Raport tehnic - (B) MyFileTransferProtocol

A2: Bărăscu Bianca

13 decembrie 2023

1 Introducere

Arhitectura care stă la baza proiectului este una de tip client & server care va permite transferul de fișiere dinspre client spre un spațiu de stocare gestionat de server. Motivul pentru care am făcut alegerea acestui proiect este acela că o astfel de aplicație își găsește rolul și în industrie, fiind o simplificare a unor servicii pe care le folosim, precum serviciile de stocare în cloud, încărcarea datelor pe un server (web) etc.

2 Tehnologii aplicate

Implementarea serverului va fi făcută prin folosirea unui protocol TCP, ce va juca atât rolul de conexiune de control (comenzile de autentificare, schimbare a directorului de lucru etc.), cât și cel de conexiune de date (transmiterea datelor) – termeni comuni întâlniți la un protocol FTP. În contextul acestei aplicații, accentul este pus pe integritatea datelor transferate și nu neapărat pe viteză, de aici și motivația alegerii protocolului TCP.

În ideea transmiterii securizate a parolei folosite la autentificarea unui utilizator, propun utilizarea unei criptări SHA256 pentru trimiterea și stocarea acesteia în regia serverului. Folosirea unui astfel de mecanism este comun întâlnită în industrie, SHA256 fiind unul dintre cei mai întâlniți algoritmi de hashing.

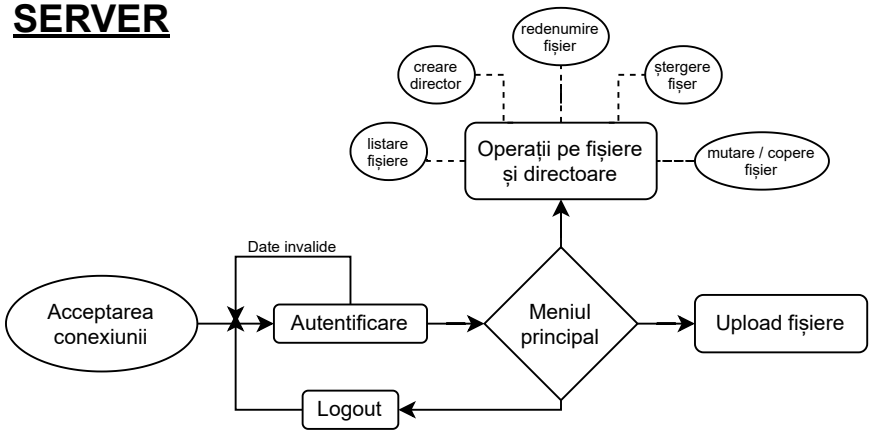
3 Arhitectura și structura aplicației

Aplicația va conecta și va permite comunicarea între client și server, reprezentând de fapt expeditorul, repsectiv destinatarul transferului de fișiere.

1. **Clientul** se va putea conecta la server și va putea trimite diferite instrucțiuni la linia de comandă, ce vor urma a fi validate și interpretate de server. Va indica și calea fișierelor destinate spre transfer.
2. **Serverul** este cel responsabil cu primirea și gestionarea comenzilor trimise de client. Putem împărți sarcinile acestuia în mai multe module:
   a) **Acceptarea conexiunilor** este primul modul expus clienților, care acceptă conexiunile acestora și crează un nou proces (asigură concurența aplicației).
   b) **Autentificarea** este una din operațiile de bază prin care clienții se pot diferența între ei și prin care ei pot primi acces la aplicație.
   Mecanismul de autentificare constă în selectarea utilizatorului, urmată de trimiterea securizată a parolei. Trimiterea repetată a unor date eronate va genera o blocare temporară a utilizatorului și revocarea posibilității de autentificare pe o perioadă determinată de timp.
   c) **Meniul principal** este modulul central al aplicației, accesat de client imediat după autentificare. De aici, clientul poate oferi diverse comenzi ce vor fi interpretate și redirecționate spre modulul care gestionează efectiv operația trimisă.
   Prin urmare, acest modul reprezintă o stare de așteptare a noilor comenzi primite de la client, care le validează și le redirecționează spre modulul potrivit (cel care va realiza acțiunea dorită).
   d) **Operațiile pe directoare și fișiere** sunt integrate într-un modul care gestionează efectiv comenzile trimise de client. Printre operațiile disponibile se numără: listarea fișierelor curent stocate, crearea directoarelor și redenumirea, mutarea, ștergerea și copierea fișierelor asociate utilizatorului logat.
   e) **Uploadul fișierelor**, probabil modulul principal al acestei aplicații, se ocupă de încărcarea fișierelor indicate de client în spațiul de stocare selectat.

f) **Logout** va deconecta utilizatorul curent autentificat, revocând accesul la comenzile meniului principal.

