

My File Transfer Protocol

Mihai Marian-Sebastian

Rețele de calculatoare - Ianuarie 2019

1 Introducere

Proiectul consta in implementarea unei aplicatii client-server ce permite transferul de fisiere intre clienti si server. Serverul va pune la dispozitia clientilor un numar minim de comenzi ce permit autentificarea, operarea cu directoare si cu fisiere.

De asemenea, trebuie implementat un mecanism de autorizare (whitelist/blacklist) pentru conturile utilizatorilor si un mecanism de transmitere securizata a parolei la autentificare.

Acesta aplicatie poate fi folosita pentru a stoca anumite fisiere temporar pe un server pentru a nu le pierde.

2 Tehnologii utilizate

Am utilizat protocolul TCP deoarece:

- asigura retransmiterea pachetelor (in cazul in care acestea se “pierd pe drum”);
- ordoneaza pachetele ajunse la destinatie (acestea pot ajunge diferit fata de cum au fost trimise);
- stabileste o conexiune intre client si server.

De asemenea, am folosit **thread-uri** pentru a asigura concurenta serverului TCP/IP si **socket-uri**.

3 Arhitectura aplicatiei

Figura 1 (de mai jos)

In momentul in care se conecteaza un client, acesta trebuie sa se autentifice pentru a putea trimite un fisier catre server. La autentificare, sunt transmise catre server utilizatorul si parola (criptata) pentru a se verifica daca contul este unul valid (nu exista in lista neagra si este corect).

