RC-proiect

Flavia Bechea

January 2025

1 Introducere

Voi prezenta proiectul **"MyFileTransferProtocol"** dn categoria B. Proiectul presupune transferul de fișiere dintre client și server. Serverul pune la dispoziția clienților comenzi ce permit autentificarea și operații cu directoare și fișiere. De asemenea există un mecanism de autorizare whitelist/blacklist care verifică utilizatorii și un mecanism de transmitere criptată. La nivelul operațiilor, clienții au posibilitatea de a:

- crea fișiere și directoare
- șterge
- redenumi
- copia
- muta
- schimba directorul de lucru
- lista director
- descărca.

2 Tehnologii aplicate

Acest proiect va avea la baza un **server concurent TCP**, având un proces separat pentru fiecare client prin **fork()**, care asigură concurența. Protocolul **TCP** este un protocol ce asigură ca:

- transferul de fișiere dintre client și server să fie controlat și de încredere, fără pierdere de informații.
- datele să fie transmise în pachete, iar la destinație să ajungă toate și în aceeași ordine.
- fișierele să fie transmise între client și server fără pierderi.

Serverul poate să deserveacă mai mulți clienți simultan fără ca transferurile să interfereze între ele, adică fiecare client este independent datorită fork-ului care crează un proces separat penru fiecare copil.

În cadrul proiectului este nevoie de integritatea datelor, deoarece autentificarea se face pe baza de transmitere la server de parolă și username. O posibilă pierdere a datelor poate să ducă la nelogarea unui client care se află în whitelist. De aceea o abordare pe baza unui protocol **UDP** nu ar fi funcționat la acest proiect, fiind protocol ce nu asigură ajungerea la destinație a pachetelor trimise independent.

3 Structura Aplicației

În aplicația făcută am luat în considerare folosirea de fișiere XML pentru o bună organizare a datelor astfel:

- **whitelist.xml** conține pentru fiecare user, 3 informații: username, password și state(delogat/logat)
- **blacklist.xml** conține pentru fiecare user, 2 informații: username și password.

Aplicația permite comunicarea între server și client. Clientul va transmite către server anumite comenzi, iar serverul le va executa, trimițând înapoi mesaje clientului în legătură cu statusul comenzii executate (daca s-a efectuat în mod corect sau dacă pe parcurs s-au întâmpinat erori).

La început clientul va putea transmite doar **login** sau **quit**. Orice altceva fiind considerată comandă invalidă. Dacă comanda este **login**, clientul va introduce username și parolă, iar atunci când o va transmite serverului, va cripta parola folosind funcția **SHA-256**. Serverul după ce va primi comanda de la client va executa în funcție de statutul de logat. Dacă clientul nu este logat, se va verifica în fișierele .xml dacă există.

- s-a găsit în whitelist.xml atunci se va verifica dacă este deja logat (de pe alt dispozitiv) sau nu și se va modifica corespunzator.
- s-a găsit în blacklist.xml atunci se va face ieșirea bruscă din aplicație a clientului.

Odată făcută logarea, serverul va realiza intrarea în directorul cu numele corespunzător al clientului (numele userului cu care s-a făcut logarea și id-ul unic care reprezintă ordinea în fișierul whitelist.xml. ex: flavia1) clientul va putea să trimită comenzi la server pentru a face operații cu fișiere și directoare. Va putea să ceară următoarele comenzi:

- create_file name
- create_dir name
- exists file/folder
- remove file/folder
- rename name_old to name_new
- copy file to folder (copiază un fișier din directorul curent în alt director)
- move file to folder (mută un fișier din directorul curent în alt director)
- download file (with new_name) (descarcă un fișier în directorul curent; dacă nu se specifică numele cu care vreau să apară descărcarea în directorul curent, atunci va rămâne cu numele fișierului original)
- printf_dir (listează toate fișierele și directoarele din directorul curent de lucru)
- change_p_dir (se merge în directorul părinte)
- change_dir (se merge într-un alt director)

Pentru delogare se scrie comanda **logout** în care se deloghează și revine la folderul cu proiectul și așteaptă eventualele logări. De asemenea, prin comanda **quit** se face ieșirea clientului din aplicație și delogarea clientului în cazul în care a fost logat.

Dacă la încercarea logării s-au introdus date care nu se găsesc nici în blacklist și nici în whitelist de 3 ori, atunci se va face ieșirea automată din aplicație a clientului și adăugarea ultimului username și password încercat în blacklist pentru a asigura protejarea utilizatorilor și a datelor personale a acestora.

