MyFileTransferProtocol (B)

Lavric Adrian-Gabriel[2A3]

Facultatea de Informatica, Universitatea "Alexandru Ioan-Cuza" Iasi
adrian.lavric33@gmail.com

Abstract. Acest raport prezinta o aplicatie client/server in C, implementarea acesteia si tehnologiile folosite. Raportul este impartit in 6 sectiuni, prima descrie viziunea generala a proiectului, sectiunea "Tehnologii Aplicate" contine protocolul de comunicare, iar "Structura Aplicatiei" ofera detalii despre conceptele folosite in modelare si o diagrama care le ilustreaza. Capitolul "Aspecte de implementare" contine sectiuni de cod relevante si protocolul la nivelul aplicatie, in "Concluzii" se gasesc potentiale imbunatatiri ale aplicatiei, iar in ultima sectiune referinte bibliografice.

Keywords: TCP · Concurent · Transfer

1 Introducere

Proiectul MyFileTransferProtocol este o aplicatie client/server in C si sistemul de operare Linux ce permite transferul de fisiere intre clienti si server(serverul fiind concurent, se pot conecta mai multi clienti la el in acelasi timp). Serverul pune la dispozitia clientilor comenzi ce permit autentificarea, operarea cu directoare si fisiere. Aplicatia va folosi un mecanism de autorizare de tipul whitelist pentru conturile utilizatorilor si un mecanism de transmitere securizata a parolei la autentificare.

2 Tehnologii Aplicate

Aplicatia foloseste modelul client/server de tip TCP(Transmission Control Protocol), stiind ca scopul protocolului TCP este acela de a controla transferul de date astfel incat sa fie de incredere si de a asigura fiabilitate in timpul transmisiei(prin mecanisme precum controlul congestionarii si controlul fluxului), doua aspecte foarte importante in cazul unui transfer de fisiere. La comunicarea dintre client si server, datele sunt trimise organizate in pachete, toate pachetele ajung la destinatie si niciunul nu este pierdut, iar apoi sunt reasamblate in ordine(deci datele nu sunt pierdute, modificate sau corupte, cum se intampla in cazul folosirii UDP, protocol ce nu se potriveste pentru acest tip de aplicatie). De asemenea, se stabileste o conexiune intre server si client inainte de a se transmite datele(connection-oriented), iar folosirea TCP este recomandata in general in cazul transferului de fisiere in masa, oferind o comunicare eficienta si securizata.

3 Structura Aplicației

3.1 Concepte folosite in modelare

Server Serverul foloseste TCP care implica stabilirea unei conexiuni pentru a comunica cu clientii si este concurent, poate gestiona conexiuni de la mai multi clienti in acelasi timp, creand cate un proces copil pentru a deservi fiecare client prin functia fork. Initial se creaza un socket de tip TCP care utilizeaza adrese IPv4(prin optiunile `SOCK_STREAM` si `AF_INET`), apoi se pregatesc structurile de date struct `sockaddr_in` folosind bzero. Serverul accepta orice adresa, familia de socket-uri este `AF_INET`, iar portul utilizator este definit 2024. Socket-ul reutilizeaza adresa si portul prin functia setsockopt cu optiunea `SO_REUSEADDR`, iar apoi este legat de adresa si port prin functia bind. Serverul asculta daca vin clienti sa se conecteze prin listen si accepta conexiuni de la clienti prin functia accept. Noul proces copil creat prin fork gestioneaza clientul, primind comenzi si dand raspunsuri prin read si write, iar in cazul in care clientul doreste sa intrerupa conexiunea, se apeleaza close cu parametrul descriptorul de socket client. In cazul oricarei erori la stabilirea conexiunii sau la comunicarea cu clientul se afiseaza un mesaj corespunzator prin perror.

Client Clientul primeste in linia de comanda adresa si portul, converteste portul la int prim functia atoi, apoi creeaza un socket cu aceleasi optiuni folosite de server, iar structura folosita pentru realizarea conexiunii cu serverul struct `sockaddr_in` foloseste adresa IP si portul serverului. Conexiunea cu serverul se realizeaza prin functia connect cu parametrii socket-ul, structura `sockaddr_in` si dimensiunea structurii. Dupa conectare, clientul trimite comenzi si primeste raspunsuri prin write si read, iar cand se introducere comanda de inchidere a conexiunii se apeleaza close cu parametrul descriptorul de socket. In cazul oricarei erori la stabilirea conexiunii sau la comunicarii cu serverul se afiseaza un mesaj corespunzator prin perror.

