Tema 3

Stamate Valentin 2B4

Universitatea "Alexandru Ioan Cuza" din Iasi
Facultatea de Informatica

Abstract. Acest raport contine detalii arhitecturale despre un proiect realizat in limbajul de programare c precum si cod relevant.

1 Introducere

Raportul contine arhitecturala si cateva detalii de implementare a unei aplicatii realizate in limbajul de programare c. Aplicatia exemplifica modul cum procesele pot comunica intre ele nu numai pe masina locala dar si pe dispozitive diferite. Astfel proietul ales este Part2Part iar motivatia din spatele alegerii este pasiunea de a face software si a curiozitatea cu privire la modul cum aplicatiile de tip server/client pot comunica la distante uriase. Aplicatia va functiona similar cu o aplicatie de tip peer-to-peer precum uTorrent. Utilizatorul va putea cauta in fisierele celorlalti utilizatori putand chiar filtra rezultatele prin denumirea fisierului, marimea lui si extensie.

2 Tehnologii Utilizate

Pentru comunicarea intre server si client se va folosi protocolul TCP/IP pentru a raspunde la clienti in mod concurent si pentru a asigura trimiterea datelor in mod sigur, spre deosebire de UDP unde datele trimise nu vor fi trimise la destinatie in mod garantat. Pentru comunicarea cu clientii se vor folosi thread-uri deoarece sunt mai eficiente spre deosebire de noi procese create cu primitiva fork() care copie toata memoria procesului parinte, operatie ce este forarte costisitoare cu privire la memorie si timpul de executie, spre deosebire de firele de executie care partajeaza memoria cu procesul parinte.

La fiecare trimitere catre server sau client, raspunsuruile de tip sir de caracter vor fi prefixate de lungimea de lor pentru a se cunoaste cat sa se aloce in buffer inaintea citirii mesajului propriu zis.

Pentru stocarea datelor va fi folosita o baza de date SQLite pentru a putea stoca utilizatorii, deoarece este o foarte cunoscuta iar interogarile pot fi usor extrase. Libraria folosita pentru interactia cu baza de date este *sqlite3*.

Serverul va functiona similar unui REST API unde raspunsurile vor fi procesate de primitiva *write()* iar preluarea unui request prin primitiava *read()*. Astfel clientii vor trimie un tip de request, acesta este preluat de server si pe baza acestuia trimite inapoi raspuns catre client. La conectare, fiecarul utilizator I se va asocia un thread care va

permite procesarea cererii. Astfel serverul va putea servi in mod simultan mai multe cereri de la clienti.

Fiecare client o sa aiba asociat cate doua threaduri, cu rolul de a transmite fisierul cautat de alt client, iar celalalt pentru a trimite clientului fisierul dorit. Acestea au rol de ascultator adica ele asteapta un mesaj de la server cu datele necesare pentru a trimite mai departe raspunsul.

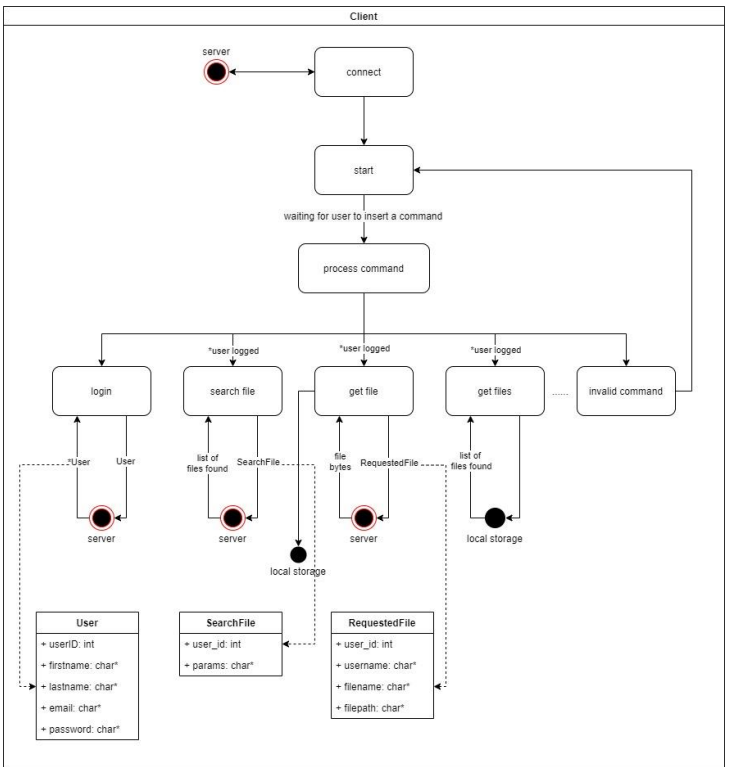
Conectarea la retea se face in mod automat. Astfel nu va trebui cautat manual ip-ul routerului. Aceasta se realizeaza prin folosirea comenzii *ifconfig* impreuna cu alte comenzi pentru a detecta ip-ul. Portul pentru conectare va fi intotdeauna 1024.

