MyFileTransferProtocol

Proiect la disciplina - Rețele de calculatoare
Raport tehnic

Ungureanu Diana-Cristina, Anul 2, Grupa A5

Universitatea Alexandru Ioan Cuza, Facultatea de Informatică Iași
ungureanu.diana@info.uaic.ro
https://www.info.uaic.ro/

Abstract. La nivelul acestui raport tehnic am încercat să detaliez diferite aspecte (tehnologii utilizate, descrierea modului de funcționalitate a aplicației, arhitectura aplicației, detalii de implementare, concluzii) legate de proiectul **MyFileTransferProtocol**.
Prin intermediul acestei aplicații un utilizator logat poate realiza diferite operații la nivel de fișiere, directoare și , de asemenea, transferuri de fișiere între clienți si server. Din punct de vedere al securității, se vor folosi diferite mecanisme de autorizare pentru conturile utilizatorilor și un mecanism de transmitere securizată a parolei la autenficare.

Keywords: Fișier · Director · TCP · Server concurent · Thread · Client

1 Introducere

MyFileTransferProtocol este un proiect ce are ca și obiectiv dezvoltarea unei aplicații de tip client-server. Funcționalitatea aceasteia constă în transferul de fișiere între clienți și server. Un utilizator are posibilitatea de a accesa un terminal (pentru client) ce pune la dispoziție diferite comenzi. În prim plan vor exista trei posibilități: **login**, **quit** și **help** . Dacă utilizatorul se răzgândește și dorește să părăsească aplicația, va accesa comanda **quit**. Dacă dorește să se autentifice, va trebui să acceseze comanda **login**. După introducerea datelor, dacă acestea sunt valide conform bazei de date, user-ul va intra în aplicația propriu-zisă. În caz contrar, se va afișa mesajul corespunzător: user invalid, parolă invalidă, etc. Mecanismul de transmitere al datelor este unul securizat, atunci când se introduce parola apărând caractere space (empty, specific linux). După autentificarea cu succes, user-ul are posibilitatea de a afectua alte comenzi, cum ar fi cele pentru **operarea cu directoare și fișiere**: copiere, mutare, ștergere, creare, redenumire, căutare,etc. De asemenea, poate accesa **comanda de transfer ASCII sau BINAR** și **comanda de logout** (by default, avem nevoie și de o astfel de comandă odată ce ne-am autentificat). Se poate accesa **quit** dacă se dorește a se părăsi aplicația. Pe de altă parte, se va implementa un **mecanism de autorizare** (whitelist/blacklist) pentru conturile utilizatorilor. Un blacklist presupunea definirea unor "entități" suspecte/rău intenționate cărora

trebuie să li se refuze accesul sau drepturile de rulare pe o rețea sau un sistem. Un whitelist este o abordare mai strictă a controlului decât blacklist-ul deoarece acceptă doar "entitățile" ce s-au dovedit a fi sigure.

2 Tehnologiile utilizate

În implementarea aplicației am folosit un protocol de tipul TCP (Transmission control protocol) pentru comunicarea client-server. Acest protocol conține un set de reguli și proceduri care guvernează modul în care datele au fost transmise. Un astfel de protocol garantează faptul că datele sunt reasamblate în ordinea în care au fost trimise și, de asemenea, este capabil să retransmită datele pierdute (nici o dată nu este pierdută). Așadar, este asigurată integritatea datelor, TCP-ul fiind responsabil de fiabilitate în timpul transmisiei. Aceste aspecte sunt importante în implementarea proiectului **MyFileTransferProtocol** deoarece avem nevoie de o transmitere sigură și fără pierdere a datelor: nu ne-am dori ca transferul de fișier să eșueze, să primim un fișier empty sau un fișier cu un text trunchiat.
Pe de altă parte, dispune de un protocol ce se potrivește obiectivului nostru, acela de transfer de informații între client și server, precum fișierele (TCP trebuie folosit pentru **bulk data transfer**). Opțiunea de server **UDP** (User datagram protocol) nu este una favorabilă doarece fluxul de transmitere al datelor este continuu, deci livrarea datelor la destinație nu este asigurată de server-ul UDP. De asemenea, nu este ideal pentru transmiterea de fișiere, ci este folosit mai degrabă pentru broadcasting și multicasting.
Aplicația vizează un model frecvent întâlnit, server-ul oferind servicii clienților care rulează pe aceeași mașină sau la distanță: **modelul client/server**.
Limbajul folosit în implementarea ei este limbajul C.
De asemenea, pentru conturile utilizatorilor am folosit o bază de date SQL.

```
+----+-----------------+----------+-----------+
| id | username        | password | status    |
+----+-----------------+----------+-----------+
|  1 | diana.ungureanu | dcu88*R  | whitelist |
|  2 | cristina.pichiu | ccp903e  | whitelist |
|  3 | paul.ungureanu  | flvu3cV  | blacklist |
|  4 | ana.baciu       | as38rdc  | blacklist |
+----+-----------------+----------+-----------+
```