3.2 Diagrama aplicatiei

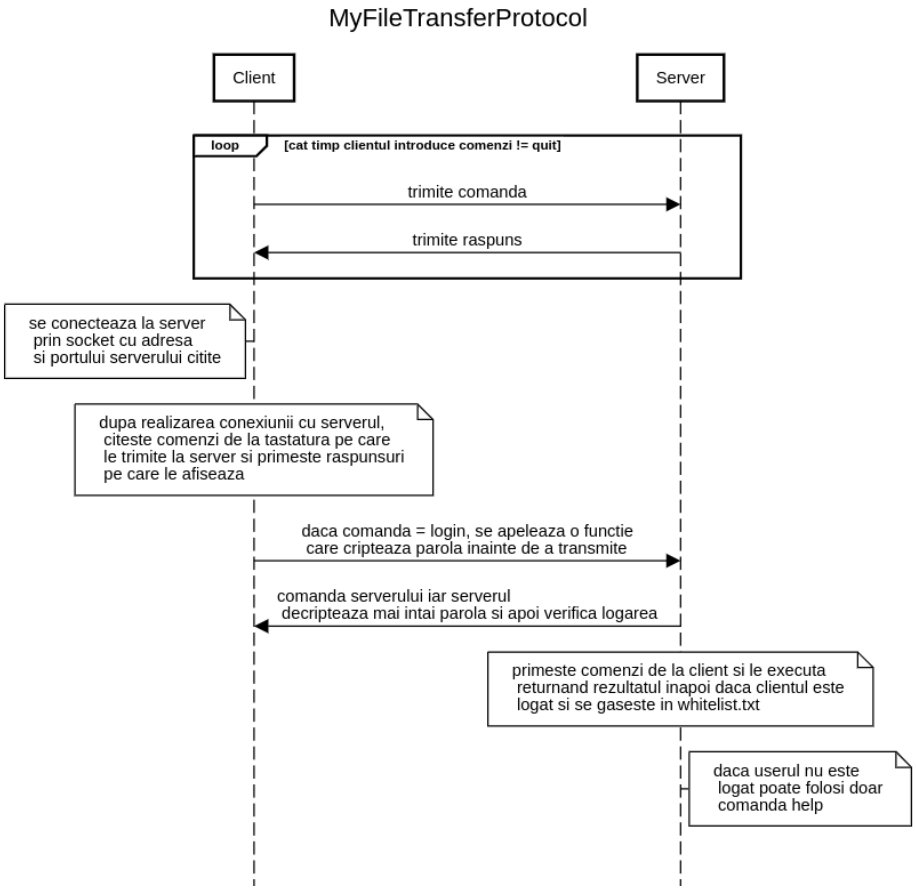

Fig. 1. Structura Aplicatiei

4 Aspecte de Implementare

4.1 Sectiuni de cod

Aplicatia contine 2 fisiere C, server.c si client.c, si un fisier text whitelist.txt. Pentru a o folosi, se vor introduce urmatoarele comenzi in terminal: gcc server.c -o server pentru compilare server, gcc client.c -o client pentru compilare client si ./server pentru rulare server, ./client (adresa) (port) pentru rulare client.(portul si adresa sunt cele ale serverului). Urmatoarea sectiune de cod din server.c prezinta implementare concurentei folosind functia fork:

```
// servim in mod concurent clientii...
    while(1) {

        int pid;
        if((pid = fork()) == -1) {

            perror("[server]Eroare la fork.\n");
            return errno;

        } else if(pid > 0) {

            // parinte
            close(client);
            while(waitpid(-1,NULL,WNOHANG));
            continue;

        } else if(pid == 0) {

            // comunicarea cu clientul si gestionarea comenzilor
            // se face in procesul copil
            close(sd);
            int logat = 1;

            while(1) {

             //prelucreare comenzi

            }
       }
  }
```

Urmatoarele sectiuni de cod din server.c si client.c descriu implementarea protocolului de comunicare folosind functiile socket, bind, accept, read si write:

```
    //server.c

    // crearea socket-ului prin care se realizeaza
    // conexiunea cu clientii
    if((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {

        perror("[server]Eroare la creare 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, 10) == -1)
{
 perror("[server]Eroare la listen.\n");
 return errno;
}
while(1)
{
    int client;
    int length = sizeof(from);
    // acceptam un client (stare blocanta pana la
    // realizarea conexiunii)
    client = accept(sd, (struct sockaddr *)&from, &length);

}


//client.c

//trimiterea comenzii la server
if(write(sd, comanda, 100) <= 0) {

    perror("[client]Eroare la write spre server.\n");
    return errno;
```

```
}

// citirea raspunsului dat de server (apel blocant pana
// cand serverul raspunde)
if(read(sd, &lungime_raspuns, sizeof(int)) <= 0) {

    perror("[client]Eroare la read de la server.\n");
    return errno;

}

char *raspuns = (char*)malloc(lungime_raspuns + 1);

if(read(sd, raspuns, lungime_raspuns) <= 0) {

    perror("[client]Eroare la read de la server.\n");
    return errno;

}

raspuns[lungime_raspuns] = '\0';

// afisam raspunsul primit
printf("[client]Raspuns : %s\n", raspuns);
```

Criptare si decriptare parola :

```
if(nr_spatii == 1) {

    for(int j = i + 1; j < lungime_comanda; j++) {

        comanda[j] = comanda[j] + 3;

    }
    break;

}

if(nr_spatii == 1 && comanda[i] == ' ') {

    for(int j = i + 1; j < lungime_comanda; j++) {

        comanda[j] = comanda[j] - 3;
```

```
    }

}
```

