Documentatie Proiect Offline Messenger

Radu Iulia-Teodora

Facultatea de Informatica Iasi

Abstract. Acest raport tehnic prezintă tehnologiile utilizate, arhitectura aplicației și detaliile de implementare ale proiectului Offline Messenger(B).

Keywords: Offline Messenger · TCP · fork

1 Introducere

Acest proiect isi propune sa dezvolte o aplicatie de comunicare intre client si server care sa permita schimbul de informatii intre clienti in privinta comunicarii online si offline.

Astfel, in momentul in care un client se va conecta la server, el va putea atat sa trimita mesaje utilizatorilor online si offline, sa trimita reply la un mesaj anume din conversatia sa cu alt utilizator, sa isi citeasca istoricul conversatiei cu fiecare utilizator in parte si sa citeasca mesajele ce au fost trimise cat timp era offline.

2 Tehnologii Aplicate

Protocolul de control al transferului(TCP - Transmission Control Protocol) a fost tehnologia utilizata in realizarea acestui proiect. Offline Messenger implică trimiterea și recepționarea mesajelor între utilizatori, iar TCP oferă anumite avantaje care pot fi considerate utile în această situație. Unul dintre acestea faptul ca mesajele vor fi livrate in ordinea in care au fost transmise, astfel incat conversatiile utilizatorilor vor avea sens si nu se vor produce confuzii. De asemenea, folosind TCP, se asigură livrarea fiabilă a datelor catre destinatie, lucru important pentru utilizatorii ce au mesaje importante de dat.

Prin folosirea primitivei fork() pentru concurență într-un server TCP, s-a realizat gestionarea eficienta pentru a trata simultan mai multe conexiuni. Procesele separate pentru fiecare conexiune sunt utile pentru a trata in acelasi timp mai multe cereri de la diferiți clienți fără a afecta performanța sau funcționalitatea serverului pentru celelalte conexiuni, iar daca un proces care gestionează o anumită conexiune întâmpină o eroare sau pică, celelalte conexiuni nu sunt afectate, fapt ce contribuie la stabilitatea generală a serverului.

