MyFileTransferProtocol*

Burcă Paul

Facultatea de informatică, Univarsitatea Alexandru Ioan Cuza, Iaşi, România
https://www.info.uaic.ro/

Abstract. În cele ce urmează voi prezenta modul de contructie şi functionare al unei aplicaţii de tip FTP, aplicaţie cu rol de transfer de fişiere, dar şi de stocare a datelor într-un server. Textul următor va conţine o introduce despre conceptul de FTP şi cum ar trebui implementat, tehnologiile folosite (TCP sau UDP...), arhitectura pe care o ia aceasta cu scopul de a rezolva cererile date de clienţi, anumite detalii de implementare (diferite secţiuni de cod, diagrama reprezentând modul de funcţionare), nişte concluzii finale aratând cum ar putea acest proiect să fie îmbunătăţit şi sursele folosite atât în cadrul acestui raport, cât şi în cadrul dezvoltării aplicaţiei.

Keywords: FTP · TCP · DB · Sockets · Computer Networks · Encryption · Qt · Threads · TLS.

1 Introducere

1.1 Descriere

MyFileTransferProtocol (B): o aplicaţie client/server ce permite transferul de fisiere între clienţi şi server. Serverul va pune la dispoziţia clientilor un număr minim de comenzi ce permit autentificarea, operarea cu directoare şi cu fişiere. De asemenea, va trebui implementat un mecanism de autorizare (whitelist/blacklist) pentru conturile utilizatorilor şi un mecanism de transmitere securizată a parolei la autentificare.

1.2 Motivaţie

Am ales acest proiect deoarece aş vrea ca să-l pot folosi de zi cu zi cu scopul de a avea acces la anumite fişiere de pe server şi cu scopul de a primi diferite fisiere de la prieteni, având posibilitatea să le descarc fără a avea o limită sau fără să trebuiască să aştept o perioadă mai lungă de timp până obțin acel fișier, putând accesa fișierul direct de pe server. Astfel eliminând acele limitări, eu sunt capabil să primesc fişiere mult mai rapid şi putând încărca/descărca fişiere de dimensiuni maxime determinate de spaţiul de stocare.

* Universitatea Alexandru Ioan Cuza

1.3 Referinţe către astfel de aplicaţii

Online mai există astfel de aplicaţii: Mega, Drive, Wetransfer, toate cu scopul de a transfera şi stoca fisiere, acestea putând fi accesate prin intermediul unui cont pe acel server. Dar nu numai, practic oriunde este necesar transferul de fişiere de la server la client şi invers: în jocuri, pe site-uri sau la update-urile sistemului de operare, toate necesită transfer de fişiere pentru ca clientul să poată profita cât mai mult de serviciile oferite, dar şi pentru ca să le îmbunătăţească.

2 Tehnologiile utilizate

Un TCP concutent, fiecare client fiind conectat la server simultan cu alţi useri şi putând interacţiona cu fişierele de pe server, dar fără să aştepte ca alt user să termine de facut ceva modificare/transfer. Am folosit această tehnologie deoarece am vrut conexiunea dintre server şi client să fie una stabilă, una în care fişierele să fie transferate integral către server fără posibilitatea ca serverul să nu primească fişierul sau să primească unul incomplet (în cazul în care o astfel de situaţie este detectată, clientul trebuie să fie anunţat de acest fapt) şi am mai vrut ca clientul să fie independent de ceilalți useri conectați, comenzile date de acesta executându-se imediat ce cererea ajunge la server.[1]

Pentru stocarea datelor legate de anumiţi utilizatori, am folosit librăria QSQLDatabase (pre-instalată în Qt Creator), librărie capabilă să folosească o serie variată de tipuri de baze de date, dar eu am folosit-o ca una de tip SQLite. Bazele de date de tip SQLite salvează datele într-un fişier local, dar prin intermediul librariei folosite, am posibilitatea de a conecta mai multe thread-uri la acelaşi fişier tip database, prin conexiuni diferite.[5]