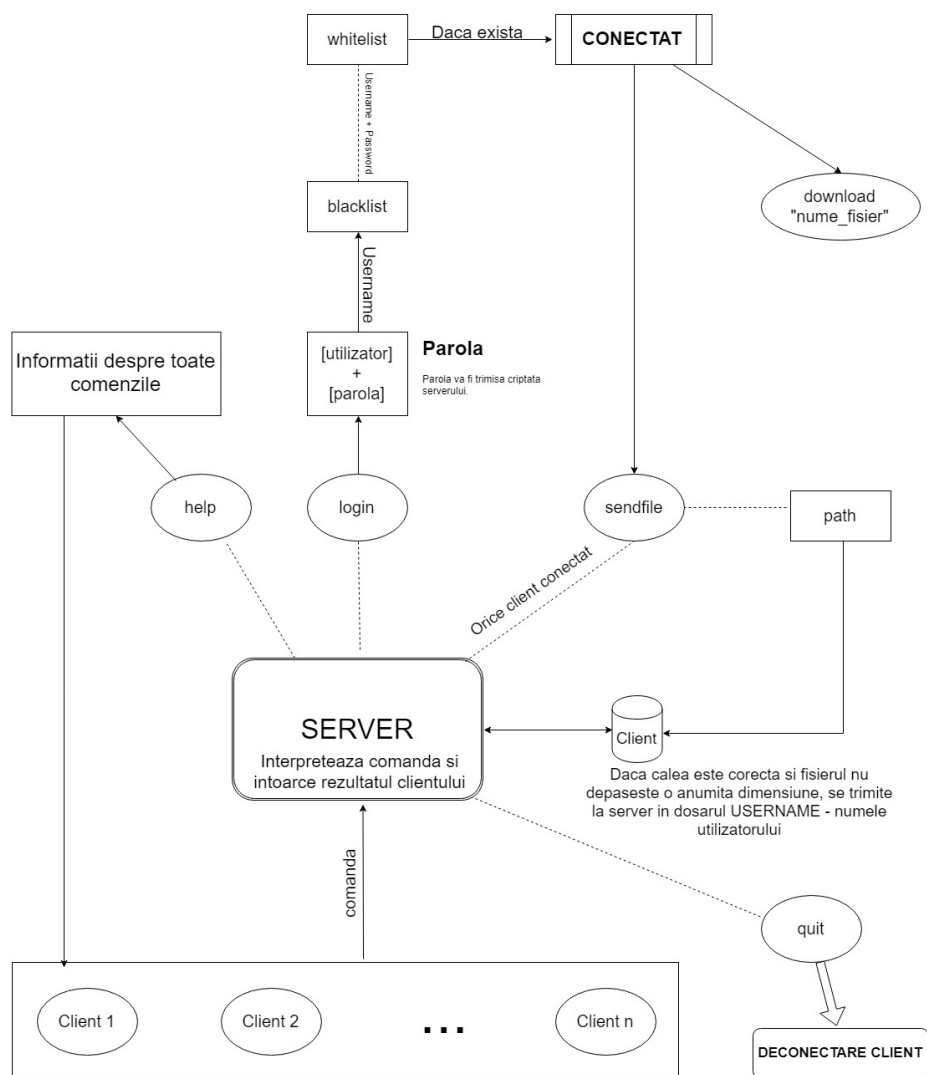


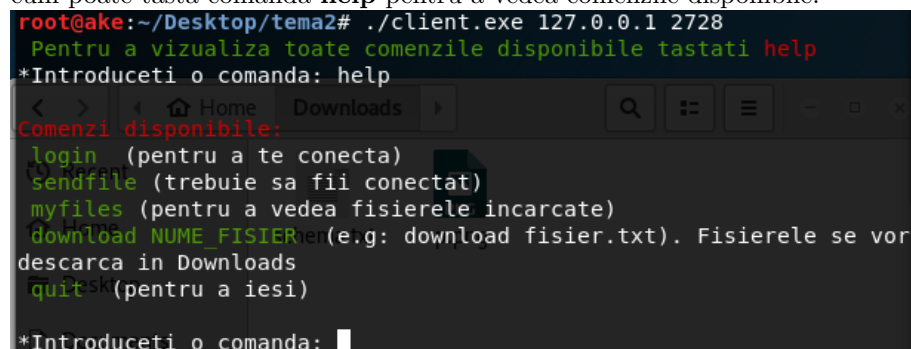
Figure 1: Schema aplicatiei server-client.

Un cont este corect daca acesta exista in **whitelist**, iar parola decriptata in server este aceeași cu parola decriptata din whitelist.

Cand se trimite un fisier cu comanda **"sendfile"** se specifica calea fisierului, iar locul unde va ajunge va fi in directorul **"Resurse"** din folderul aplicatiei, intr-un folder numit exact ca numele de utilizator.

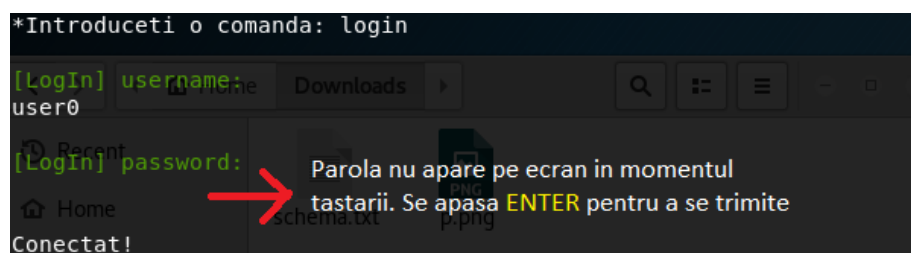
4 Detalii de implementare

In momentul conectarii unui client la server, acestuia ii apare un mesaj precum poate tasta comanda **help** pentru a vedea comenzile disponibile!



```
root@ake:~/Desktop/tema2# ./client.exe 127.0.0.1 2728
Pentru a vizualiza toate comenzile disponibile tastati help
*Introduceti o comanda: help
Comenzi disponibile:
login (pentru a te conecta)
sendfile (trebuie sa fii conectat)
myfiles (pentru a vedea fisierele incarcate)
download NUME_FISIER (e.g: download fisier.txt). Fisierele se vor
descarca in Downloads
quit (pentru a iesi)
*Introduceti o comanda: 
```

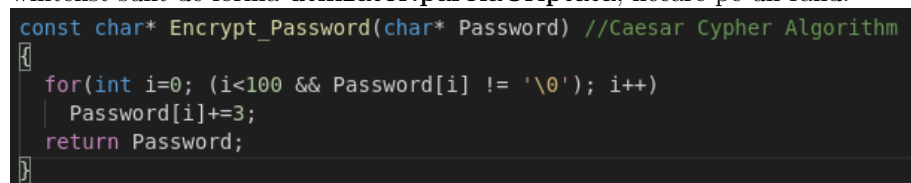
La introducerea comenzii **login** clientul va introduce numele de utilizator si parola (**care nu va fi afisata pe ecran**), urmand sa primeasca un mesaj de la server: **Conectat**, **Cont invalid**, **Cont blocat**, **Parola incorecta**. Clientul primeste mesajul **cont blocat** daca numele de utilizator se afla in fisierul **blacklist.txt**.



```
*Introduceti o comanda: login
[LogIn] username: user0
[LogIn] password:
Conectat!
```

Parola nu apare pe ecran in momentul tastarii. Se apasa ENTER pentru a se trimite

Pe partea de back-end, parola este trimisa la server criptata. Serverul o decripteaza si o compara cu parola DECRYPTATA din whitelist. Conturile in whitelist sunt de forma **utilizator:parolaCriptata**, fiecare pe un rand.



```
const char* Encrypt_Password(char* Password) //Caesar Cypher Algorithm
{
    for(int i=0; (i<100 && Password[i] != '\0'); i++)
        Password[i]+=3;
    return Password;
}
```

Criptarea parolei consta in adaugarea cifrei 3 la fiecare cod ASCII corespunzator

fiecarui caracter din parola.