3 Arhitectura Aplicatiei

In imaginile de mai jos este infatisata arhitectura aplicatiei. Pentru a putea fi mai clara, arhitectura este impartita pe nivele: client, server si baza de date. Deoarece clietul se poate afla pe un dispozitiv diferit fata de server mediul de comunicare dintre cei doi va fi un router wireless adica serverul si fiecare client se va conecta la adresa ip a routerul respectiv. Astfel dispozitivul care ruleaza serverul, respectiv clientii va trebui sa fie conectat la aceeasi retea wireless sau prin cablu.

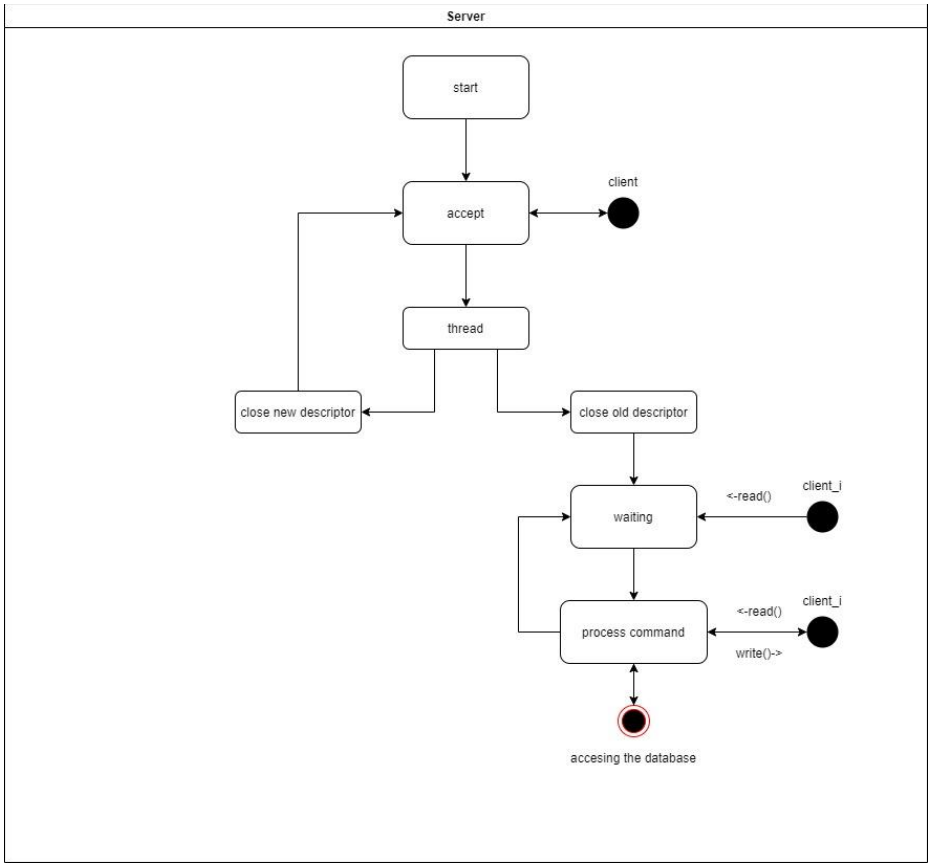
Baza de date va contine un tabel care va contine utilizatorii inregistrati. Atributele asociate tabelului sunt id, username, email, password.