Alături de acea librărie avem o alta varietate de librării pre-instalate în Qt Studio, majoritatea fiind legate de interfaţa grafică, dar nu numai, programul oferind posibilitatea de a crea aceasta cu ajutorul unui tool drag and drop, care se foloseşte de o clasă şi de un fişier de tip XML. Unul dintre cele mai importante widget-uri pe care le-am folosit sunt: QStackedWidget, care ne dă posibilitatea de a avea mai multe pagini, QLineEdit, unde am putut scrie datele de autentificare, QTreeWidget în care am afişat fişierele din folderul actual (atat clientului, cat şi serverului), QPushButton de care sunt legate toate interacţiunile interfaţă-client-server.[3]

Mai este şi libraria mbedtls, pe care am folosit-o pentru criptarea şi decriptarea parolelor primite de la client în cadrul secventei de autentificare / inregistrare. Mbed TLS este o implementare a protocoalelor TLS și SSL și a algoritmilor criptografici respectivi și a codului de suport necesar.[7] Secure Sockets Layer (SSL) și Transport Layer Security (TLS), succesorul său, sunt protocoale criptografice care permit comunicații sigure pe Internet. SSL (Secure Sockets Layer) este un acronim care reprezintă un protocol web dezvoltat de compania Netscape pentru a transmite fără risc documente private prin Internet. Pentru a cripta datele, SSL utilizează un sistem criptografic cu două chei: una publică, cunoscută de oricine, și una privată, secretă, cunoscută numai de destinatarul mesajului.[11]

3 Arhitectura aplicaţiei

3.1 Concepte implicate

Un server este un program care furnizează servicii altor aplicații (numite aplicații client), aflate pe același calculator sau pe calculatoare diferite. De obicei, aplicația server așteaptă conexiuni din partea aplicațiilor client. Se mai numește server și un calculator pe care rulează una sau mai multe asemenea aplicații. Deseori soluția pentru mari aplicații cu mulți utilizatori se bazează tocmai pe arhitectura client-server, care constă din ce'l puțin 2 aplicații (și deseori cel puțin 2 computere).[9]
Proces server:
– Ofera servicii in retea
– Accepta cereri de la un proces client
– Realizeaza un anumit serviciu si returneaza rezultatul

Un client este o unitate hardware sau software informatic (calculator sau program), care accesează un serviciu disponibil printr-un server.[2]
Proces client[1]:
– Initializeaza comunicarea cu serverul
– Solicita un serviciu, apoi asteapta raspunsul serverului

Un protocol de comunicații este un set de reguli și norme care permite ca două sau mai multe entități dintr-un sistem de comunicații să comunice între ele prin transmiterea de informație printr-un mediu de orice tip prin variația unei mărimi fizice.[10]

Socket: tunel de conectare intre 2 procese străine (în cazul nostru client şi server), atât din aceeaşi maşină, cât şi din maşini diferite.
Facilitate generală, independentă de arhitectura hardware, de protocol şi de tipul de transmisiune a datelor, pentru comunicare între procese aflate pe maşini diferite, în reţea.[1]
Oferă suport pentru familii multiple de protocoale
– Protocolul domeniului UNIX – folosit pentru comunicaţii locale
– Protocolul domeniului Internet folosind TCP/IP
– Altele: XNS Xerox,...
Abstracţiune a unui punct terminal (end-point) la nivelul transport

Firele de execuţie [1] (threads) sunt numite si lightweight processes (LWP). Pot fi văzute ca un program aflat în executie fără spatiu de adresă proprie.
Pthreads (POSIX Threads) standard ce defineşte un API pentru crearea şi manipularea firelor de execuţie.

3-way handshake: modul cum se stabileşte conexiunea într-un protocol de tip TCP. Aceasta, după cum sugerează şi numele, are 3 etape:
- 1: clientul doreşte să stabilească o conexiune cu serverul, astfel trimiţând un segment care îl informează pe server că clientul ar dori să înceapă o conexiune şi cu ce descriptor.
- 2: serverul răspunde şi confirmă crearea conexiunii dintre cei 2, trimiţând un segment la client care să-l anunţe de acest fapt.
- 3: clientul confirmă răspunsul de la server şi amândoi realizează o conexiune prin care eventual vor transfera date.