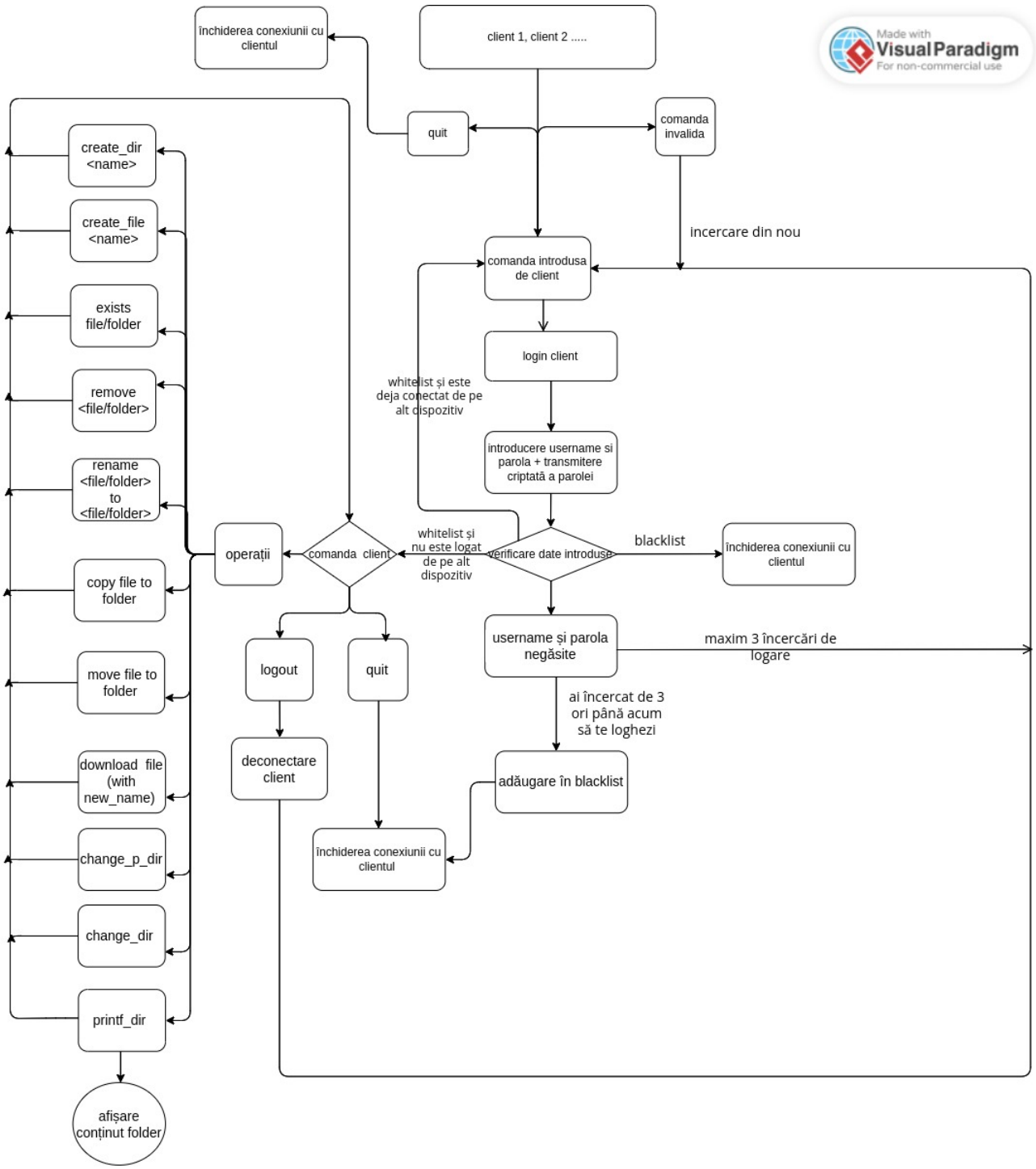

Figure 1: Diagrama aplicației

4 Aspecte de Implementare

4.1 Prezentarea secțiunilor de cod specifice

4.2 În server

Serverul este de tip TCP concurent, deci va trebui creat un socket pentru comunicarea cu clientii.

```
1 if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
```

```
        handle_error("Eroare la creare socket");
```

Se umple structura folosită de server prin specificarea informațiilor de adresă

```
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl (INADDR_ANY);
server.sin_port = htons (PORT);
```

După crearea socket-ului trebuie atașat la port și pregătit pentru a asculta cererile clienților de stabilire a conexiunii

```
if (bind (sd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
        handle_error("Eroare la atasarea socketului");
if (listen (sd, 1) == -1)
        handle_error("Eroare la ascultarea clientilor");
```