Verificare validitate logare in fisierul whitelist.txt :

```
char *user_parola = comanda + 6;
int ok = 0;

FILE* whitelist = fopen("whitelist.txt", "r");
if(whitelist) {

    char info[100];
    while(fgets(info, sizeof(info), whitelist)) {

        int lungime_info = strlen(info);
        info[lungime_info - 1] = '\0';

        if(strcmp(user_parola, info) == 0) {

            ok = 1;
            char *user_curent = strtok(user_parola, " ");
            strcpy(user, user_curent);
            logat = 1;
            printf("[server]Userul %s s-a logat.\n", user);
            break;

        }

    }
    fclose(whitelist);

}
```

Descarcarea unui fisier de pe server :

```
if(nu_exista_local) {

    FILE* fisier_download2 = fopen(cale_fisier_download, "r");
    fseek(fisier_download2, 0, SEEK_END);
    dimensiune_fisier2 = ftell(fisier_download2);
    fseek(fisier_download2, 0, SEEK_SET);
```

```
        char* continut2 = (char*)malloc(dimensiune_fisier2 + 1);
        fread(continut2, 1, dimensiune_fisier2, fisier_download2);
        continut2[dimensiune_fisier2] = '\0';
        fclose(fisier_download2);
        write(client, &dimensiune_fisier2, sizeof(long));
        write(client, continut2, dimensiune_fisier2);
        free(continut2);
        continue;

    }
```

Functie recursiva de stergere director :

```
    if(director) {

        while((director_curent = readdir(director)) != NULL) {

            if(strcmp(director_curent->d_name, ".") != 0
            && strcmp(director_curent->d_name, "..")!= 0) {

                char cale_director_curent[500];
                strcpy(cale_director_curent, cale);
                strcat(cale_director_curent, "/");
                strcat(cale_director_curent, director_curent->d_name);
                if(stat(cale_director_curent, &info_folder) == 0) {

                    if(S_ISDIR(info_folder.st_mode)) {

                        int stergere_subdirector =
                        sterge_director(cale_director_curent);
                        if(!stergere_subdirector) {

                            return 0;

                        }

                    } else {

                        if(remove(cale_director_curent) != 0) {

                            return 0;

                        }
```

```
                }

            } else {

                return 0;

            }

        }

    }
    closedir(director);

} else {

    return 0;

}
```

4.2 Protocolul la nivel aplicatie

Dupa compilare si rularea programelor server.c si client.c, clientul se conecteaza la server, citeste comenzi de la tastatura, le transmite serverului, care executa comanda si ii trimite inapoi un raspuns. Pentru criptarea parolei se va folosi un algoritm similar cu Cifrul lui Cezar, iar verificarea logarii se va face folosind fisierul whitelist.txt. Comenzile disponibile sunt urmatoarele:

- login (user) (parola) pentru autentificare user
- logout pentru deconectare user
- quit pentru intreruperea conexiunii clientului cu serverul
- list pentru a afisa continutul directorului curent pe server
- listc pentru a afisa continutul directorului curent local
- pwds pentru a afisa locatia curenta pe server
- pwdc pentru a afisa locatia curenta local
- help pentru a afisa comenzile disponibile
- mkd (nume) pentru a crea un director in locatia curenta pe server
- cwds (.. sau nume director) pentru a schimba locatia curenta pe server
- cwdc (.. sau nume director sau cale absoluta) pentru a schimba locatia curenta local
- rmd (nume) pentru a sterge un director din locatia curenta pe server
- dlf (nume) pentru a sterge un fisier de pe server
- rnm (nume nume nou) pentru a redenumi un fisier de pe server
- download (nume) pentru a descarca un fisier de pe server in locatia curenta local
- upload (nume) pentru a incarca un fisier local pe server

4.3 Scenarii reale de utilizare

1. Doi utilizatori se conecteaza la server cu succes folosind comanda login, primul incarca un fisier pe server folosind comanda upload, iar al doilea il descarca folosind download, realizand practic un transfer de fisiere intre useri.
2. Un utilizator are nevoie urgent de spatiu pe hard-disk-ul local, acesta se conecteaza la server, foloseste comanda login, apoi da upload unui fisier cat mai mare pe server si il sterge din propriul computer. Dupa ce elibereaza spatiul de care avea nevoie, acesta poate folosi comanda download pentru a descarca inapoi fisierul de pe server.
3. Crearea unui director folosind comanda mkd de catre un utilizator si folosirea directorului de catre mai multi utilizatori prin incarcare, descarcare si stergere de fisiere.(de exemplu daca lucreaza la un proiect comun)

5 Concluzii

Printre imbunatatirile posibile se numara folosirea unei metode de criptare a parolei mai securizata precum o functie hash, salvarea informatiilor utilizatorilor intr-o baza de date in loc de un fisier text si adaugarea unei comenzi de creare a unui utilizator nou(la care sa aiba acces doar administratorul sau un numar mic de utilizatori) in plus fata de cea de logare.

6 Referinte Bibliografice

– https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
– https://www.andreis.ro/teaching/computer-networks
– https://man7.org/linux/man-pages/index.html
– https://sequencediagram.org
– https://www.geeksforgeeks.org/tcp-ip-model/?ref=lbp