Fig. 1. Diagramă ce detaliază comunicarea dintre client și server

3 Arhitectura aplicației

3.1 Conceptele implicate

La implementarea aplicației se va utiliza un server concurent ce are ca și obiectiv îndeplinirea mai multor cereri provenite de la mai mulți clienți, acestea fiind procesate simultan (un astfel de server poate deservi mai mulți clienți la un moment dat). Am considerat că un server iterativ nu ar fi potrivit întrucât acesta iterează prin fiecare client, realizând cererile pe rând.
În acest sens, o idee ar fi să folosim primitiva **fork()** pentru a crea câte un proces fiu pentru deservirea fiecărui client. Din păcate, fork() poate fi un mecanism costisitor și trebuie să folosim ceva mai optim, cum ar fi **thread-urile** (primitiva caracteristică : **pthread_create()**. În principiu, thread-ul principal este blocat când se apelează primitiva accept(), iar de fiecare dată când este acceptat un client se crează un thread ce îi va procesa cererea. Cu toate acestea, mecanismul prin care procesele copil sau thread-urile pot apela accept() este mai simplu și mai rapid decât cel în care thread-ul principal apelează accept() și apoi transmite descriptorul thread-urilor.
Conceptual, la nivelul transport se remarcă câteva primitive importante, atât în server, cât și în client.

Table 1.

Primitivă	Rol
socket()	Asigură comunicarea dintre server și clienți.
bind()	Intervine conceptul de **port**, prin intermediul căruia un serviciu poate corespunde mai multor procese. Pentru a se realiza comunicarea despre care am menționat, trebuie să atașăm portului un socket. Acest lucru este posibil prin intermediul acestei primitive.
listen()	După ce portul va fi atașat, punem serverul să asculte dacă vin clienți să se conecteze prin intermediul acestei primitive (permiterea unui socket să accepte conexiuni) .
accept()	Acceptarea propriu-zisă a conexiunilor cu anumiți clienți. Prin intermediul acesteia, programul se blochează până când apare o cerere din partea unui client.
read() write()	Comunicarea între clienți și server se realizează prin intermediul acestor primitive (read() - citire de date, write() - scriere de date).
close()	Încheierea efectivă a comunicării dintre clienți și server (eliberează un socket).
connect()	În contextul unui client de tipul TCP, vom folosi această metodă, în locul primitivei accept().