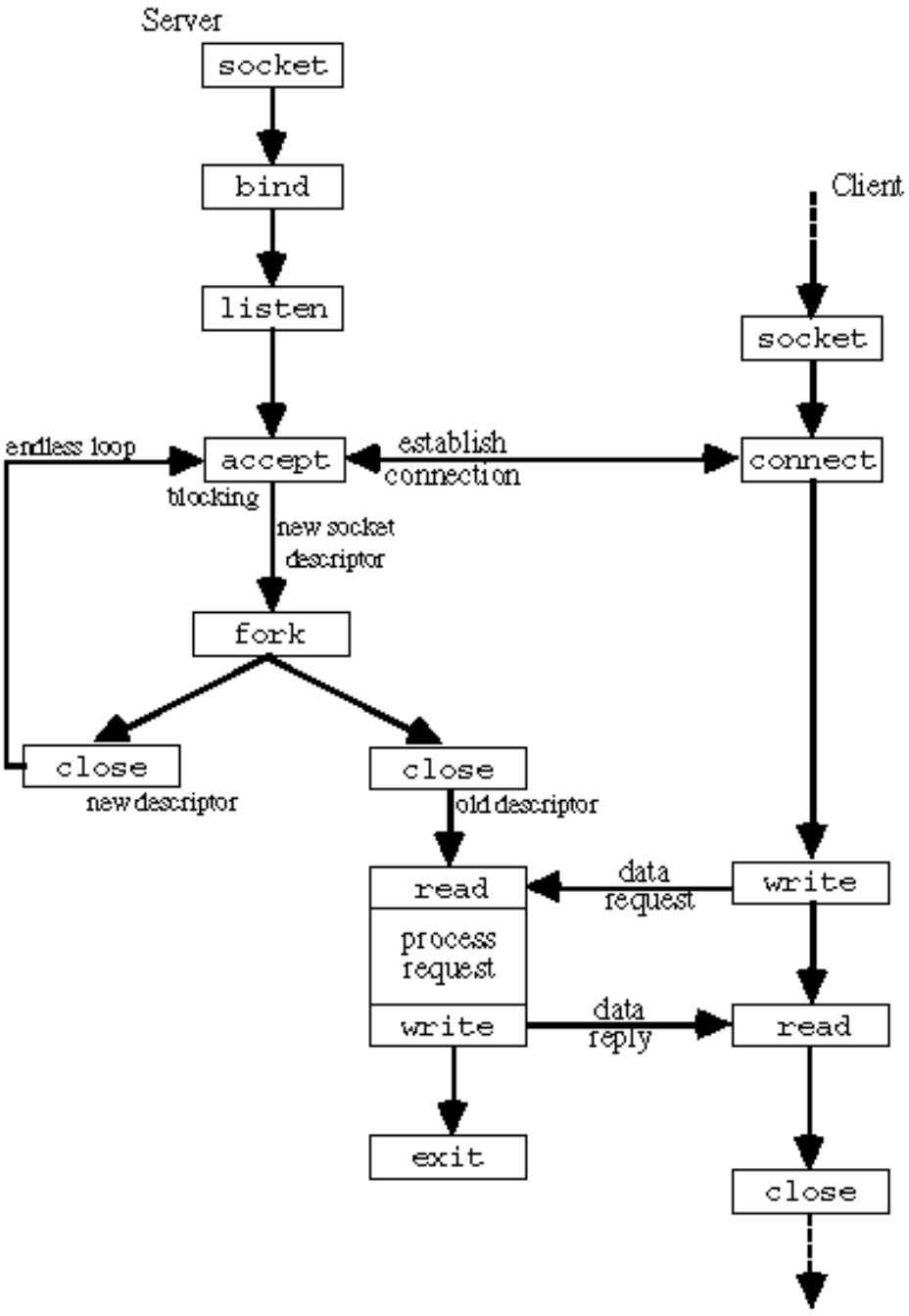


Fig. 1. Schema TCP Concurent

3 Structura Aplicației

Conceptele folosite in modelare sunt:

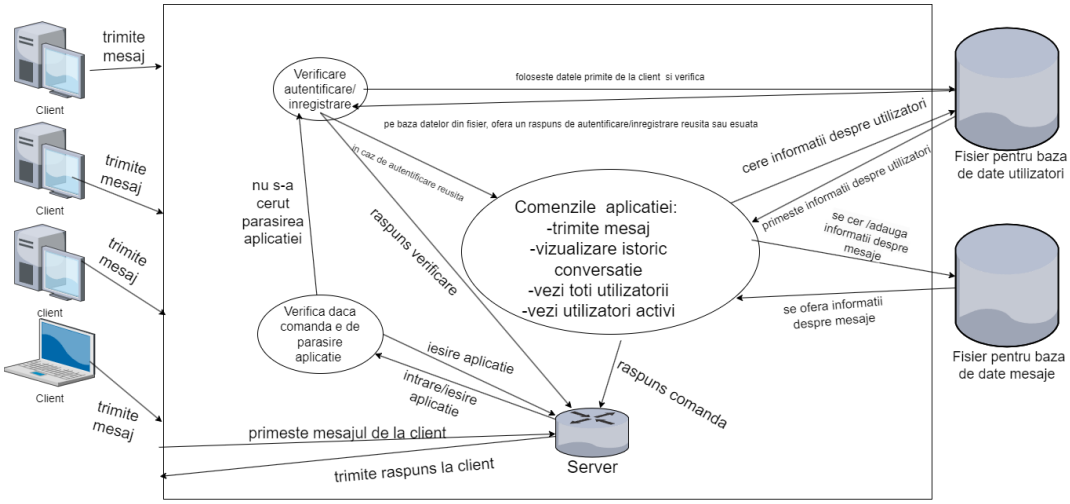
-utilizatorii: reprezintă entitățile individuale care folosesc aplicația,care isi pot crea cont folosind optiunea de inregistrare sau pot intra in cont folosind optiunea de autentificare, urmand sa aiba dupa acces la toate comenzile. De asemenea, au optiunea de iesire din aplicatie.

-mesajele: sunt elementele centrale ale unei aplicații precum Offline Messenger. Acestea sunt texte,iar modelarea lor implica detalii precum expeditor, destinatar, conținut, statusul daca a fost citit sau nu de catre destinar si daca reprezinta in sine un reply la un mesaj existent.

-conversațiile: reprezintă schimburile de mesaje între utilizatori. O conversație este alcatuita dintre mesajele dintre doi utilizatori.

-starea de conectivitate: fiind o aplicație offline, este util să se modeleze starea de conectivitate a utilizatorilor. Acest lucru include gestionarea mesajelor primite în timpul in care un utilizator nu este conectat și sincronizarea acestora când utilizatorul revine in aplicatie.

-sincronizare offline: o alta caracteristică cheie într-o aplicație offline este capacitatea de a salva și sincroniza datele în mod corespunzător când utilizatorul revine online. Acest fapt este crucial pentru a asigura coerența datelor, logica in conversatiile utilizatorilor si transmiterea mesajelor.