Cand comanda trimisa de client este **"myfiles"** acestuia ii apar toate fisierele incarcate, de pe acel cont, pe server.

```
*Introduceti o comanda: myfiles

Fisiere disponibile:
p.c - 0.3 Kbytes
schema.txt - 1.7 Kbytes
protocol.c - 6.5 Kbytes
cod.c - 8.5 Kbytes
p.png - 575.5 Kbytes
aaa.zip - 4.5 Kbytes
```

Cand comanda trimisa de client este **"sendfile"** acesta primeste raspuns de la **server** ca sa introduca calea absoluta, iar daca aceasta este corecta, se va trimite catre server.

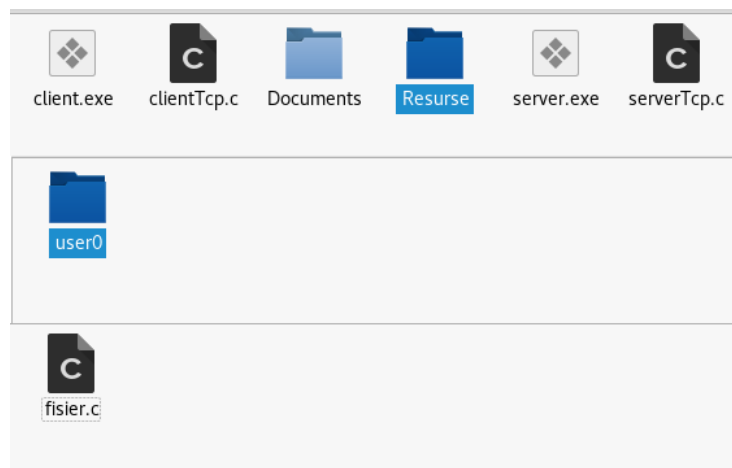
Aceasta comanda poate fi utilizata doar dupa conectare!

```
*Introduceti o comanda: sendfile

[Send file] Introdu calea absoluta catre fisier.
[Send file] E.g : /root/Desktop/Folder1/fisier.txt
/root/Desktop/fisier.c

[Info File]: Calea fisierului este corecta!
[File Size]: 6674 bytes
[Info File]: Fisierul a fost trimis la server (6674 bytes) !
[Server] Comanda terminata!
```

Pe partea de back-end, se creeaza in folderul **Resurse** un folder numit exact ca numele de utilizator (in caz ca nu exista deja) unde se copiaza fisierul respectiv.



Cand comanda trimisa de client este ”download NumeFisier” acestuia i se descarca de pe server, in folderul **Downloads** fisierul respectiv.

```
*Introduceti o comanda: download p.png
[Download File] Completed: 575.5 kb
*Introduceti o comanda: download schema.txt
[Download File] Completed: 1.7 kb
```

Pe partea de **server** o functie principala este cea care interpreteaza fiecare comanda primita de la **client**.

Dupa ce se identifica comanda primita, se ruleaza functia specifica comenzii si apoi se intoarce un raspuns clientului.

```
const char* InterpretareComanda(void *arg)
{
    struct thData tdL;
    tdL = *((struct thData*)arg);

    char comanda[buffer]=" "; //comanda primita de la client
    char msgrasp[buffer]=" "; //mesaj de raspuns pentru client

    if(Lista_Useri[tdL.idThread].Downloading == 1) //Acum e momentul sa se trimita fisierul ca
    {
        DownloadFile((struct thData*)arg);
        Lista_Useri[tdL.idThread].Downloading = 0;
        bzero(msgrasp,buffer);
        strcpy(msgrasp, "[Server] Comanda terminata!");
        printf ("[Thread %d] Trimitem raspuns clientului: %s\n", tdL.idThread, msgrasp);
        if (write (tdL.cl, msgrasp, strlen(msgrasp)+1) < 0)
        {
            perror ("[Server] Eroare la write() catre client.\n");
            return "exit";
        }
    }
    else
    {
        bzero(comanda,buffer);
        if ((read (tdL.cl, comanda, buffer) < 0))
        {
            perror ("Eroare la read() de la client.\n");
            return "exit";
        }
        // interpretam comanda si apelam functia necesara fiecarei comenzi
        printf ("-----\n");
        printf ("[Thread %d] Comanda receptionata: %s\n", tdL.idThread, comanda);
        if(strcmp("help",comanda) == 0)
        {
            strcpy(msgrasp, Help());
            printf ("[Thread %d] Trimitem raspuns clientului: %s\n", tdL.idThread, msgrasp);
            if (write (tdL.cl, msgrasp, strlen(msgrasp)+1) < 0)
            {

