Proiect Offline Messenger

Aftanase Cosmin

December 1, 2020

Abstract

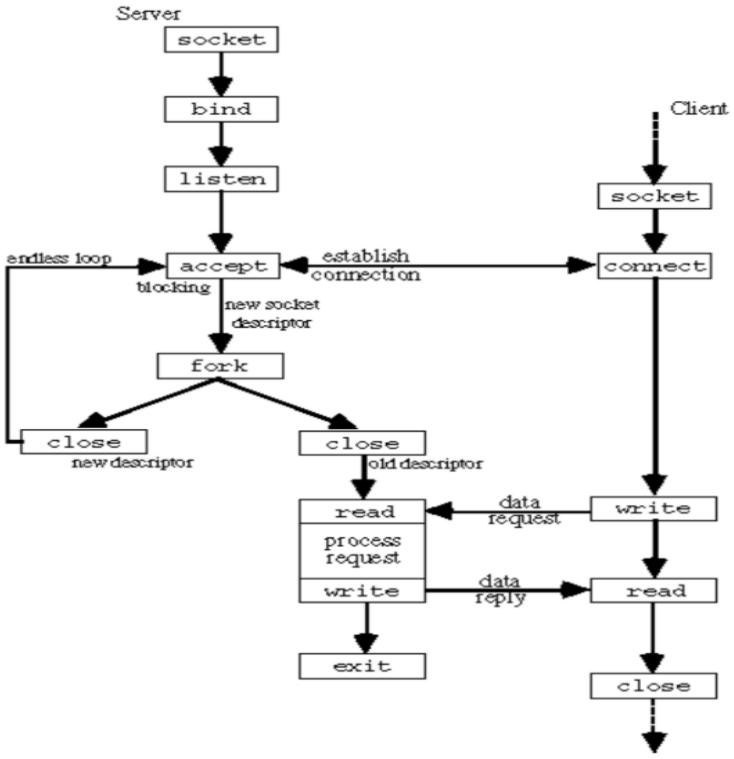
Prezentarea tehnologiilor utilizate, a arhitecturii si a detaliilor de implementare a proiectului Offline Messenger(B).

1 Introducere

Scopul acestui proiect este de a dezvolta o aplicatie client/server care sa permita schimbul de mesaje intre utilizatori care sunt conectati si sa ofere functionalitatea trimiterii mesajelor si catre utilizatorii offline, acestora din urma aparandu-le mesajele atunci cand se vor conecta la server. De asemenea, utilizatorii vor avea posibilitatea de a trimite un raspuns (reply) in mod specific la anumite mesaje primite. Aplicatia va oferi si istoricul conversatiilor pentru si cu fiecare utilizator in parte.

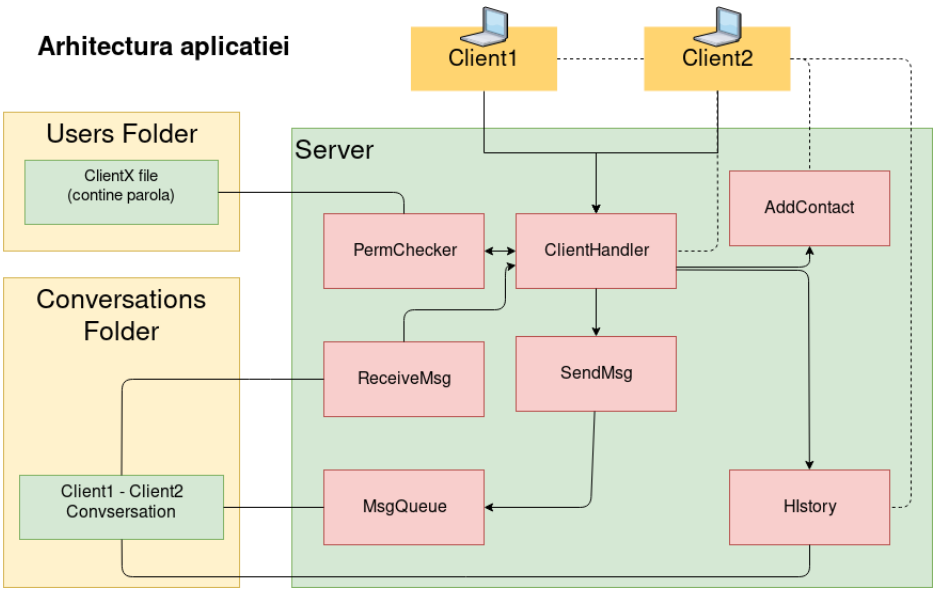
2 Tehnologiile utilizate

Pentru acest proiect am folosit Transmission Control Protocol (TCP).
TCP este un protocol de transport orientat, cu conexiune, **fara pierdere de informatii, ce controleaza fluxul de date**[1].
Spre deosebire de UDP, TCP-ul asigura faptul ca integritatea mesajelor este pastrata si ca mesajele utilizatorilor messenger-ului sunt trimise in ordinea lor cronologica.
De asemenea pentru acest proiect am folosit varianta TCP concurent in loc de TCP iterativ pentru a asigura dinamicitatea aplicatiei.
Scopul este de a le permite utilizatoriilor sa scrie si sa trimita mesaje in acelasi timp.