- **Client**



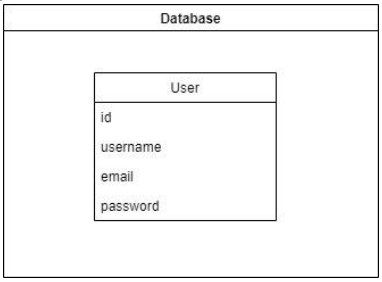
Dupa ce conexiunea are loc cu succes, clientul asteapa de la utilizator introducerea unei comenzi. Comanda este apoi procesata si in functie de caz va intra pe una din ramurile exemplificate mai sus de pe nivelul 4. Clientul trimite date serverului si va primi inapoi un raspuns, dupa care va astepta din nou urmatoarea comanda.

- **Server**



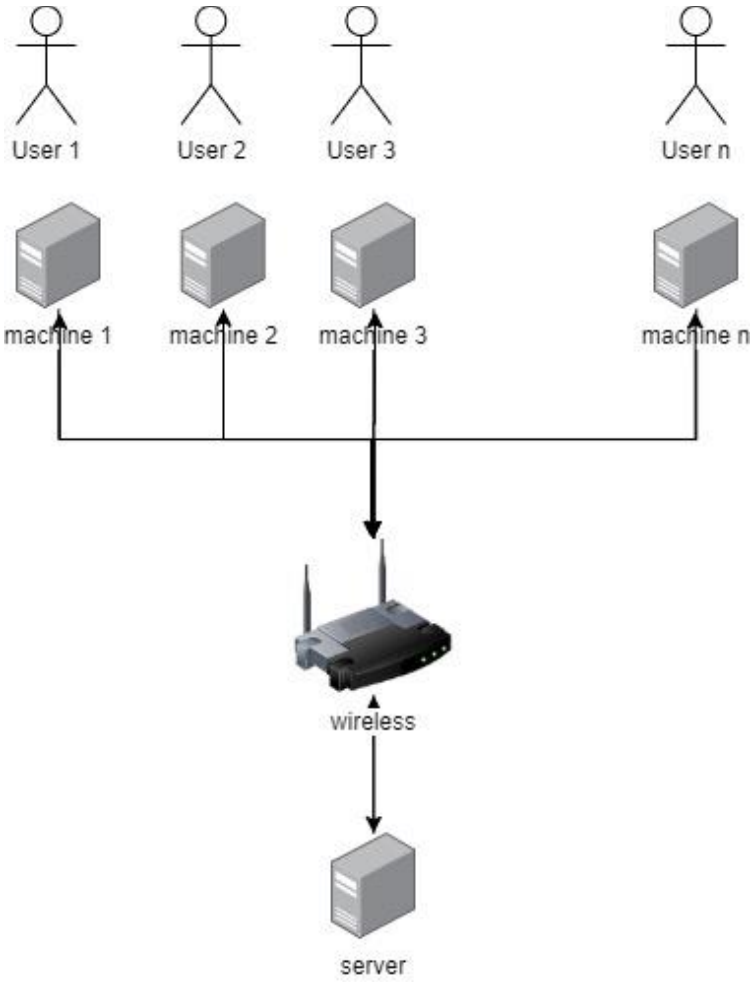
Aici fiecare thread este asociat unui client exemplificat prin denumirea *client_i*. Fiecare thread asteapta de la clientul asociat tipul de request dupa care in functie de cerere ii este intors raspunsul.