3.2 Diagrama aplicației detaliată

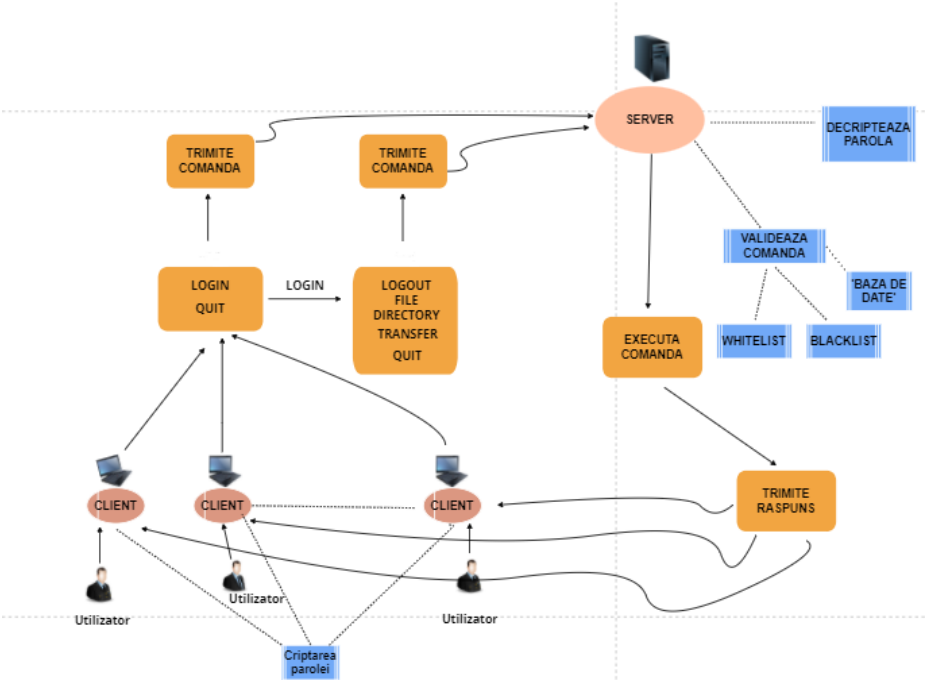

Fig. 2. Diagramă ce detaliază comunicarea dintre client și server

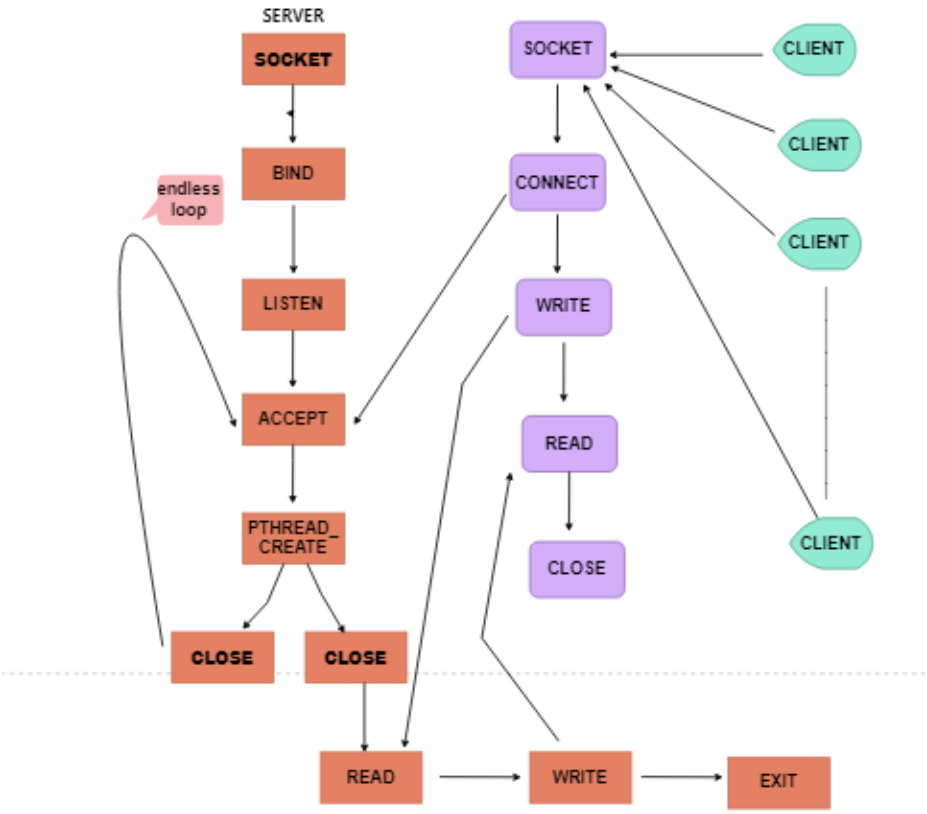

Fig. 3. Diagrama aplicației

4 Detalii de implementare

4.1 Scenarii de utilizare

Principiul de funcționare al acestei aplicații este destul de simplu. La nivel de comunicare între server și client, server-ul este blocat până când un user execută o comandă în client. Practic, server-ul așteaptă să deservească clienții. În prim plan, user-ul are opțiunea de a accesa comanda de login sau de quit. Odată ce selectează comanda de quit, va ieși din aplicație, iar execuția server-ului și clientului se va termina. Dacă alege opțiunea de login, acesta are posibilitatea de a introduce username-ul și parola.
La nivel de username, programul va verifica dacă acesta a introdus corect datele, conform formatului impus de aplicație, iar dacă nu, va primi un warning specific erorii făcute. Practic, clientul va citi prin intermediul primitei **read()** un șir de caractere introdus de la tastatură, îl va trimite mai departe server-ului , care va verifica dacă acesta coincide standardului propus. Server-ul, la rândul său, va trimite înapoi clientului mesajul corespunzător prin intermediul primitivei **write()**.
La nivel de parolă, aceasta va fi criptată, atunci când se tastează, nu vor apărea caracterele propriu-zise, ci caractere space (empty, specific Linux). Clientul va trimite, de asemenea, șirul de caractere criptat server-ului, acesta îl va interpreta, îl va decripta și va trimite mesajul corespunzător clientului.
Dacă ambele câmpuri sunt completate corect (se potrivesc datelor stocate în baza de date), programul va verifica pe baza mecanismului de autorizare implementat, dacă într-adevar este safe ca acel utilizator să fie logat în aplicație (whitelist/blacklist tot din baza de date) și să execute anumite comenzi. De asemenea, server-ul va trimite un șir de caractere ce va da un răspuns în acest sens, clientul în va citi șirul de caractere și se va decide dacă user-ul va putea executa comenzile propriu-zise ce stau la baza funcționalității proiectului. În caz afirmativ, se vor putea executa comenzile respective. Este important de precizat că se poate face logout , cu consecința faptului că odată delogat, se revine la posibilitatea accesării unor comenzi minimale, cum ar fi quit și bineînțeles, login din nou. Odată logat, în fișierul de validare, va apărea valoarea 1, ceea ce înseamnă că există posibilitatea utilizării aplicației , iar odată delogat, va apărea 0, ceea ce înseamnă contrariul.
Funcționalitatea aplicației constă, de fapt, în accesarea comenzilor **Transfer** și cele pentru **operarea cu fișiere și directoare**.
La nivel de fișiere și directoare, se vor utiliza comenzile:rnf (redenumire fișier) , ,mv (mutare fișier) ,create (creare fișier) , ls:1 (fișiere din client), ls:2 (fișiere din server), crd (director curent),upload:ASCII,upload:BINARY, download:ASCII, download:BINARY.