Serverul va trebui să accepte în mod constant clienții care se conecteaza la el.

```
if((client = accept (sd, (struct sockaddr *) &from, &length)) < 0) //
    acceptare a clientilor
            {
              perror("Eroare la acceptare clienti");
              continue;
            }
```

După acceptarea fiecarui client, se va apela funcția **fork()** pentru a crea un proces copil corespunzător pentru fiecare client, astfel se realizează **concurența** necesară în acest proiect. Părintele trebuie să închidă descriptorul client de după **accept()** pentru a nu trimite și primi date prin intermediul lui. Fiul creat va închide socket-ul inițial (sd) pentru a nu aștepta noi conexiuni în rețea.

```
if ((pid = fork()) == -1) //cream un proces pentru fiecare client
        {
                close(client);
                continue;
        }
        else if (pid != 0) // parinte
        {
                close(client); //se inchide conexiunea cu clientul
                while(waitpid(-1,NULL,WNOHANG));
                continue; //se intoarce la while(1) ca sa accepte si alti
                    clienti
        }

        else if (pid == 0) // copil
        {
                close(sd); //se inchide socketul care comunica cu server
            //nu trebuie sa ascultam alte conexiuni
            ...
        }
```

Pentru asigurarea unui schimb de mesaje între clienți și server, se va adauga un **while(1)** în fiecare copil din server și în client.

```
while(1)
    {
        /* citirea mesajului */
        if (read (client, msg, 100) <= 0)
        {
            perror ("[server]Eroare la read() de la client.\n");
            close (client);     /* inchidem conexiunea cu clientul */
            continue;           /* continuam sa ascultam */
        }
        printf ("[server]Mesajul a fost receptionat...%s\n", msg);
```

```
11
12          ///comenzile
13
14          if (write (client, msgrasp, 100) <= 0)
15          {
16              perror ("[server]Eroare la write() catre client.\n");
17              continue;           /* continuam sa ascultam */
18          }
19          else
20              printf ("[server]Mesajul a fost trasmis cu succes.\n");
21      }
```

La partea de comenzi s-au adăugat pe lângă recunoașterea lor, funcționalitățile.

4.3 În client

Clientul va asigura crearea socket-ului,unde structura addr conține IP-ul și portul serverului la care se conectează prin apelul **connect()**.

```
1  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
2        handle_error("Eroare la creare socket");
3
4  server.sin_family = AF_INET;       // familia socket-ului
5  server.sin_addr.s_addr = inet_addr(argv[1]);       //adresa IP a serverului
6  server.sin_port = htons (port);         //portul de conectare
7  if (connect (sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) ==
       -1)
8        handle_error("Eroare la conectarea clientului la server");s
```

4.4 Transmiterea parolei în mod securizat

Clientul înainte să trimită la server perechea username, password, va folosi o funcție de criptate a parolei **SHA-256**. Această funcție este una hash criptografică care face parte din familia **SHA-2** și este folosită pentru a genera o valoare hash fixă pe 256 de biți adica 32 de octeți. Transformarea în hexazecimal se folosește pentru a reprezenta datele binare într-un format lizibil și compact, astfel parola va fi trasmisă la server ca o secventă de 64 de caractere după transformarea în hexa.

Am ales acest algoritm de criptare pentru că:

- oferă siguranță ridicată
- este rezistent la coliziuni(2 intrări diferite generează același hash)
- este dificil să deduci intrarea originală pe baza unui hash SHA-256
- rezistent la atacuri

```
1  void hash(const char *password, char *hashed_password)
2  {
3      unsigned char hash[32];
4      SHA256((unsigned char *)password, strlen(password), hash);
5
6      for (int i = 0; i < 32; i++)
7      {
8          sprintf(hashed_password + (i * 2), "%02x", hash[i]); // convertire
               in hexa
9      }
10     hashed_password[64] = '\0';
11 }
```

4.5 Folosirea fișiere XML pentru păstrarea datelor

Am ales să folosesc fișiere **XML** pentru că au o organizare ierarhică și sunt ușor de citit. Datorită faptului că se dorea reținerea statusului de logat sau delogat a utilizatorului, folosirea unor fișiere normale nu ar fi fost o soluție optimă pentru că ar fi trebuit să reconstruim complet fișierul de fiecare dată când se modifica statusul. În cazul fișierelor **XML**, modificarea unui anumit loc se face ușor și practic și nu se afectează întreaga structură a fișierului. Tot scopul acestui status de logat sau delogat este gestionarea logării unui utilizator adică odată logat într-un proces să nu se mai poată loga cu aceleași date și în alt proces, pentru a evita lucrul pe același director din două procese simultan și diferite.