Criptarea este procesul de mascare a informației pentru a o face ilizibilă fără cunoștințe speciale. Criptarea poate fi folosită pentru a asigura discreția și/sau intimitatea, dar și alte tehnici sunt necesare pentru a face comunicațiile sigure.

Baza de date: mod de stocare a datelor sub format tabelar şi care pot fi uşor accesate prin intermediul interogărilor, dar şi vizual dacă sunt folosite anumite programe pentru asta.
O bază de date, uneori numită și bancă de date (abreviat BD), reprezintă o modalitate de stocare a unor informații și date pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora[12].

3.2 Diagramă

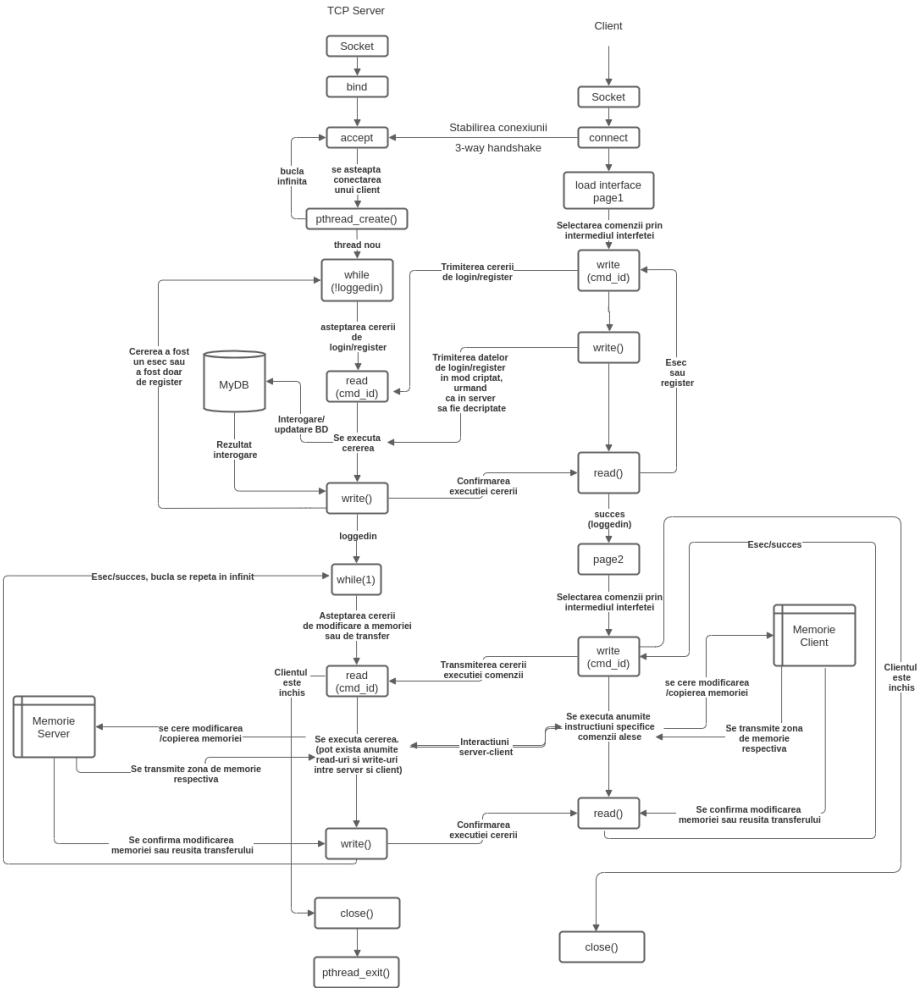

Fig. 1. Diagrama aplicației MyFileTransferProtocol

4 Detalii de implementare

4.1 Cod relevant particular proiectului

Server

```
 struct sockaddr_in server;  // structura folosita de server
 struct sockaddr_in from;
 int sd;         //descriptorul de socket
/* crearea unui socket */
if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror ("[server]Eroare la socket().\n");
    return errno;
}
 /* pregatirea structurilor de date */
 bzero (&server, sizeof (server));
 bzero (&from, sizeof (from));

 /* umplem -structura folosita de server */
 /* stabilirea familiei de socket-uri */
 server.sin_family = AF_INET;
 /* acceptam orice adresa */
 server.sin_addr.s_addr = htonl (INADDR_ANY);
 /* utilizam un port utilizator */
 server.sin_port = htons (PORT);

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

 /* punem serverul sa asculte daca vin clienti sa se conecteze */
 if (listen (sd, 1) == -1)
 {
     perror ("[server]Eroare la listen().\n");
     return errno;
 }
  pthread_t t_id;//pregatim o variabila
 while(1)
 {
     int* client=new int;
     unsigned int length = sizeof (from);
     printf ("[server]Asteptam la portul %d...\n",PORT);
     fflush (stdout);
     //acceptam un client sa se conecteze la server
```

```
    *client = accept(sd, (struct sockaddr *) &from, &length);
    if (*client < 0)
        {
        perror ("[server]Eroare la accept().\n");
        continue;
        }
    //cream un thread pentru clientul conectat
    if(pthread_create(&t_id,NULL,thread_func,client)){
       perror ("[thread]Eroare la pthread_create()");
       continue;
    }
}
void* thread_func(void* client_sock)
{
    //facem castare de la void* la int pentru client
    int client= (*(int*)client_sock);
    char name[30];
    char password[30];
    bool i=false, loggedin=false;
    int cmd_ID;
    //deschidem baza de date de fiecare data
    //un user se conecteaza
    DbManager dbm("MyDB");
    //incrementam id-ul conexiunii
    dbm.connections++;
    //verificam daca user-ul este logat sau nu
    while(!loggedin){
        //asteptam o comanda
        if(read(client, &cmd_ID, 4)<=0){
            //clientul s-a deconectat
            qDebug("Client disconnected");
            close(client);
            pthread_exit(NULL);
            //inchidem conexiunea si thread-ul
        }
        //verificam ce comanda este
      switch(cmd_ID)
      {
          case 1: //login
              //citim datele de la client
              read(client,&name, 30);
              read(client,&password, 30);
              //apelam functia de verificare
              //a userului in baza de date
```

```
            if(dbm.CheckUser(name,password))
                i=true;
            //ii trimitem clientului raspunsul
            write(client, &i, 1);
            //resetam comanda
            cmd_ID=0;
            //user-ul este logat
            loggedin=i;
            break;
        case 2://register
            //citim datele de la client
            read(client,&name, 30);