Arhitectura Offline Messenger

4 Aspecte de Implementare

În cadrul acestei secțiuni, se vor detalia tehnic procesele de construcție care stau la baza funcționării acestei aplicații, concentrându-se pe aspectele practice precum gestionarea datelor la nivel local și sincronizarea eficientă între diferitele componente.

```
/* servim in mod concurent clientii... */
while (1)
{
    int client;
    int length = sizeof (from);
    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;
    }
    int pid;
    if ((pid = fork()) == -1) {
            close(client);
            continue;
    } else if (pid > 0) {
            // parinte
            close(client);
            while(waitpid(-1,NULL,WNOHANG));
            continue;
    } else if (pid == 0) {
            // copil
            close(sd);
```

Fig. 2. Exemplul numarul 1 de cod sursa

Aceasta sectiune de cod surprinde concurenta serverului, atribut ce ii ofera posibilitatea sa accepte mai multi clienti in acelasi timp si sa trateze cererile fiecaruia in parte cu ajutorul procesului copil creat prin primitiva fork().

```
printf ("Bun venit la Offline Messenger!\n");
printf ("Inregistrare / Autentificare: ");
int iesire=1;
while(iesire)
{
fflush (stdout);
bytes=read (0, msg, sizeof(msg));
msg[bytes-1]='\0';
/* trimiterea mesajului la server */
if (write (sd, &msg, sizeof(msg)) <= 0)
  {
    perror ("[client]Eroare la write() spre server.\n");
    return errno;
  }

/* citirea raspunsului dat de server
   (apel blocant pina cind serverul raspunde) */
if (read (sd, &msg, sizeof(msg)) < 0)
  {
    perror ("[client]Eroare la read() de la server.\n");
    return errno;
  }
/* afisam mesajul primit */
if (strcmp(msg,"out")==0)
{
/* inchidem conexiunea, am terminat */
iesire=0;
if (write (sd, &msg, sizeof(msg)) <= 0)
  {
    perror ("[client]Eroare la write() spre server.\n");
    return errno;
  }
close (sd);
}
else
printf ("\n%s", msg);
```

Fig. 3. Exemplul numarul 2 de cod sursa

In ceea ce priveste codul din partea clientului, acesta nu face decat sa citeasca de la server si sa scrie mesajele inapoi. Singurul moment in care se face o verificare, este in momentul deconectarii, astfel incat aplicatia sa se inchida fara probleme.

```
while(activ)
{
        /* citirea mesajului */
if ((num=read (client, msg, 1024)) <= 0)
{
        perror ("[server]Eroare la read() de la client.\n");
        close (client); /* inchidem conexiunea cu clientul */
        continue;               /* continuam sa ascultam */
}
else{
 while(num==0)
 {sleep(1);}
 msg[num]='\0';
 }
printf ("[server]Mesajul a fost receptionat...%s\n", msg);
if (strcmp(msg,"exit")==0 || strcmp(msg,"iesire")==0)
   {
   strcat(msgrasp,"out");
   activ=0;
   }
   else
switch (logat)
{
case 0:
```

Fig. 4. Exemplul numarul 2 de cod sursa

Cu ajutorul unor varibile de control, se poate observa cum, cata vreme un utilizator este activ, anume nu a introdus comanda de parasire a aplicatiei "iesire"/"exit", se vor citi comenzile trimise de acesta si vor fi tratate in functie de starea acestuia: autentificat sau nu.

```
printf("[server]Trimitem mesajul inapoi...%s\n",msgrasp);
/* returnam mesajul clientului */
if (write (client, msgrasp, 1024) <= 0)
{
        perror ("[server]Eroare la write() catre client.\n");
        continue;               /* continuam sa ascultam */
}
else
        printf ("[server]Mesajul a fost trasmis cu succes.\n");
}
/* am terminat cu acest client, inchidem conexiunea */
close (client);
exit(0);
```

