MyFileTransfer

Galateanu Petru-Ioan

Faculteatea de Informatica, Universitatea "Alexandru Ioan Cuza" din Iasi

Abstract

Această documentație descrie implementarea unui server FTP scris în limbajul C utilizând protocolul TCP. Proiectul include autentificare utilizatori, transfer de fișiere și alte funcționalități de bază ale unui server FTP.

Contents

1 Introducere

- Proiectul de față își propune să implementeze un server FTP simplu, care utilizează protocolul TCP pentru a facilita transferul fișierelor între client și server într-un mod eficient și securizat. FTP (File Transfer Protocol) este un protocol standard de rețea utilizat pentru transferul fișierelor între un client și un server, fiind folosit frecvent în aplicațiile de administrare a serverelor și în procesul de partajare a fișierelor.
- Scopul principal al acestui proiect este de a crea o aplicație client-server bazată pe tehnologiile de programare C și socket programming, care va permite utilizatorilor să se autentifice pe server, să încarce și să descarce fișiere, și să gestioneze directoare.
- Proiectul va include funcționalități de bază, precum autentificarea utilizatorilor pe baza unui fișier de tip whitelist, manipularea fișierelor de pe server (upload, download, ștergere, redenumire), precum și protecția împotriva accesului neautorizat.

2 Tehnologii aplicate

2.1 Networking (TCP/IP)

- **Sistem de socket-uri**:
  - Functiile `socket()`, `bind()`, `listen()`, `accept()` sunt utilizate pentru a crea si gestiona conexiunile de retea.
  - Functiile `send()` si `recv()` trimit si primesc date intre server si client.
- **Protocol TCP**: Serverul utilizeaza protocolul TCP, care asigura comunicare fiabila, orientata pe conexiune.

2.2 Procesare concurenta

- **Crearea proceselor cu `fork()`**:
  - Serverul foloseste functia `fork()` pentru a crea un proces nou pentru fiecare client.
- **Managementul proceselor**: Procesul parinte continua sa accepte conexiuni, iar procesul copil gestioneaza fiecare client.

2.3 Manipulare fisiere

- **I/O pentru fisiere**: Serverul implementeaza functii pentru:
  - *Upload*: Functia `recieve_file()` salveaza fisiere primite de la client.
  - *Download*: Functia `respond_to_download()` trimite fisiere clientului.
- **Functii POSIX**: Utilizarea functiilor precum `fopen()`, `fwrite()`, `fread()`, `remove()` si `rename()` pentru manipularea fisierelor.
- **Directorul curent**: Comenzi precum `chdir()` si `getcwd()` sunt utilizate pentru a schimba si afisa directorul curent.

2.4 Securitate de baza

- **White-listing pentru autentificare**:
  - Serverul verifica daca un utilizator este autorizat comparand datele din fisierul `whitelist.txt`.
  - Parolele sunt criptate si decriptate cu o functie manuala (`decrypt()`).
- **Restrictii**: Nu permite manipularea fisierelor critice (`server`, `client`, `whitelist`).

2.5 Manipularea directoarelor

- **Crearea directoarelor**: Functia `mkdir()` creeaza noi directoare.
- **Listarea fisierelor**: Utilizarea `opendir()` si `readdir()` pentru afisarea fisierelor dintr-un director.

2.6 Structuri de date si criptare custom

- **Structura** `account`:
  - Structura stocheaza informatii despre utilizator (nume si parola).
- **Decriptare custom**: Algoritm simplu care mapeaza caractere criptate la cele originale.

2.7 Managementul erorilor

- Functia `perror()` este utilizata pentru afisarea mesajelor de eroare.

2.8 Comunicare intre server si client

- Implementarea comenzilor `mkdir`, `rename`, `delete`, `list`, `upload`, `download`, `help`.