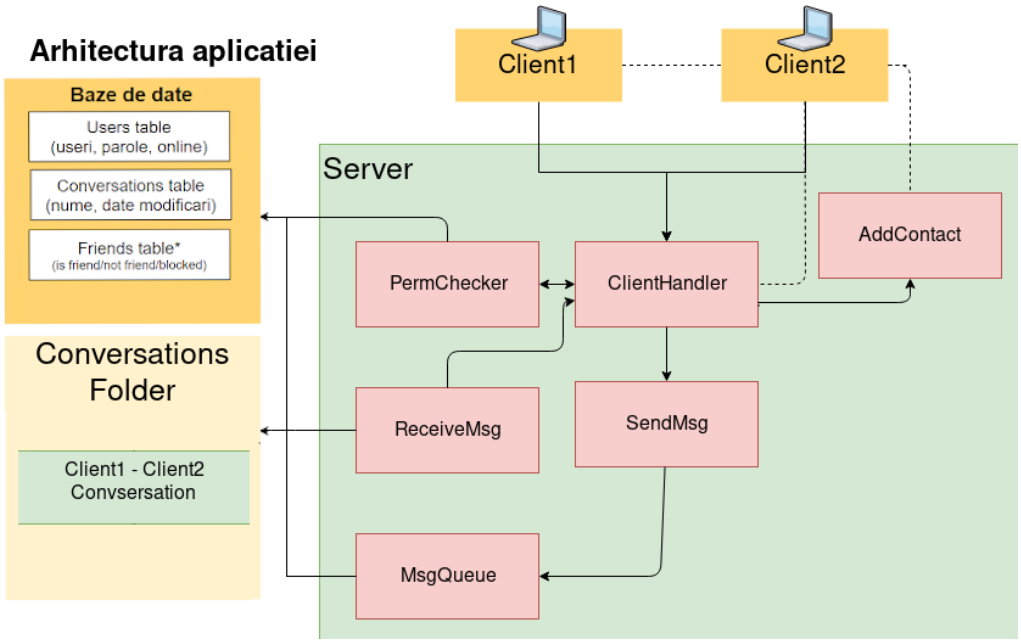


3 Arhitectura si detalii de implementare

Vechea arhitectura:



Noua arhitectura:

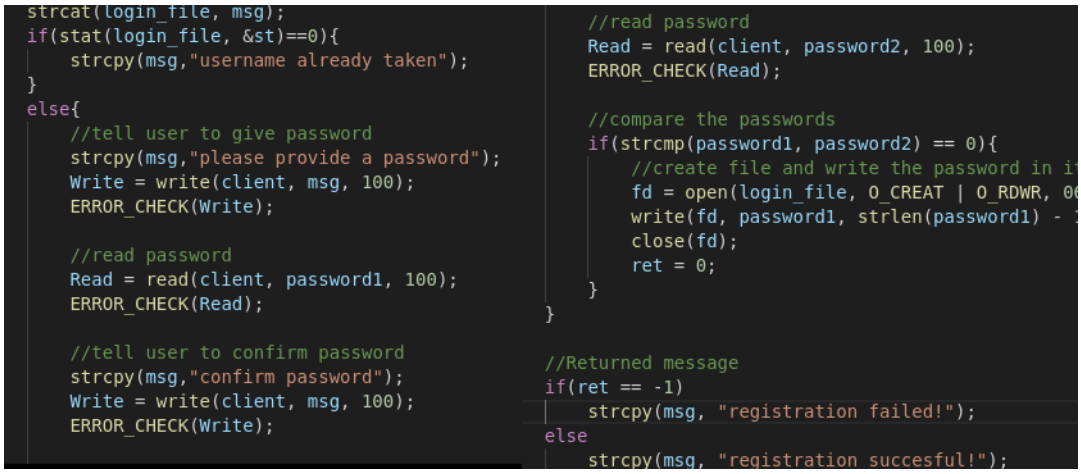


3.1 Componenta PermChecker

Permission Checker-ul se ocupa de implementarea comenzilor login si register. Orice alta comanda diferita de "/login" si "/register" va fi ignorata.