Fig. 5. Exemplul numarul 3 de cod sursa

Astfel, dupa ce se prelucreaza mesajul raspuns pentru client, acesta este trimis iar comunicarea dintre client si server are loc pana cand clientul trimite comanda de parasire a aplicatiei. Drept urmare, serverul va termina cu acel client si cu acel proces copil.

```
bzero(msgrasp,1024);
if (strcmp(msg,"help")==0 || strcmp(msg,"ajutor")==0||strcmp(msg,"x")==0)
{
de_deschis=0;
strcat(msgrasp,"\nMeniu comenzi:\n-ajutor/help\n-vezi utilizatori activi/see online users\n-vezi toti
utilizatorii/see all users\n-trimite mesaj/send message\n-intra in conversatie/open conversation \n-iesire/exit");
strcat(msgrasp, "\nComanda ta: ");
}
else
if (strstr(msg,"utilizatori activi")!=NULL || strstr(msg,"online users")!=NULL)
{
strcat(msgrasp, "Lista utilizatorilor activi:\n");
strcat(msgrasp, "\nComanda ta: ");
}
else
if (strstr(msg,"toti utilizatorii")!=NULL || strstr(msg,"all users")!=NULL)
{
const char *filename="utilizatori.txt";
FILE *file=fopen(filename, "r");
char buf[2048];
char linie[128];
char list[2048];
bzero(list,2048);
strcat(list,"Lista tuturor utilizatorilor este:\n");
while (fgets(linie,sizeof(linie),file))
{
char *primul=strtok(linie," ");
strcat(list,primul);
strcat(list,"\n");
}
strcat(msgrasp,list);
strcat(msgrasp, "\nComanda ta: ");
}
else
if (strstr(msg,"intra in conversatie")!=NULL || strstr(msg,"open conversation")!=NULL)
{
strcat(msgrasp,"Cu cine: ");
de_deschis=1;
```

Fig. 6. Exemplul numarul 4 de cod sursa

In momentul in care utilizatorul este intrat in cont, acesta va avea dreptul sa execute comenzile aplicatiei care sunt bine predefinite.

```
else
if (strstr(msg,"trimite mesaj")!=NULL || strstr(msg,"send message")!=NULL)
{
ok_catre=1;
strcat(msgrasp,"Destinatar mesaj: ");
}
else
 {
 strcat(msgrasp,"Comanda inexistenta!");
 strcat(msgrasp, "\nComanda ta: ");
 }
break;
```

Fig. 7. Exemplul numarul 5 de cod sursa

Daca acesta va introduce o comanda care nu face parte din meniul aplicatiei, va primi un mesaj de eroare pentru a reintroduce o comanda valida.

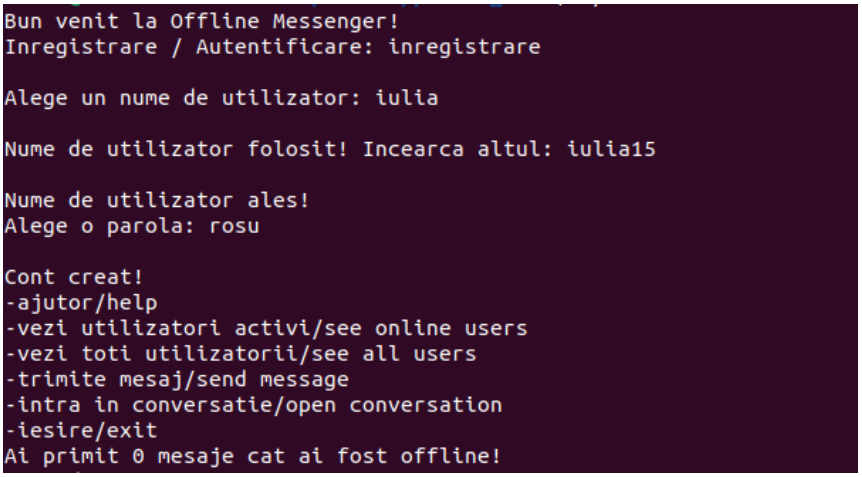


Fig. 8. Exemplul 1 de utilizare al aplicatiei: o posibila inregistrare

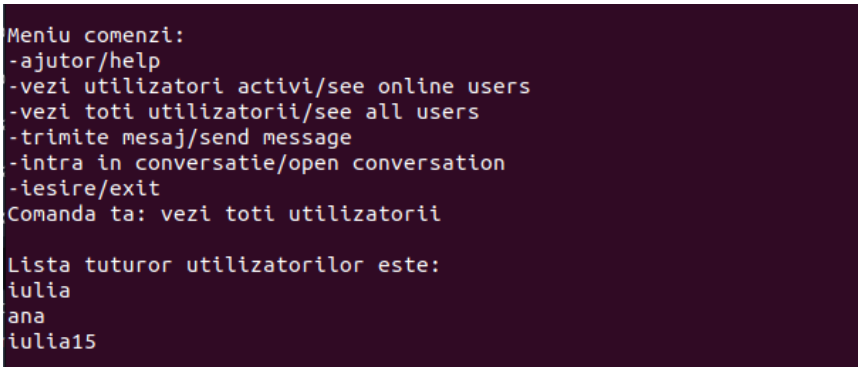


Fig. 9. Exemplul 2 de utilizare al aplicatiei: lista tuturor utilizatorilor care folosesc Offline Messenger