2.9 Proiectare modulara

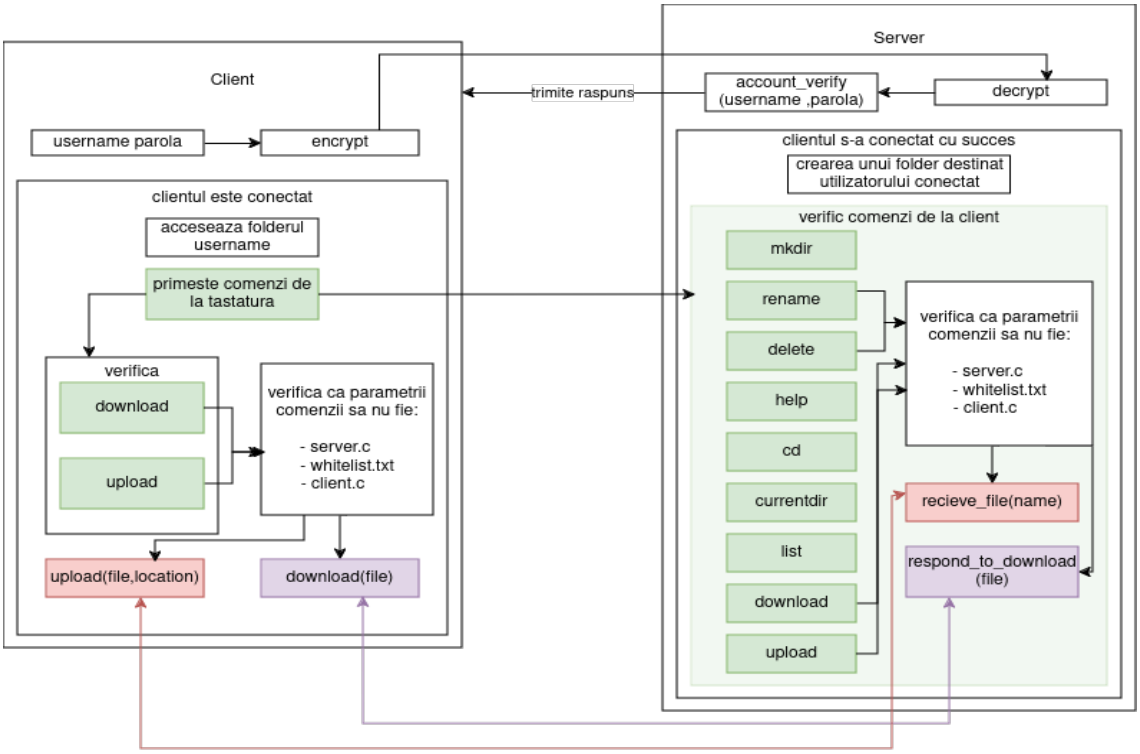
- Functiile sunt separate pentru a gestiona fiecare functionalitate (`client_handler()`, `recieve_file()`, `account_verify()` etc.).

2.10 Utilizarea bibliotecilor POSIX

- Biblioteci precum `sys/socket.h`, `netinet/in.h`, `unistd.h`, `dirent.h` sunt utilizate pentru retea, procese si manipularea directoarelor.

3 Structura aplicatiei

3.1 Diagrama



3.2 Prezentarea componentelor principale ale diagramei

- **Client:**
  - Primeste de la tastatura datele introduse de utilizator *username parola* si le cripteaza.
  - Trimite catre server datele criptate.
  - Daca primeste de la server mesajul specific pentru o conectare reusita, acceseaza folderul cu numele *username* si primeste comenzi de la utilizator.
  - Trimite serverului comenzile.
- **Server:**
  - Verifica datele de conectare folosind fisierul *whitelist.txt*
  - Daca datele de conectare sunt corecte, creeaza un folder cu numele utilizatorului care s-a conectat.
  - Primeste comenzi de la client.
  - Gestioneaza comenzile clientului.

4 Aspecte de implementare

4.1 Arhitectura generala

Aplicatia are o arhitectura client-server, utilizand protocolul TCP/IP.

4.2 Autentificare

- Serverul decripteaza parola primita de la client si valideaza utilizatorii folosind un fisier *whitelist.txt* care contine perechi de forma: *username parola*.

- Administratorul serverului este singurul utilizator care dupa inregistrare poate crea sau sterge utilizatori. Conturile pe care acesta le creeaza sunt salvate in *whitelist.txt*.
  - Functionalitatea de stergere a conturilor functioneaza astfel:
    * Administratorul foloseste comanda $delete\_account < username >$.
    * Functia *delete_account* creeaza un fisier *temp.txt* in care va scrie toate perechile *username parola*, mai putin contul specificat de administrator. Am folosit functia $fscanf()$ pentru a verifica toate conturile existente in fisierul *whitelist.txt*.
    * Dupa ce tot fisierul *whitelist.txt* a fost parcurs, il sterg si redenumesc fisierul *temp.txt* in *whitelist.txt*.
- Algoritmul de codificare si decodificare utilizat pentru protectia parolelor este unul simplu, bazat pe substituirea fiecarui caracter conform unui tabel de mapare.
- Fluxul de autenitificare:
  - Clientul trimite numele de utilizator si parola codificata catre server.
  - Serverul decodifica parola primita si o compara cu valorile stocate in *whitelist.txt*.
  - Daca combinatia *username parola* este valida, clientul este conectat si poate trimite comenzi.