3.1.1 comanda "/register [nume]"

Vechea implementare:
Atunci cand se face register, in fisierul [nume] din folderul Users este adaugat un fisier cu numele utilizatorului ce va contine parola acestuia (parola va fi desigur verificata de 2 ori ca este scrisa corect inainte de a fi scrisa in fisierul respectiv, si daca acel fisier exista deja "/register" esueaza si retunreaza mesajul "username already taken").



```
strcat(login_file, msg);
if(stat(login_file, &st)==0){
    strcpy(msg,"username already taken");
}
else{
    //tell user to give password
    strcpy(msg,"please provide a password");
    Write = write(client, msg, 100);
    ERROR_CHECK(Write);

    //read password
    Read = read(client, password1, 100);
    ERROR_CHECK(Read);

    //tell user to confirm password
    strcpy(msg,"confirm password");
    Write = write(client, msg, 100);
    ERROR_CHECK(Write);
```

```
    //read password
    Read = read(client, password2, 100);
    ERROR_CHECK(Read);

    //compare the passwords
    if(strcmp(password1, password2) == 0){
        //create file and write the password in i
        fd = open(login_file, O_CREAT | O_RDWR, 0
        write(fd, password1, strlen(password1) -
        close(fd);
        ret = 0;
    }
}

//Returned message
if(ret == -1)
    strcpy(msg, "registration failed!");
else
    strcpy(msg, "registration succesful!");
```

Noua implementare:
Folderul users ce contine fisiere [nume] a fost inlocuit cu tabelul users care va tine minte numele si parola utilizatorilor. Acest tabel are in plus si coloana online care ne ajuta sa ne asiguram ca nu sunt 2 utilizatori conectati pe acelasi cont in acelasi timp.

3.1.2 comanda "/login [nume]"

Atunci cand se face login, se verifica daca parola introdusa corespunde cu cea din fisierul [nume]. De asemenea se verifica daca utilizatorul **nu este deja logat**.
Implementare veche: Acest lucru a fost implementat cu ajutorul functiei flock. Atunci cand un utilizator se logheaza, se incearca sa se puna lacat pe fisierul respectiv. Daca punerea lacatului a esuat inseamna ca cineva este deja logged in.

```
//Check if file(username) exists
strcpy(user, msg);
strcat(login_file, msg);
if(stat(login_file, &st)==0){

    //Check for password
    strcpy(msg, "please provide a password");
    Write = write(client, msg, 100);
    ERROR_CHECK(Write);

    //Read password
    Read = read(client, msg, 100);
    ERROR_CHECK(Read);

    //Verify password
    log_fd = open(login_file, O_RDONLY);
    pread(log_fd, password, 100, 0);
    msg [strlen(msg) - 1] = '\0';
    password[strlen(msg)] = '\0';
    //Password matched!
    if(strcmp(password, msg) == 0){
        //Check if user is not already logged in
        if (flock(log_fd, LOCK_EX | LOCK_NB) == -1){
            ret = -2;
        }
        else {
            ret = 0;
        }
    }
}

//Returned message
if(ret == 0)
    strcpy(msg,"login succesfull");
else if(ret == -1)
    strcpy(msg,"login failed, try again");
else if(ret == -2)
{
    strcpy(msg, "somebody is already logged in!");
```

Implementare noua: Se verifica daca coloana online a user-ului este 0 (offline). In caz afirmativ, aceasa este schimbata in 1 si este permisa logarea la server.
Daca utilizatorul a reusit sa se logheze, poate poate sa foloseasca comnezile din Client Handler.

3.2 Componenta ClientHandler

Client Handler-ul este componenta ce redirectioneaza mesajele catre celalte componente, in functie de inputul clientilor. Comanda "/help" va afisa pe ecran lista tuturor comenzilor.
Totodata Client Handler-ul va primi semnal ca a fost primit un mesaj de la alt user, via ReceiveMsg, si il va lasa sa afiseze de la cine a fost primit mesajul respectiv.
Implementare noua: Comanda "/logout" va intoarce user-ul catre componenta PermChecker si va scoate lacatul de pe fisierul cu numele acestuia din folderul Users (pentru a permite conectarea/ reconectarea la acel cont).
Implementare veche: Comanda "/logout" actualizeaza tabelul users setand coloana online de la 1 la 0 pentru utilizatorul respectiv si va intoarce user-ul spre componenta PermChecker.