            read(client,&password, 30);
            //apelam functia de verificare
            //a userului in baza de date
            if(dbm.NewUser(name,password))
                i=true;
            //ii trimitem clientului raspunsul
            write(client, &i, 1);
            //resetam comanda
            cmd_ID=0;
            //asteptam sa le logheze
            break;
        default: //comanda care nu este in lista
        pthread_exit(NULL);
    }
}
#include <QString>
#include <QSqlDatabase>
class DbManager
{
public:
    static int connections;
    DbManager(const char* path);
    bool NewUser(const char* name, const char* password);
    bool CheckUser( const char* name, const char* password);
private:
    QSqlDatabase m_db;
};
```

```
DbManager::DbManager( const char* path)
{
   //crearea unei baze de date de tip SQLite
   m_db = QSqlDatabase::addDatabase("QSQLITE",QString::number(connections));
   //legarea fisierului de baza de date creata
   m_db.setDatabaseName(path);
   //deschiderea bazei de date
   if (!m_db.open())
   {
      qDebug() << "Error: connection with database failed";
   }
   else
   {
      qDebug() << "Database: connection ok";
   }
}
bool DbManager::NewUser(const char* name, const char* password)
{
    //bool cu rol de verificare
    //daca este deja inregistrat acel user
    bool already = false;
    //cream o variabila de tip query
    //pentru baza de date
    QSqlQuery query(m_db);
    //"pregatim" ce interogare
    //va trebui executata
    query.prepare("SELECT name FROM users WHERE name = :name");
    //punem in ":name" valoarea lui name
    query.bindValue(":name", name);
    //executam interogarea
    if(query.exec())
    {
        //trecem la primul element
        //rezultat din interogare
        if(query.next())
        {
            //daca exista user-ul
            already = true;
        }
    }
    //verificam daca exista user-ul sau nu
    if(!already){
        //"pregatim" ce interogare
        //va trebui executata
```

```
        query.prepare("INSERT INTO users VALUES (:name, :password, 0)");
        //punem in ":name" valoarea lui name
        query.bindValue(":name", name);
        //aici ar trebui prima data sa decriptam parola
        //si dupa sa o adaugam in interogare
        //punem in ":password" valoarea lui password
        query.bindValue(":password", password);
        //executam interogarea
        if(query.exec())
            //interogarea a reusit adaugarea unui user nou
            return true;
        else
            //interogarea nu a reusit
            qDebug() << "NewUser error:"
                     << query.lastError();
        return false;
    }
    else{
        //user-ul deja exista
        printf("User already exists");
        return false;
    }
}
bool DbManager::CheckUser(const char* name, const char* password)
{
    //cream o variabila de tip query
    //pentru baza de date
    QSqlQuery query(m_db);
    //"pregatim" ce interogare
    //va trebui executata
    query.prepare("SELECT * FROM users WHERE name = :name "
                  "AND password = :password AND blacklisted = 0");
    //punem in ":name" valoarea lui name
    query.bindValue(":name", name);
    //aici ar trebui prima data sa decriptam parola
    //si dupa sa o adaugam in interogare
    query.bindValue(":password", password);
    //executam interogarea
    if(query.exec())
        //trecem la primul element
        //rezultat din interogare
        if(query.next())
```

```
            //daca exista user-ul si toate
            //detalii de conectare sunt corecte
            return true;
    //nu exista user-ul sau a gresit parola
    return false;
}
```