4.3 Operatiuni gestionate de server

Serverul gestioneaza mai multe tipuri de comenzi trimise de client:

- **Comanda help(`help`)**
  Ofera informatii utilizatorului referitor la comenzile pe care le poate utiliza si functionalitatea acestora.

4.3.1 Operatii pe directoare:

- **Creare directoare(`mkdir nume`):**
  Creeaza un director denumit *nume*.

```
if(strncmp(command, "mkdir", 5) == 0)
    char directory_name[BUFF_SIZE];
    sscanf(buff+6, "%s", directory_name);

    if(mkdir(directory_name, 0777) == 0)
        send(client_socket, "[server]Directory created successfully!",..);
    else
        send(client_socket, "[server]Failed to create directory",..);
```

- **Aflarea directorului curent(`currentdir`):**
  Afiseaza directorul curent de lucru.

- **Schimbarea directorului de lucru (`cd <dir>`)**
  Comanda `cd ..` schimba directorul de lucru catre parintele directorului actual.

```
if (strncmp(command, "cd", 2) == 0)
    char new_location[BUFF_SIZE];
    sscanf(buff + 3, "%s", new_location);
    int ok = 0;

    if (strncmp(new_location, "..", 2) == 0)
        ok = 1;
        if (chdir("..") == 0)
            char cwd[1024];
            if (getcwd(cwd, sizeof(cwd)) != NULL)
                send(client_socket, cwd, 100, 0);

    if (ok == 0)
        if (chdir(new_location) == 0)
            send(client_socket, "[server] Directory changed",..);
        else
            send(client_socket, "[server] Cannot change directory",..);
        ok = 0;
```

4.3.2 Operatii pe fisiere:

- **Afisare fisierelor din directorul curent(`list`):**
  Afiseaza fisierele din directorul curent de lucru.

```
if (strncmp(command, "list", 5) == 0)
    char output[BUFF_SIZE];
    struct dirent *entry;
    DIR *dir = opendir(".");
    memset(output, 0, BUFF_SIZE);

    while ((entry = readdir(dir)) != NULL)

        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
        // excludem intrarile implicite ce reprezinta directorul curent si cel parinte
            continue;
        strcat(output, entry->d_name);
        strcat(output, "\n");
    closedir(dir);
    send(client_socket, output, sizeof(output), 0);
```

- **Redenumire fisier(`rename nume nume1`):**
  Redenumeste fisierul *nume* in *nume*1.

```
if(strncmp(command, "rename", 6) == 0)
    char old_name[BUFF_SIZE],new_name[BUFF_SIZE];
    sscanf(buff+7, "%s %s", old_name, new_name);

    if(rename(old_name, new_name) == 0)
        send(client_socket, "[server]File renamed successfully",..);
    else
        send(client_socket, "[server]Failed to rename the specified file",..);
```

- **Stergere fisier(`delete <nume>`):**
  Elimina fisierul *nume* daca acesta nu este critic pentru functionarea corespunzatoare a aplicatie.

```
if(strncmp(command, "delete", 6) == 0)
    char file_to_delete[BUFF_SIZE];
    sscanf(buff+7, "%s", file_to_delete);

    if(strncmp(file_to_delete,"server",6)!=0 && ..)
        if(remove(file_to_delete) == 0){
            send(client_socket, "File deleted!",..);
        else
            send(client_socket, "Can't delete the file",..);
    else
        send(client_socket, "Not allowed",..);
```

- **Incarcarea fisierelor pe server(`upload <path> <nume>`):**
  Incarca fisierul indicat de *path* sub numele *nume*.
  - Client:
    - Trimite serverului numele sub care utilizatorul doreste sa salveze fisierul incarcat.
    - Functia *upload(path)* trimite dimensiunea fisierului pe care doreste sa il incarce, apoi incepe sa trimita pachetele de date.
    - Dupa ce a terminat, trimite semnalul *END*.
  - Server:
    - Functia *recieve_file* creeaza un fisier cu numele care a fost furnizat de client si primeste dimensiunea fisierului pe care urmeaza sa il primeasca.
    - Cat timp dimensiunea pachetelor primite este mai mica decat dimensiunea totala a fisierului si nu a fost gasit semnalul *END*, scrie in fisierul creat.
    - Dupa ce s-a incheiat transferul, trimite clientului un mesaj de incheiere si asteapta alte comenzi.