3.3 Componenta Contacts

Pentru a initia o conversatie cu cineva, utilizatorul trebuie sa il aibe in lista de prieteni. Aceasta componenta este cea care verifica daca 2 utilizatori sunt prieteni si este cea care creeaza fisierul conversatiei.
Nota: pentru a evita creearea a doua fisiere, de exemplu: ana-mihai si mihai-ana, componenta Contacts creaza si verifica existenta fisierului luand in ordine

lexicografica numele celor doi.
Cu alte cuvinte fie ca ana il adauga la prieteni pe mihai sau ca mihai o adauga la prieteni pe ana, pentru amandoi va fi creat fisierul "ana-mihai" folosind primitiva open("ana-mihai", O_RDWR | O_CREAT, 0777). Open asigura faptul ca fisierul nu va fi creat de 2 ori (O_CREAT creaza fisierul doar daca acesta nu exista deja).

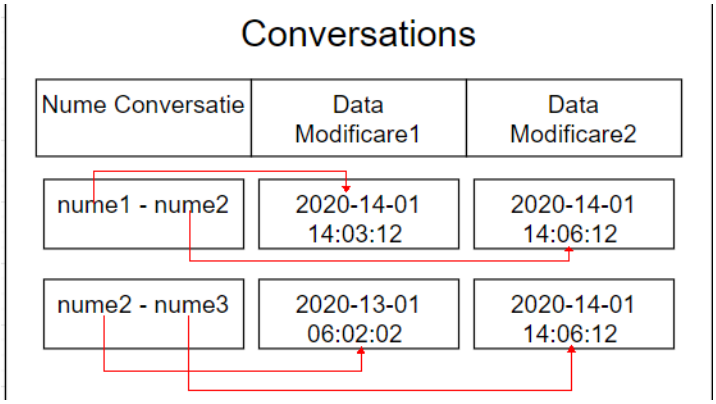
```
//Add him as a friend
file_name = computeConvName(ourUser, targetUser);
if(stat(file_name, &st) != 0){
    int ConversationFile = open(file_name, O_CREAT | O_RDWR, 0
    ERROR_CHECK(ConversationFile);
    close(ConversationFile);
    strcpy(msg, "friend added (todo: ask for his consent)");
    Write = write(client, msg, 100);
    ERROR_CHECK(Write);
    ret = 0;
}
else{
    strcpy(msg, "you are friends!");
    Write = write(client, msg, 100);
    ERROR_CHECK(Write);
}

return ret;
```

```
har *computeConvName(char name1[100], char name2[100]){
  char *ConvName = (char*) malloc(200);
  char computed[200];

  if (strcmp(name1, name2) < 0){
      strcpy(ConvName, "./conversations/");
      strcpy(computed, name1);
      strcat(computed, " - ");
      strcat(computed, name2);
      strcat(ConvName, computed);
  }
  else {
      strcpy(ConvName, "./conversations/");
      strcpy(computed, name2);
      strcat(computed, " - ");
      strcat(computed, name1);
      strcat(ConvName, computed);
  }
```

Conversatia va fi adaugata in **tabelul Conversations** unde vom tine evidenta utilizatorilor care nu sunt la curent cu modificarile din text file-urile conversatiilor lor.



Comanda folosita pentru a adauga contacte este "/add [nume]". In momentul de fata, comanda add nu cere si consimtamantul persoanei adaugate atunci cand este adaugata la prieteni.

Idee de implementare: in momentul cererii, receptorul va primi mesajul pe ecran si va avea de ales dintre 3 optiuni: 1.accept (daca accepta prietenia), 2.refuz (refuza pe moment prietenia, lasand posibilitatea de a primi din nou cerere), 3.blocare (emitatorul nu va mai putea trimite cereri de prietenie).

3.4 Componenta SendMsg

Send message preia mesajul si ii adauga in fata ora la care a fost trimis, emitatorul si receptorul mesajului, urmat de mesajul in sine.
Acest mesaj este dupa trimis catre Componenta MsgQueue.
SendMsg verifica daca utilizatorul vorbeste cu o persoana si daca persoana catre care a fost trimis mesajul este in lista de prieteni(comunica cu componenta Contacts).

```
//Check if is talking to somebody
strcpy(ourUser, user);
strcpy(beforeMsg, msg);
while (strlen(targetUser) == 0){
    if( targetUserFunction(client, msg) == -1)
        return -1;
}

//Check if ourUser is friend with targetUser
if( checkContact(client, ourUser, targetUser) == -1)
{
    strcpy(targetUser, "");
    return -1;
}

//Add date and person you are talking before message
strcpy(msg, beforeMsg);
time = getTime();
strcpy(computed, time);
strcat(computed, ourUser);
strcat(computed, " > ");
strcat(computed, targetUser);
strcat(computed, ": ");
strcat(computed, msg);
computed[strlen(computed) - 1] = '\0';
```

Orice input diferit de comenzile de mai sus este luat drept mesaj.
Comanda "/reply" va fi urmata de tasta (W), respectiv tasta (S) pentru a alege mesajul la care se doreste a se raspunde.

