectiuni de cod este exemplificata implementarea unei aplicatii TPC client/server - modul concurent, in care de fiecare data cand un client este acceptat, se creaza un nou fir de executie. Pentru fiecare comanda primita, serverul va trimite confirmari clientului.

CLIENT

```
    int server_descriptor;
    struct sockaddr_in server_addr;
    char comanda[100];
    char raspuns[100];
    int cod_write, cod_read;

    if(argc != 3)
    {
        printf ("Sintaza: %s <adresa_server> <port>.\n", argv[0]);
        return -1;
    }

    port = atoi(argv[2]);

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

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(argv[1]);
    server_addr.sin_port = htons(port);

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

```c
while(1)
{
    printf("Introduceti comanda dorita: ");
    fflush(stdout);

    fgets(comanda, 100, stdin);
    comanda[strlen(comanda) -1] = '\0';

    cod_write = write(server_descriptor, &comanda, strlen(comanda));
    if(cod_write == -1)
    {
        perror("[client]Eroare la write() catre server.\n");
        exit(EXIT_FAILURE);
    }

    cod_read = read(server_descriptor, &raspuns, sizeof(raspuns));
    if(cod_read == -1)
    {
        perror("[client]Eroare la read() de la server.n");
        exit(EXIT_FAILURE);
    }

    raspuns[strlen(raspuns)] = '\0';

    printf("[client]%s", raspuns);

}
```

SERVER

Offline_messager > server.c > main()

```c
#define PORT 2222
extern int errno;

typedef struct thData
{
    int idThread; // id_il thread-ului tinut in evidenta de acest program
    int cl; // descriptorul intors la accept
}thData;


static void *treat(void *); /*functie executata de fiecare thread ce realizeaza comunicarea cu clientii*/
int verificare_comanda(void *);

int main()
{
    printf("[server]Serverul este pornit!\n Asteptam comenzi din partea clientilor.\n");

    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    int server_descriptor;
    pthread_t th[100];
    int i=0;
    int on=1;

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

    setsockopt(server_descriptor, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
    //reutilizarea unei adrese locale asociate unui socket care a fost recent inchis

    bzero(&server_addr, sizeof(server_addr));
    bzero(&client_addr, sizeof(client_addr));

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    server_addr.sin_port = htons(PORT);

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

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

    while(1)
    {
        int client_descriptor;
        int length = sizeof(client_addr);
        thData *td;

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

        if((client_descriptor = accept(server_descriptor, (struct sockaddr *) &client_addr, &length)) < 0)
        {
            perror("[server] Eroare la accept().\n");
            continue;
        }

        td = (struct thData*)malloc(sizeof(struct thData));
        td->idThread = i++;
        td->cl = client_descriptor;

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

Functia *treat(void*) : aceasta functie este executata de fiecare thread pentru a gestiona comunicarea cu clientul pentru care a fost creat. Cand clientul isi incheie activitatea in aplicatie, se termina si executia thread-ului.

```
87   static void *treat(void *arg)
88   {
89       struct thData threat;
90       threat = *((struct thData*)arg);
91
92       printf("[thread %d]Asteptam mesajul....\n", threat.idThread);
93       fflush(stdout);
94       pthread_detach(pthread_self());
95       verificare_comanda((struct thData*)arg);
96
97       //am terminat cu acest client
98
99       close((intptr_t)arg);
100      return(NULL);
101
102  };
```

Functia verificare comanda(void*) : aceasta comanda este folosita pentru a trata comenzile clientului si de a trimite confirmari(mesaje privind succesul si esecul anumitor operatiuni), practic gestioneaza schimbul de mesaje dintre thread si client.

```
int verificare_comanda(void *arg)
{
    int i=0;
    char comanda[1000];
    char raspuns[1000] = "";
    struct thData thread;
    thread = *((struct thData*)arg);
    int size;
    int cod_read, cod_write;
    int conectat = 0;

    while(1)
    {
        cod_read = read(thread.cl, &comanda, sizeof(comanda));
        if(cod_read == -1)
        {
            printf("[thread %d]", thread.idThread);
            printf("Eroare la read() de la client.\n");
        }

        comanda[strlen(comanda)] = '\0';

        printf("[thread %d]Am primit comanda: %s\n", thread.idThread, comanda);

        /* Urmeaza verificarea tipului fiecarei comenzi
           si executarea comportaamentului dorit pentru fiecare comanda
        */

       if(strstr(comanda, "conectare") != 0)
       {
            /*aici vom trimite un raspuns clientului pentru a furniza numele de utilizator -> daca aceste este in baza de date
            se va cere parola, altfel Conectare esuata . Daca parola este corecta conectare reusita, altfel */

            cod_write = write(thread.cl, &raspuns, strlen(raspuns));
            if(cod_write == -1)
            {
                perror("Eroare la write() catre client.\n");
                exit(EXIT_FAILURE);
            }
       }
    }
}
```

BAZA DE DATE

Baza de date are un rol foarte important in dezvoltarea acestei aplicatii. Aceasta va contine 3 tabele, cum am mentionat si mai sus. Fiecare apritie a unui mesaj, va fi indexata cu id-ul sau pentru a le putea deosebi (spre exemplu: un utilizator se conecteaza si alege sa isi afiseze mesajele primite cat timp acesta nu a fost online. Presupunem ca un acelasi utilizator i-a trimis un acelasi mesaj de doua ori. Cum ar putea alege utilizatorul nostru carui mesaj sa dea replay? Solutia este indexarea mesajelor cu id unic).

Un alt avantaj este organizarea informatiilor pe linii si coloane, fiind simplu sa filtram anumite informatii. Spre exempl: utilizatorul1 doreste sa ii fie afisate toate mesajele cu utilizatorul2. Astfel cu o singura interogare afisam toate mesajele (din tabela mesaje) unde destinatarul este utilizatorul1 si sursa este utilizatorul2 sau destinatarul este utilizatorul2 si sursa este utilizatorul1.

Pentru crearea bazei de date se vor folosi functiile din biblioteca "$< sglite3.h >$"

Concluzii

Proiectl "Offline Mesessenger" se bazeaza pe modelul client/server folosindu-se comunicarea TCP concurent. Imbunatatiri care ar putea fi aduse la aceasta aplicatie sunt:

-posibilitatea crearii unui grup (mai multi clienti sa trimita mesaje in acelasi mediu)

-posibilitatea de a conometra timpul petrecut in aplicatie

-posibilitatea editarii unui mesaj trimis

-posibilitatea stergerii unui mesaj trimis (unui utilizator offline)

-posibilitatea te a bloca un utilizator (cand un utilizator este blocat de un alt utilizator, atunci acestia nu mai pot sa-si trimita mesaje)

Bibliografie

Bibliografia accesata in scopul realizarii acestei documentatii:
1) https://sites.google.com/view/fii-lab-retele-de-calculatoare/laboratoare?pli=1
2 https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
3) https://www.geeksforgeeks.org/multithreading-in-c/

4) https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/
5) https://www.w3schools.blog/advantages-sqlite
6) https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/