- **Baza de date**

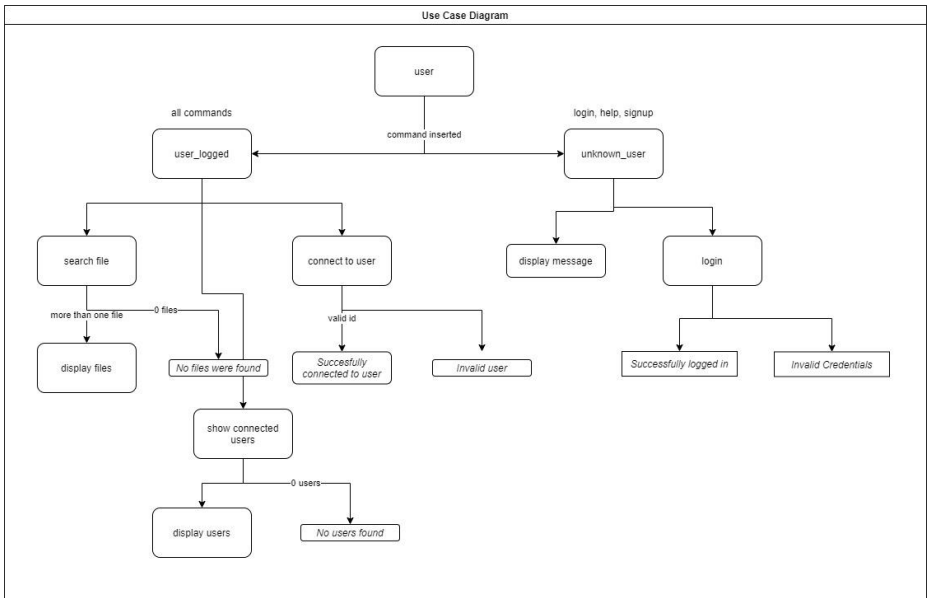


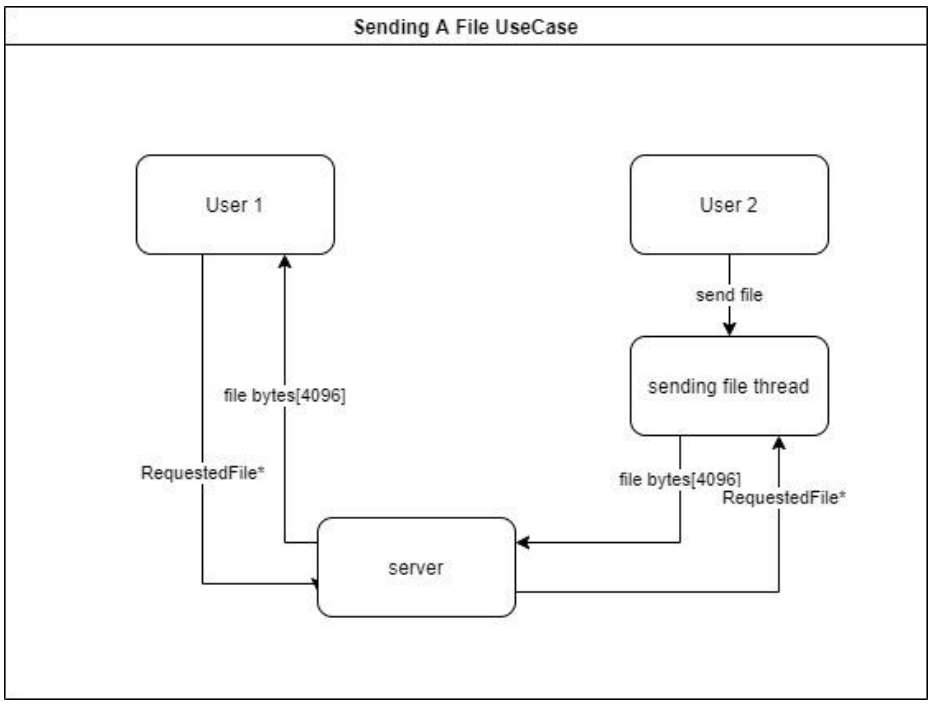
Baza de date are o tabela pentru utilizatori, iar atributele acesteia se pot vedea in imagine cu cheia primara id.

- **Arhitectura fizica**



Aici este un model fizic al utilizatrii aplicatiei. Pot exista mai multe dispozitive care pot transfera fisiere intre ele dar un singur server. Toate dispozitivele trebuie sa fie conectate la aceeasi retea.





Mai sus este descris modul cum se transmite un fisier de la un utilizatori la altul. Utilizatorul 1 transmite datele necesare preluarii unui fisier serverul. Accesta face o cautare prin toti utilizatorii conectati si cauta utilizatorul al carui id corespunde cu cel

din structura transmisa. In acest timp utilizatorul 1 asteapta primirea fisierului. Serverul, dupa gasirea utilizatorului si a threadului asociat, transmite catre socketul de descarcare structura de date RequestedFile. Threadul utilizatorului 2 ramane in „repaus" pana cand comanda ‚send file' este folosita. Apoi threadul primeste datele necesare si transmite inapoi in packete a cate 4kb fisierul dorit. Dimensiunea lui este prefixata de numarul de octeci citici, in cazul in care dimensiunea fisierului nu este multiplu de 4096.