Client

```
#include "client_interface.h"
#include <QApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    //cream o aplicatie in Qt cu argumentele date
    QApplication a(argc, argv);
    //cream interfata necesara
    //client_interface este o clasa
    //pe care o putem edita
    Client_interface w;
    //afisam pe ecran interfata
    w.show();
    //asteptam ca aplicatia sa se inchida
    return a.exec();
}
bool conn(int* sd){
    // descriptorul de socket
    struct sockaddr_in server;  // structura folosita pentru conectare
    /* cream socketul */
    if ((*sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
      {
        perror ("Eroare la socket().\n");
      }
    /* umplem structura folosita pentru realizarea conexiunii cu serverul */
    /* familia socket-ului */
    server.sin_family = AF_INET;

    /* adresa IP a serverului */
    server.sin_addr.s_addr = inet_addr(IP);
    /* portul de conectare */
    server.sin_port = htons (port);
   /* ne conectam la server */
   if (connect (*sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) == -1)
     {
```

```
        perror ("[client]Eroare la connect().\n");
        return false;
      }
    qDebug() << "connect ok";
    return true;
}
bool login(int sd, QString name, QString password){
    bool i;
    int x=1;
    //transmitem codul comenzii la server
    write(sd,&x,4);
    //convertim textul de la QString la const char*
    std::string text_nume = (name.toStdString());
    std::string text_parola = (password.toStdString());
    const char* nume = text_nume.c_str();
    const char* parola = text_parola.c_str();
    //transmitem datele la server
    write(sd, (nume), 30);
    write(sd, (parola), 30);
    //primim raspuns de la server
    read(sd, &i, 1);
    //transmitem raspunsul catre functia principala
    return i;
}
bool reg(int sd,QString name, QString password){
    bool i;
    int x=2;
    //trimitem codul comenzii la server
    write(sd,&x,4);
    //convertim textul de la QString la const char*
    std::string text_nume = (name.toStdString());
    std::string text_parola = (password.toStdString());
    const char* nume = text_nume.c_str();
    const char* parola = text_parola.c_str();
    //transmitem datele la server
    write(sd, (nume), 30);
    write(sd, (parola), 30);
    //primim raspuns de la server
    read(sd, &i, 1);
    //transmitem raspunsul catre functia principala
    return i;
}
```

```
void Client_interface::on_pushButton_clicked()
{//butonul de login este apasat
    //iau datele din casturele QLineEdit
    QString user = ui->username->text();
    QString password = ui->password->text();
    //execut comanda de login
    if(!login(sd,user, password))
        //daca nu reuseste sa se logheze
        perror("[login]");
    else
    {
        //pune numele fisierelor in treewidget
        displayTree(user);
        //schimbam pagina
        ui->stackedWidget->setCurrentWidget(ui->page2);
    }
}

void Client_interface::on_pushButton_2_clicked()
{//butonul de register este apasat
    //iau datele din casturele QLineEdit
    QString user = ui->username->text();
    QString password = ui->password->text();
    //execut comanda de register
    if(!reg(sd, user, password))
        //daca nu reuseste sa se inregistreze
        perror("[register]");
}
```