3.5 Componenta MsgQueue

Implementarea veche: Dupa cum am mentionat mai sus Message Queue primeste mesajele de la Send Message, le serializeaza si le adauga intr-un array, urmand a fi adaugate dupa ce devine disponibil fisierul.

Implementarea noua: Message queue creeaza cate un thread pentru fiecare mesaj ce se ocupa cu scrierea acestuia in fisier. Aceste thread-uri incearca sa

obtina, folosind flock, lacat peste fisierul "[nume1]-[nume2]" din folderul Conversations pentru a scrie mesajul primit.

3.6 Componenta ReceiveMsg

Implementarea veche: Receive Message va monitoriza folder-ul conversatii pentu schimbari, folosind functia inotify si va verifica daca numele fisierului modificat contine numele utilizatorului. In caz de schimbare a fisierului "[nume1]-[nume2]" va trimite un semnal componentei ClientHandler ca utilizatorul a primit un mesaj nou.

Implementarea noua: Receive Message se uita peste tabela conversatii si cauta fisierele in care datetime-ul user-ului nostru este mai mic (adica au aparut modificari de care nu a fost instiintat).
Acesta comunica pe un **alt thread** cu client handler si ii anunta daca avem mesaje noi.
De asemenea acesta se ocupa in aceasta implementare si de comandata "/history" ce ne va arata intreagul conversation file-ul cu o anumita persoana si ajuta componenta SendMessage cu comanda "/reply".

4 Utilizare

Utlizatorul va fi nevoit sa se logheze folosind comanda "/login" (sau sa isi creeze cont folosind comanda "/register"). Dupa, folosind comanda "/user [nume]" isi poate alege un "targetUser" cu care sa vorbeasca. Orice mesaj ce nu e o comanda enuntata mai sus va fi luat drept mesaj. Mesajul nu va fi trimis daca nu are un targetUser setat (emitatorul va fi intrebat cu cine vorbeste) sau daca targetUser-ul respectiv nu exista sau nu este in lista de prieteni. Orice mesaj trimis va fi scris in fisierul conversatiei dintre cei doi. Cand un mesaj este primit inapoi (via ReceiveMsg), vom primi o notificare de la cine. Se vor putea folosi comenzile "/help" pentru a afla lista de comenzi, "/user [nume]" din nou pentru a schimbat targetUserul, "/add [nume]" pentru a adauga pe cineva la prieteni, "/history" pentru a vedea istoricul conversatiei, "/reply" pentru a da reply la un mesaj sau a vedea mesajele, "/logout" pentru a iesi de pe contul curent si a reveni la inceput si "/quit" sau ctrl+c pentru a iesi din program.

5 Concluzii

Am folosit in implementare TCP concurent si am impartit toate functiile ce se ocupa de comenzile respective in 6 componente(plus una de sql). Aceste componente comunica intre ele si cu clientii si se folosesc de un folder numit Conversations (ce contine fisiere sub forma "nume_client1-nume_client2" cu conversatiile dintre clientul1 si clientul2) si o baza de date cu 2 tabele, una pentru useri (pentru nume, parole, stare de activitate) si pentru conversations (pentru a fi instiintati in legatura cu mesajele primite).

Idei de dezvoltare: o prima idee ar fi implementarea unei interfete grafice. Pe langa aceasta ar mai fi la componenta **Contacts cererea consimtamantului**, mentionata mai sus. o idee ar fi ca toate relatiile dintre utilizatori sa fie tinute minte intr-un tabel cu 3 coloane (similar cu cel al conversatiilor), o coloana text nume si 2 coaloane user1 si user2 ce pot avea valori 0, 1 si -1 (0 inseamnand ignorare, 1 prietenie si -1 block).
Pentru ca cei 2 useri sa fie prieteni trebuie ca atat in user1 cat si in user2 sa fie 1, altfel nu isi pot trimite mesaje. Daca unul din useri are -1 la coloana, cererile de prietenie de la celalat utilizator vor fi ignorate pentru a evita spamul.

References

[1] https://profs.info.uaic.ro/ computernetworks

[2] https://sites.google.com/view/fii-rc/