4 Detalii de Implementare

Utilizatorii care nu sunt inregistrati in baza de date nu vor putea executa comenzi inafara de comenzile *help*, exit, login si signup afisand in loc un mesaj de eroare. In cazul in care o comanda nu exista se va o notificare cu mesajul „The command doesn't exist."

Pentru o afisare mai buna a mesajelor, clientii le vor primi sub forma de notificari. Acestea reprezinta fie mesaje de notificare, fie liste cu rezultate.

Fiecare client poate vedea lista utilizatorilor curent conectati. Astfel, se poate crea o conexiune intre orice clienti. El poate vedea lista utilizatorilor cu care este conectat si poate efectua cautari specificand id-ul utilizatorului. Dupa efectuarea cautarii i se va intoarce numarul de fisiere gasite urmati de numele lor, dar si un id atasat acestora. In cazul in care numarul fisierelor este egal cu 0 se va afisa notificarea „*No files were found.*". Dupa aceasta comanda utilizatorul poate selecta fisierul dorit pentru transfer. Pentru a pastra siguranta datelor, clientul care detine fisierul dorit trebuie sa foloseasca comanda *send file* pentru a permite transferul. Fisierele transferate sunt salvate in folderul *downloads* ce pot fi vazute folosing comanda *show downloaded files*.

Urmatoarele functii vor fi incluse in proiect dar si documentate. Ele nu vor fi puse in intregime intrucat ele pot fi foarte lungi. Doar partea esentiala se va afisa.

- **Server**

```
sqlite3* openDatabase(char* databaseName) {
    sqlite3* db;

    sqlite3_open(databaseName, &db);

    SQLExecute(db, "PRAGMA foreign_keys = ON");

    return db;
}
```

Functia *openDatabase* primeste ca parametru un sir de caractere ce reprezinta locatia bazei de date. Dupa ce ce se face conexiunea cu baza de date, instanta ei este salvata in structur-o structura de tip pointer *sqlite3* si returnata pentru interactiunile viitoare.

SQLExecute este o functie ce ruleaza o interogare in baza de date. "PRAGMA foreign keys = ON" activeaza folosirea cheilor straine.

```
void insertUser(sqlite3* db, User* user) {
    char sql[1250];

    sprintf(sql, "INSERT INTO users(firstname, lastname, em
ail, password) VALUES ('%s', '%s', '%s', '%s');", user-
>firstname, user->lastname, user->email, user->password);

    SQLInsert(db, sql);

    User u = getUserByEmail(db, user->email);
    user->userID = u.userID;

}
```

Aceasta functie introduce un utilizator in baza de date. Ca si parametri functia preia o instanta a bazei de date, si datele userul care se vrea a fi introdus. Daca utilizatorul este deja existent se va afisa o eroare. Dupa ce a fost adaugat, se va cauta din nou userul pentru a prelua totate metadatele associate unui user cum ar fi id-ul unui utilizator care se va cunoaste dupa ce a fost introdus in baza de date.

```
for(;;) {
    int client;
    thData *td;
    int length = sizeof(from);

    if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0) {
        perror("[server] accept() error\n");
        continue;
    }
    ...
```

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

    tdDat[nTdData].idThread = nTdData;
    tdDat[nTdData].sdCl = client;

    pthread_create(&th[nTdData], NULL, &treat, tdDat + nTdData);

    nTdData++;
}
```

Aici se face conexiunea cu fiecare client folosind protocolul TCP/IP. La fiecare loop, executia va ramane blocata la functia accept pana cand un nou utilizator se va conecta. In momentul conexiunii se pun datele dorite(cum ar fi descriptorul de socket al clientului) intr-o instanta a structurii de date thData ce este pus intr-un vector si este transmisa threadului nou creat ce se va ocupa de requesturile utilizatorului. Vectorul are rolul de a conecta doi clienti fie prin transferul de fisiere, fie prin cautari.