4.2 Scenarii

Scenarii de reușită :

- Clientul se înregistrează şi se loghează cu succes fără probleme, serverul raspunzând corespunzător şi făcând acele modificări în baza de date. Acest scenariu este cel mai probabil dacă acel client foloseşte un username unic, neexistent în baza mea de date.

- Clientul face transfer de fişiere între el şi server fără probleme, toate fişierele ajungând îm locul potrivit, corespunzător comenzii date de client.

Scenarii de eșec :

- Clientul doreşte să trimită un fişier la server, dar serverul nu are destul spaţiu disponibil. Acest scenariu se poate rezolva, prin verificarea mărimii fişierului înainte de a trimite şi compararea acestuia cu spaţiul liber rămas pe server, iar dacă scenariul se întampla, trimitem un mesaj la client referitor la acest fapt.

- Clientul este deja înregistrat, dar acesta apasă din nou pe butonul register. Pentru acest scenariu putem verifica primă dată dacă acel user este deja înregistrat în baza de date şi dacă există, atunci trimitem la client un mesaj şi îi spunem că deja există acel utilizator. Dacă facem în acest fel, eliminăm posibilitatea înregistrărilor multiple cu acelaşi username.

5 Concluzii

Sistemul prezentat anterior, după părerea mea, este unul acceptabil, dar poate fi îmbunătățit şi astfel aş putea adăuga, cu scopul de a crea un sistem mai eficient şi lipsit de probleme, un set mai mare de error checking-uri pentru a acoperi toate posibilităţile, dar şi să schimbăm modul de structurare a fisierelor astfel încât fiecare utilizator să aibă propria zonă în memorie, astfel eliminând posibilitatea ca 2 sau mai mulţi clienţi să dorească acces la un fişier în acelaşi timp. Pe lângă acestea, aş mai putea lucra la modul de transmitere a fişierelor între server şi client, facând un sistem care să fie mai stabil, dar şi la modul de conectare între aceştia, facând un sistem mai optim capabil să menţină un număr mai mare de clienţi în acelaşi timp. Toate acestea sunt cu scopul de a face aplicaţia mai eficientă.

6 Bibliografie

References

1. Cursurile de RC predate - `https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php`
2. Client - `https://ro.wikipedia.org/wiki/Client_(informatic%C4%83)`
3. Qt 5.15 documentation - `https://doc.qt.io/qt-5/`
4. How to use QTreeWidget - `https://www.youtube.com/watch?v=81tYrNw18Ak`
5. Sqlite in Qt - `https://katecpp.github.io/sqlite-with-qt/`
6. SQLite documentation - `https://www.sqlite.org/cintro.html`
7. Mbed TLS - `https://en.wikipedia.org/wiki/Mbed_TLS`
8. Mbed TLS documentation - `https://tls.mbed.org/api/`
9. Server - `https://ro.wikipedia.org/wiki/Server`
10. Protocol - `https://ro.wikipedia.org/wiki/Protocol_de_comunica%C8%9Bii`
11. TLS - `https://ro.wikipedia.org/wiki/Transport_Layer_Security#TLS_1.1`
12. Baze de date - `https://ro.wikipedia.org/wiki/Baz%C4%83_de_date`