Crearea de fișiere XML:

În fișierul whitelist.xml vor exista mereu niste utilizatori standard.

În fișierul blacklist.xml se pot adăuga în funcție de numărul de încercări ale clienților de logare.

```
xmlDocPtr fisier = xmlNewDoc(BAD_CAST "1.0");
xmlNodePtr radacina = xmlNewNode(NULL, BAD_CAST "whitelist");
xmlDocSetRootElement(fisier, radacina);

// adaug 4 utilizatori pentru whitelist

Adaugare_utilizator_white(radacina, "flavia", "
    fab71cb26e6a0ff4d25ead092b09df43fec91d2182f48769e066a4a7eb6d483b", "
    delogat");
```

Căutarea în fișierele XML:

Se va încărca fișierul într-o structura de date XML și se va afla nodul rădăcină.

```
xmlDocPtr fisier;
if ((fisier = xmlReadFile("whitelist.xml", NULL, 0)) == NULL)
{
    return;
}

xmlNodePtr radacina;
if ((radacina = xmlDocGetRootElement(fisier)) == NULL) // se va lua
    radacina
{
    return;
}
```

Se iterează prin nodurile copil ale rădăcinii și identifică toate nodurile user

```
xmlNodePtr user = radacina->children;
while (user != 0) {
    if (user->type == XML_ELEMENT_NODE && xmlStrcmp(user->name, BAD_CAST "
        user") == 0) {
        (*id)++;
        printf("Al␣%d␣utilizator\n", *id);
        ...
    }
    user = user->next;
}
```

Caută nodurile username, password și state în cadrul unui nod user.

```
xmlNodePtr copil = user->children;
while (copil != 0) {
    if (copil->type == XML_ELEMENT_NODE && xmlStrcmp(copil->name, BAD_CAST
        "username") == 0) {
        u = (const char *)copil->children->content;
    }
    if (copil->type == XML_ELEMENT_NODE && xmlStrcmp(copil->name, BAD_CAST
        "password") == 0) {
```

```
7           p = (const char *)copil->children->content;
8       }
9       if (copil->type == XML_ELEMENT_NODE && xmlStrcmp(copil->name, BAD_CAST
            "state") == 0) {
10          s = (const char *)copil->children->content;
11          nod_state = copil;
12      }
13      copil = copil->next;
14  }
```

4.6 Scenarii reale de utilizare

- Accesarea documentelor doar de personalul autorizat dintr-o companie.
- Transfer de fișiere dintre angajații unei companii. Adică un angajat poate să îi transfere un fișier altui angajat, astfel se realizează distribuirea de documente.
- Distribuirea documentelor către toții elevii care au cont de către profesori.(comanda copy si move permit acest lucru de a lua un fisier din directorul curent si de a-l deplasa oriunde). Din contul său de profesor poate să transfere către ceilalți elevi documente sau chiar să descarce temele lor prin comanda download.

5 Concluzii

Posibile înbunătățiri sunt:

- crearea unei interfațe grafice pentru a face utilizarea aplicației de către client cât mai bună.
- folosirea TCP cu multiplexare sau thread-uri, pentru o eficiență mai bună si pentru a nu întampina probleme atunci când există mulți clienți simultan.
- folosirea unei baze de date în SQLite pentru reținerea utilizatorilor pentru a oferi o siguranță a datelor suplimentară și a face gestionarea datelor mai ușoară.
- adăugarea de informații noi despre utilizatori cum ar fi rolul, adică fiecare să aibă anumite drepturi asupra unui fișier în funcție de rolul său
- implementarea unui sistem de salvare a istoricului
- istoric al fiecărui utilizator stocat într-o bază de date

6 Referințe Bibliografice

1. https://www.overleaf.com/learn/latex/Code_listing
2. https://online.visual-paradigm.com/community/share/login-1evf80545q
3. https://profs.info.uaic.ro/andrei.scutelnicu/teaching/ComputerNetworks/Lab7/Lab7.pdf
4. https://profs.info.uaic.ro/andrei.scutelnicu/teaching/ComputerNetworks/Lab6/Lab6.pdf
5. https://edu.info.uaic.ro/computer-networks/files/7rc_ProgramareaInReteaIII_En.pdf
6. https://www.w3schools.com/xml/
7. https://stackoverflow.com/questions/399704/xml-parser-for-c?utm_source=chatgpt.com

8. https://gnome.pages.gitlab.gnome.org/libxml2/devhelp/libxml2-tree.html

9. https://stackoverflow.com/questions/2262386/generate-sha256-with-openssl-and-c