4.2 Cod relevant particular proiectului

Funcția pentru tranfer de fisiere media de la server la client
Atunci cand se introduce comanda **download_media:fis**, serverul preia ramura "dmedia" si efetueaza transferul.

```
if(strncmp(raspuns2,"dmedia##",8)==0)
{

FILE *pentru_server1,*pentru_server2;
char username1[30],username11[3000],username111[3000];

bzero(username1,30);
bzero(username11,3000);
//bzero(username111,30000);
int nr=0,poz=0;

if(strlen(raspuns2)==8)
{

strcpy(mesaj,"Comanda invalida!");
return 1;

}
for(int i=0;i<strlen(raspuns2);i++)
{

if(raspuns2[i]=='#')
{poz=i; break;}
}

if(raspuns2[poz]=='#'&&raspuns2[poz+1]=='#')

{
nr=0;
for(int i=poz+2;i<strlen(raspuns2);i++)
{
if(raspuns2[i]!='#')
username1[nr++]=raspuns2[i];
```

```
if(raspuns2[i]!='#')
username1[nr++]=raspuns2[i];
else
{poz=i;
 lungime=atoi(username1);
break;
}
}
}
else
{strcpy(mesaj,"Comanda invalida!");
return 1;
}

if(raspuns2[poz]=='#'&&raspuns2[poz+1]=='#')

{
nr=0;
for(int i=poz+2;i<strlen(raspuns2);i++)
{
if(raspuns2[i]!='#')
username11[nr++]=raspuns2[i];
else
{poz=i;break;}
}
}
else
{strcpy(mesaj,"Comanda invalida!");
return 1;}



char sir8[100],sir9[100];
strcpy(sir8,"/home/diana/clientFiles/");
strcpy(sir9,"/home/diana/serverfiles/");
strcat(sir8,username11);
strcat(sir9,username11);
```

```
strcpy(sir9,"/home/diana/serverfiles/");
strcat(sir8,username11);
strcat(sir9,username11);
strcpy(username11,sir8);
strcpy(username111,sir9);


pentru_server1=fopen(username111,"rb");
pentru_server2=fopen(username11,"wb");




while( (size = fread(m, 1, sizeof(m), pentru_server1) ) > 0)

{

 if(fwrite(m, 1, size, pentru_server2)!=size)
```

5 Concluzii

Așadar, la nivel de implementare, am încercat să fac o aplicație interactivă și cât mai optimizată, însă există unele îmbunătățiri pe care încă le-aș putea face. Una dintre ele ar fi aceea de a încerca să fac transfer de fișiere și pentru mașini aflate la distanță.

References

1. http://www.posdru.ugal.ro/topacademic/documente/GHID_SCRIERE_ARTICOL.pdf
2. https://online.visual-paradigm.com/
3. https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
4. https://profs.info.uaic.ro/~andreis/index.php/computernetworks/
5. https://www.lifesize.com/en/blog/tcp-vs-udp/
6. https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
7. https://stackoverflow.com/questions/790932/how-to-wrap-text-in-latex-tables
8. https://www.cs.dartmouth.edu/~campbell/cs50/socketprogramming.html
9. https://youtu.be/Vdc8TCESIg8
10. https://youtu.be/ve82kSSj_Hs
11. https://youtu.be/VkfN4kWU2P8