```
void process_request (void *arg) {
    thData *tdL;
    tdL = (struct thData *)arg;

    int sd = tdL->sdCl;
    repeat:
    int REQUEST_TYPE;

    if (read(sd, &REQUEST_TYPE, sizeof(int)) <= 0) {
        printf("[Thread %d] ", tdL.idThread);
        perror("read() error\n");
    }
    ...
    switch (REQUEST_TYPE) {
    case LOGIN: ;
        printf("Login Request\n");

        if (read(sd, &u, sizeof(User)) == -1) {
```

```
            perror("[server] " READ_ERROR);
            tdL->user_id = -1;
            return;
        }

        verifyUser(db, &u);

        if (u.userID != -1) {
            sprintf(tdL->user_email, "%s", u.email);
            tdL->isActive = 1;
        }

        // allow only one sesion for users
        for (int i = 0; i < nTdData && u.userID != -1; i++) {
            if (tdDat[i].user_id == u.userID) {
                u.userID = -2;
                break;
            }
        }
        tdL->user_id = u.userID != -2 ? u.userID : -1;

        if (write(sd, &u, sizeof(User)) == -1) {
            perror("[server] " WRITE_ERROR);
            tdL->user_id = -1;
            return;
        }
        break;
    case: OTHER_CASES: ;
        .....
        break;
    default:
        break;
    }
    printf("Request end\n\n");
}
```

Functia *raspunde()* preia si proceseaza requestul dat de client. Initial se citeste tipul de request (de ex *LOGIN*) ce reprezinta un integer definit de un macro intr-un fisier .h, iar intr-un Switch Case se interationeaza cu clientul conform requestului respectiv. In cazul de mai sus, avem LOGIN unde se citeste o structura *User*. Se verifica utilizatorul

cu emailul si parola si completeaza instanta u cu datele asociate. In caz de esec, se pune in structura u, id = -1. Aceasta va semnala ca userul nu exista sau ca credentialele sunt invalide. Se intoarce instanta catre client si se termina requestul. Clientul va prelua structura iar daca id-ul este -1 se afiseaza notificarea „*User does not exist.*”

Pentru alte tipuri de requesturi cum ar fi *SEARCH_FILE* unde este necesara autentificarea utilizatorului. Pentru recunoasterea acestuia, clientul, dupa ce pune tipul de request, va pune dupa si o structura de tip SearchFile ce contine mai multe tipuri de date inclusiv id-ul userului. Se preia dupa socketul asociat threadului de cautare si se pune structura instanta SearchFile. Threadul asteapta apoi raspusul si transmite mai departe clientului ce a facut request fisierele gasite.

- **Client**

```
if (connect(sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) == -1) {
    perror (CONNECT_ERROR);
    return errno;
}
...
pthread_t thF;
pthread_create(&thF, NULL, &treat, &sdFt);

pthread_t thSr;
pthread_create(&thF, NULL, &treat_search, &sdSr);

repeat:
...
int COMMAND_TYPE = process(command, blocks);
...
switch (COMMAND_TYPE) {
    case LOGIN:

    sendLoginCredentials(sd, command, user);

    isLogged = (user->userID != -1);

    if (user->userID == -2) {
        isLogged = 0;
        pushNotif(BYEL "Only one session is allowed per user" reset);
        break;
    }
```

```
    if (isLogged == 1) {
        pushNotif(tempLine);

        initializeTransferDescriptors(sd, &sdFt, &sdSr, user, ip, port);
    } else {
        pushNotif(BRED "Invalid credentials" reset);
    }
    break;
case: OTHER_CASES: ;
        .....
        break;
    default:
        break;
    }
```

Dupa ce clientul se conecteaza cu serverul utlizatorul va putea introduce comenzi. Comanda introdusa este prelucrata(spargerea ei in tocken-uri si eliminarea spatiilor) si asocierea ei cu tipul de request(LOGIN, LOGOUT, etc). Acel Switch Case face apel la server in functie de tipul de request. Utilizatorii care nu sunt logati nu vor putea rula comenzi ina fara de help, exit, login si signup. Cand utilizatorul se logheaza trimite serverului datele si daca intanta User trimisa catre server are id-ul -2, insemna ca utilizatorul este deja logat, iar daca id-ul este -1, utilizatorul va primi notificarea *„Invalid credentials”*. Dupa terminarea comunicarii cu serverul se asteapta din nou introducerea unei comenzi la tastatura.