```

De exemplu, functia **SendFileToClient(void *arg, char* filename)** care are ca parametri structura de la thread-ul specific clientului si numele fisierului de trimis, scrie intr-un **buffer** continutul binar al fisierului si apoi il trimite catre client, acesta urmand sa apeleze o functie care ii copiaza acel continut intr-un fisier cu acelasi nume, si il salveaza.

Practic, s-a descarcat fisierul.

```

void* SendFileToClient(void *arg, char* fileName)
{
    struct thData tdL;
    tdL = *((struct thData*)arg);

    char path[256];
    if(getcwd(path, sizeof(path)) == NULL)
        perror("[Server] getcwd() error.\n");
    else
    {
        strcat(path, "/Resurse/");
        strcat(path, Lista_Useri[tdL.idThread].Username);
        strcat(path, "/");
        strcat(path, fileName);
    }

    unsigned long fsize=0;
    FILE *fs = fopen(path, "r");
    if(fs == NULL)
    {
        perror("File open error");
    }
    fseek(fs, 0L, 2);
    fsize = ftell(fs);
    fclose(fs);

    FILE *fp = fopen(path, "rb");
    if(fp == NULL)
    {
        perror("File open error");
    }
    else
    {
        char* file;
        file = (char*) malloc(fsize);

        long nread = fread(file, 1, fsize, fp);

        if(nread > 0)
        {
            if (write (tdL.cl, file, fsize) <= 0)
            {
                perror ("[Server]Eroare la write() spre client.\n");
            }
            printf("[Thread %d] [Download File]: Fisierul a fost trimis la client (%ld bytes) ! \n", tdL.idThread, nread);

            fflush(stdout);
        }
        else
        {
            printf("[Thread %d] [Download File]: Fisierul NU a putut fi trimis la client ! \n", tdL.idThread);
            fflush(stdout);
        }
    }
}

```

Fiecare client conectat are in **server** anumite informatii stocate sub forma unei structuri.

```

struct User{
    int Conectat;
    char* Username;
    char* Password;
    int Downloading;
    char* Path;
}Lista_Useri[100];

```

Conectat devine 1 in momentul in care un client se conecteaza la server, si 0 cand acesta trimite comanda "quit" (se deconecteaza).

In **Username** se retine numele de utilizator al clientului (bineinteles, dupa

ce acesta se conecteaza).

De asemenea, in **Password** parola acestuia.

Variabila **Downloading** este necesara pentru a sti ca urmatoarea comanda dupa ce se trimite calea fisierului de catre client, este sa se incarce acel fisier. Aceasta devine **1** cand se trimite o cale valida a unui fisier, si **0** cand s-a terminat de incarcat fisierul respectiv.

In **Path** se retine calea catre fisier, trimisa de client.

Fiecare client se identifica in lista prin id-ul de client.

5 Concluzii

In cele din urma, cred ca acest protocol este util pentru cine vrea sa stocheze pe un server la distanta anumite fisiere pentru impiedicarea pierderii acestora, si sa le poata descarca ulterior, cand are nevoie.

Consider ca acest proiect poate sa aiba o versiune mai buna daca:

- va avea o interfata grafica, astfel ar putea fi mult mai accesibila clientilor;
- implementarea altui algoritm mai complex pentru criptarea parolei;
- impiedicarea incercarii repetate de conectare la server a unui client care introduce parola gresit de cateva ori;
- posibilitatea stergerii unui fisier din spatiul de stocare de pe server;
- adaugarea unei limite de spatiu pentru fiecare cont.

6 Bibliografie

<https://sites.google.com/view/fii-rc>

<https://www.draw.io/>

<https://www.overleaf.com>