Putem defini mai bine structura serverului prin următoarea diagramă:



4 Aspecte de implementare

1. **Transmiterea securizată a parolei la autentificare**

   Este important ca parola introdusă de către client să fie trimisă și stocată într-o formă securizată, ea fiind vulnerabilă în cazul în care o sursă externă are acces la socket.

   Mecanismul de securizare gândit are la bază criptarea SHA-256 făcută la nivel de client, transmisă apoi spre server. Acesta din urmă va stoca doar hash-ul parolei și tot cu el va compara (la autentificare) potrivirea.

   Procedura de criptare cu acest algoritm va implementa următorul pseudocod:

```
/* CLIENT */
read(stdin, username);
send(to_server, username);

read(from_server, status);
if status == enter_passwd_status
    read(stdin, password);
    clientHash = SHA256(password);
    send(to_server, clientHash);
```

```
/* SERVER */
read(from_client, username);
send(to_client, enter_passwd_status);

//read client hash value and compare to stored user's password
userPasswdHash = getUserPassword(username);
read(from_client, clientHash);
if(clientHash == userPasswdHash)
    login();
else
    deny();
```

   O implementare a funcției SHA256 poate fi găsită la următorul link GitHub.

2. **Protocolul TCP - implementare în server și client**

   Crearea socket-ului TCP pe server (pseudocod), fără partea de listen:

```
/* SERVER */
int sd;
```

```
struct sockaddr_in server;

if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("Eroare la creare socket.\n");
    return errno;
}

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

if (bind(sd, (struct sockaddr*) &server, sizeof (struct sockaddr)) == -1)
{
    perror("Eroare la atasare socket (bind).\n");
    return errno;
}
```

Pseudocod de conectare la socket din client:

```
/* CLIENT */
int sd;
struct sockaddr_in server;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
{
  perror ("Eroare la crearea socketului.\n");
  return errno;
}

server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr("127.0.0.1");
server.sin_port = htons(PORT);

if (connect (sd, (struct sockaddr *) &server,sizeof (struct sockaddr)) == -1)
{
  perror ("Eroare la connectarea la socket.\n");
  return errno;
}
```

3. **Gestionarea erorilor**

   În vederea gestionării erorilor cauzate de defecțiuni interne, sau de un stream de date eronate/invalide trimise de client, este important să afișăm mesaje de eroare adecvate și să tratăm corect toate tipurile de input.

   Astfel, aplicația face verificări la toate operațiile pe care le realizează, începând cu conectarea la socket și până la erori de acces la fișiere. Pentru cazul în care utilizatorul încearcă autentificarea și trimite o parolă invalidă de mai multe ori la rând, el va fi blocat temporar și va fi deconectat de la socket (se previn astfel atacuri de tip brute-force pentru ghicirea parolei asociate unui username).

   Un alt caz important de tratat este acela când clientul se deconectează de la server în mijlocul transferului de fișiere. În această situație, serverul anulează transferul și termină procesul asociat anterior conexiunii cu acel client.

5 Concluzii

În opinia mea, aplicația ar putea beneficia de noi funcționalități ce ar îmbunătăți experiența utilizatorilor și le-ar oferi mai multă libertate în aplicație. Spre exemplu, pe viitor, se poate adăuga opțiunea de a descărca fișierele încărcate anterior de un client.

Împreună cu această funcționalitate, s-ar putea adăuga și un sistem de partajare al fișierelor, oferind clienților posbilitatea de a stoca fișiere într-un spațiu comun. O variațiune pentru acest feature ar putea reprezenta acordarea permisiunilor de citire și de editare altor utilizatori înscriși, ei având acces strict asupra anumitor fișiere din spațiul de stocare al altuia.

De asemenea, implementarea pe viitor a unei interfețe grafice ar putea ajuta la ridicarea gradului de intuitivitate asupra folosirii și identificării tuturor funcționalităților disponibile.

6 Referințe bibliografice

În realizarea acestui material, au fost extrase informații din următoarele surse:
https://www.programmingalgorithms.com/algorithm/sha256/c/
https://github.com/B-Con/crypto-algorithms/blob/master/sha256.c
https://www.investopedia.com/terms/f/ftp-file-transfer-protocol.asp
https://stackoverflow.com/questions/23119615/coding-ftp-service-over-tcp-in-c-code
https://linuxhint.com/catch-socket-errors-c/