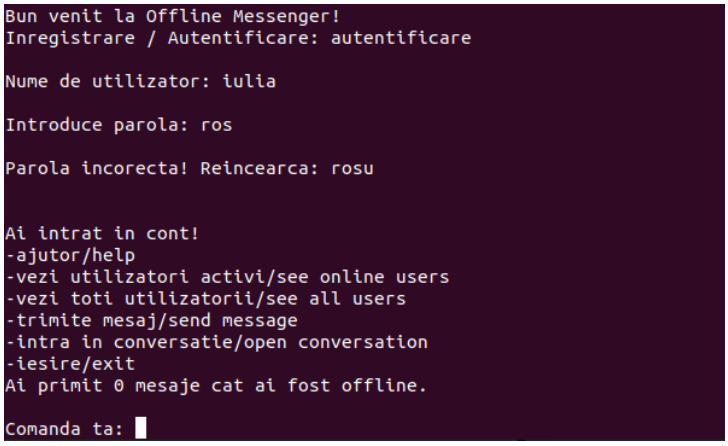


Fig. 10. Exemplul 3 de utilizare al aplicatiei: o autentificare cu o greseala la introducerea parolei

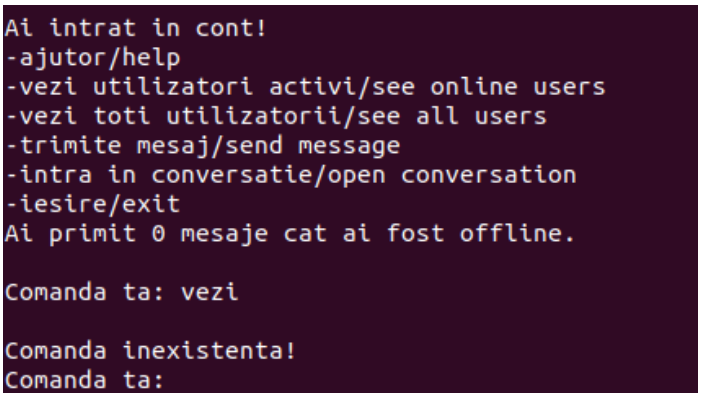


Fig. 11. Exemplul 4 de utilizare al aplicatiei: comanda inexistenta/incompleta

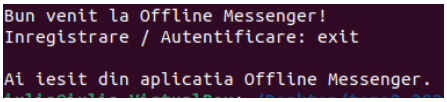


Fig. 12. Exemplul 5 de utilizare al aplicatiei: deconectarea

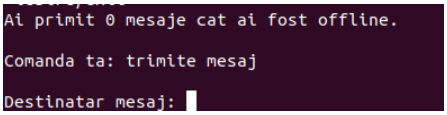


Fig. 13. Exemplul 6 de utilizare al aplicatiei: comanda de trimitere mesaj

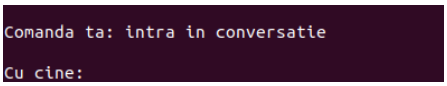


Fig. 14. Exemplul 7 de utilizare al aplicatiei: comanda de a vedea istoricul conversatiei

5 Concluzii

În concluzie, aplicația Offline Messenger permite schimbul de mesaje intre utilizatori care sunt conectati si ofera functionalitatea trimiterii mesajelor si catre utilizatorii offline. Cu toate acestea, există întotdeauna spațiu pentru îmbunătățiri menite să optimizeze experiența utilizatorului și să aducă funcționalitățile la un nivel superior. În primul rând, integrarea notificărilor mai detaliate și personalizabile ar putea contribui la o conștientizare mai bună a evenimentelor relevante, cum ar fi mesajele noi. De asemenea, implementarea unor opțiuni de filtrare și căutare a mesajelor ar putea facilita găsirea rapidă a informațiilor în cadrul conversațiilor. În plus, integrarea unei interfete grafice ar face totul mai placut vizual, adaugarea de butoane in locul scrierii unor comenzi ar face interactiunea cu clientul mai placuta. Aceste îmbunătățiri ar contribui la consolidarea poziției aplicației ca un instrument eficient și intuitiv pentru comunicarea offline.

6 Referințe Bibliografice

References

1. Pagina cursului si a laboratorului, https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
2. Schema TCP, https://profs.info.uaic.ro/ gcalancea/Laboratorul7.*pdf*