- **Descarcarea fiserelor de pe server(`download <fisier>`):**
  Descarca fisierul specificat in folderul utilizatorului care a solicitat transferul.
  - Client: utilizeaza functia *download*.
  - Server: utilizeaza functia *respond_to_download*.
  - Fluxul de informatii transmise este asemanator metodei *upload*.

5 Concluzii

5.1 Final

- În cadrul acestui proiect, am implementat un server FTP simplu, utilizând protocolul TCP, pentru a demonstra aplicarea principiilor fundamentale ale rețelelor de calculatoare, a protocoalelor de comunicație și a tehnologiilor de securitate. Proiectul mi-a oferit oportunitatea de a înțelege și aplica concepte esențiale din domeniul rețelelor, cum ar fi gestionarea conexiunilor prin socket-uri, utilizarea protocoalelor de rețea și implementarea autentificării și controlului accesului.
- Folosind limbajul C, am dezvoltat un server capabil să gestioneze multiple cereri simultane, să permită transferul de fișiere între client și server și să aplice măsuri de securitate de bază, precum whitelist-ul pentru autentificare. În acest sens, proiectul mi-a permis să aprofundez cunoștințele despre programarea rețelelor și să îmbunătățesc abilitățile de manipulare a fișierelor și directoarelor într-un context de server-client.

5.2 Potentialele imbunatatiri

- **Sistem de logare si monitorizare**
  Presupune inregistrarea tuturor operatiunilor si incercarilor de autentificare intr-un fisier (log) pentru detectarea atacurilor sau a abuzurilor.
- **Utilizarea thread-urilor in locul fork()**
  Pentru o performanta mai buna, codul poate fi optimizat folosind thread-uri in locul proceselor copil pentru fiecare client conectat.
- **Premisiuni pentru utilizatori**
  Introducerea unui sistem de permisiuni care sa defineasca cine poate incarca sau descarca un fisier.
- **Functionalitati pentru gestionarea utilizatorilor**
  Adaugarea posibilitatii de inregistrare a utilizatorilor noi si gestionarea conturilor (schimbarea username-ului, resetarea parolei, etc.)
- **Interfata grafica**
  Dezvoltarea unei interfete grafice pentru utilizatori, care sa simplifice utilizarea aplicatiei, eliminand necesitatea introducerii manuale a comenzilor.

6 Referinte bibliografice

1. Geeks for Geeks. *Differences between TCP and UDP*. https://www.geeksforgeeks.org/differences-between-tcp-and-udp/
2. C++ Documentation. *strcspn*. https://cplusplus.com/reference/cstring/strcspn/
3. Geeks for Geeks. *memset*. https://www.geeksforgeeks.org/memset-in-cpp/
4. Diagram. *Online Diagram Software*. https://app.diagrams.net/
5. Linux Documentation. *socket(7)*. https://man7.org/linux/man-pages/man7/socket.7.html
6. C++ Documentation. *strncmp*. https://cplusplus.com/reference/cstring/strncmp/
7. Overleaf. *online LaTeX editor*. https://www.overleaf.com/
8. Geeks for Geeks. *Socket Programming in C*. https://www.geeksforgeeks.org/socket-programming-cc/
9. Linux Documentation. *bind(2)*. https://man7.org/linux/man-pages/man2/bind.2.html

10. Geeks for Geeks. *TCP 3-Way Handshake Process.* https://www.geeksforgeeks.org/tcp-3-way-handshake-process/

11. Geeks for Geeks. *Create Directory or Folder with C/C++ Program.* https://www.geeksforgeeks.org/create-directoryfolder-cc-program/

12. Geeks for Geeks. *rename function in C.* https://www.geeksforgeeks.org/rename-function-in-ccpp/

13. Geeks for Geeks. *C program to delete a file.* https://www.geeksforgeeks.org/c-program-delete-file/

14. Geeks fo Geeks. *Switch Statement in C.* https://www.geeksforgeeks.org/c-switch-statement/

15. C++ Documentation. *sscanf.* https://cplusplus.com/reference/cstdio/sscanf/

16. Stack Overflow. *dirent.h.* https://stackoverflow.com/questions/40813865/c-dirent-h-and-syntax-for-getting-absolute-path

17. Geeks for Geeks. *fork().* https://www.geeksforgeeks.org/create-processes-with-fork-in-cpp/