```
void initializeTransferDe-
scriptors(int sd, int* sdF, int *sdSr, User* user, char* ip, int port) {
    struct sockaddr_in socket_file;
    struct sockaddr_in socket_search;

    int type = CONNECT_TRANSFER;

    socket_file.sin_family = AF_INET;
    socket_file.sin_addr.s_addr = inet_addr(ip);
    socket_file.sin_port = htons(port);

    if (((*sdF) = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
        perror (SOCKET_ERROR);
        return;
    }
```

```
    if (connect ((*sdF), (struct sockaddr *) &socket_file,sizeof (struct sock-
addr)) == -1) {
        perror (CONNECT_ERROR);
        return;
    }

    write((*sdF), &type, sizeof(int));
    write((*sdF), user, sizeof(User));

    type = CONNECT_SEARCH;

    socket_search.sin_family = AF_INET;
    socket_search.sin_addr.s_addr = inet_addr(ip);
    socket_search.sin_port = htons(port);

    if (((*sdSr) = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
        perror (SOCKET_ERROR);
        return;
    }

    if (connect ((*sdSr), (struct sock-
addr *) &socket_search,sizeof (struct sockaddr)) == -1) {
        perror (CONNECT_ERROR);
        return;
    }

    write((*sdSr), &type, sizeof(int));
    write((*sdSr), user, sizeof(User));
}
```

Cand logarea utilizatorul se efectueaza cu succes, se initilizeaza doi socketi, unul pentru cautarea fisierelor si unul pentru descarcarea fisierelor. Acesti doi socketi sunt procesati in care un thread ce are rol de ascultare, adica in momentul cand un alt client efectueaza o cautare/cerere de descarcare, serverul trimite un mesaj catre socketul de caurare/descarcare si primeste inapoi raspuns.

Structura de comunicare TCP/IP este preluata din codul titularului de curs si se gaseste in original *aici* pentru server si *aici* pentru client.

5 Concluzii

Aplicatia reprezinta un exemplu de conectare a mai multe procese aflate pe diferite dispozitive conectate la aceeasi retea prin protocolul TCP/IP. O imbunatatire a acestei aplicatii ar fi introducerea unei interfete grafice cu drag and drop pentru o experienta mai placuta a utilizatorului. Astfel poate fi extins putand fi folosita ca si aplicatie p2p similara cu uTorrent pentru distribuirea de fisiere intre mai multe dispozitive.

De asemenea, proiectul poate fi modificat prin adaugarea protocolului cu http pentru ca serverul sa foloseasca drept server web iar impreuna cu tehnologiile actuale web sa se poata realiza aplicatii fara a fi nevoie de cumpararea unui server online.

Referinte

1. LNCS Homepage, http://www.springer.com/lncs
2. Crearea unui tabel: https://www.sqlitetutorial.net/sqlite-create-table/
3. Cum se foloseste sqlite : https://youtu.be/-C5HSdPU3TI
4. Documentatia oficiala sqlite3 : https://www.sqlite.org/cintro.html
5. Define pe mai multe linii : https://stackoverflow.com/questions/6281368/multi-line-define-directives
6. Null nu e definit: https://www.educative.io/edpresso/what-is-the-null-undeclared-error-in-c-cpp
7. Crearea unui header : https://stackoverflow.com/questions/7109964/creating-your-own-header-file-in-c
8. Testarea interogarilor SQLite : https://sqliteonline.com/
9. Get Columns From Ibterroagration : https://stackoverflow.com/questions/142789/what-is-a-callback-in-c-and-how-are-they-implemented : https://stackoverflow.com/questions/11637701/sqlite-c-interface-get-single-value-result
10. Foreign Keys : https://stackoverflow.com/questions/5890250/on-delete-cascade-in-sqlite3
    Exemplu de server TCP : https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
11. Exemplu de Client TCP :
12. https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
13. Convertirea unei adrese IP de tip text :
14. https://www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html
15. Cursul 7
16. Connect local server to router ip : https://www.oreilly.com/library/view/linux-network-administrators/0596005482/ch04.html
17. Functii din tema 1(trimString(), getBlocks(), process().etc)
18. ANSI color coldes : https://gist.github.com/RabaDabaDoba/145049536f815903c79944599c6f952a
19. Eroare swich case https://www.educative.io/edpresso/resolving-the-a-label-can-only-be-part-of-a-statement-error
20. Malloc : https://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm
21. Move file from a directory to another: https://askubuntu.com/questions/172629/how-do-i-move-all-files-from-one-folder-to-another-using-the-command-line