Documentatie Proiect

Maciuc Mihai

January 6, 2025

1 Introduction

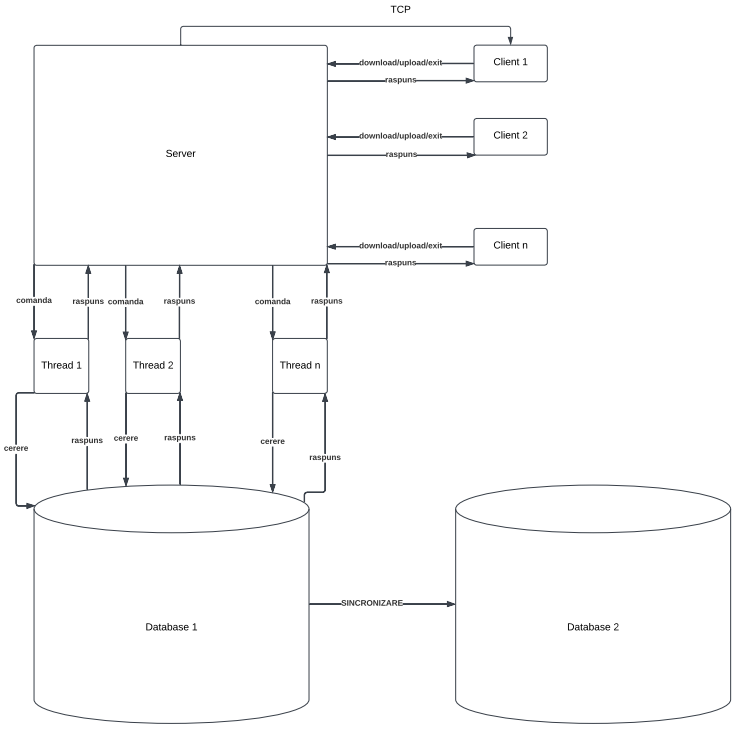
Proiectul ales este Personal Cloud Storage (A).El reprezintă implementarea unei soluții de stocare în cloud personală, sigură și accesibilă, care permite utilizatorilor să stocheze, acceseze și partajeze fișiere între dispozitive. Soluția include redundanță pentru protecție împotriva pierderii datelor și o interfață intuitivă pentru utilizator. Aplicatia este scrisa in limbajul C++ si foloseste librariile externe sqlite3 si pthread.

2 Tehnologii Aplicate

Una din tehnologiile aplicate în acest proiect este un server concurent TCP pentru a ne asigura că nu pierdem pachete sau corupem inoformația transmisă. Fiind un spațiu de stocare în cloud, fiabilitatea și siguranța datelor este mai importantă decat viteza de transmisie. Așadar am implementat un sistem de socket-uri TCP care prin procesul handshake face comunicarea să fie sigură între client și server. Pentru a rezolva problemele legate de concurentă vom folosi thread-uri dedicate fiecărui client. Fișierele vor fi stocate într-o bază de date. Ca în orice spațiu de stocare, integritatea datelor este deosebit de importantă așa că vom implementa doua baze de date. Una în care vom efectua operațiile de citire și scriere, iar a doua va fi o opțiune de backup în cazul în care prima baza de date nu răspunde.

3 Structura Aplicației

Aplicația este structurată în două componente: un server care gestionează fișierele și cererile și un client care interacționează cu serverul pentru a încărca sau descărca fișiere. Serverul va avea trei componente principale: prima componenta de operare a conexiunilor (sockets si threads),a doua de efectuare a operatiilor (download/upload/delete/list/exit) și ultima, de operare a bazelor de date. Cele două baze de date (implementate cu ajutorul librariei sqlite) vor fi structurate astfel: o bază principală în care vor avea loc operațiile și încă una care va face o sincronizare periodică cu prima. Clientul va manageria conexiunea cu serverul și va avea o interfață usor de utilizat.



4 Aspecte de Implementare

Structura TCP

Serverul creează un socket după care setează informațiile de conectare folosind `sockaddr_in`, care specifică:

- `sin_family = AF_INET`: Protocolul de rețea folosit este IPv4.
- `sin_addr.s_addr = htonl(INADDR_ANY)`: Permite serverului să asculte pe orice adresă IP disponibilă.
- `sin_port = htons(port)`: Setează portul de ascultare.

`bind()` leagă socket-ul la adresa și portul specificat, iar `listen()` pune serverul într-o stare de ascultare, așteptând conexiuni de la clienți.

Când un client se conectează, serverul folosește `accept()` pentru a stabili o conexiune cu acesta și pentru a obține un nou thread dedicat clientului. Pe fiecare thread, serverul citește comenzi de la client (folosind `read()`) și le procesează în funcție de comanda primită: upload, download, delete sau list. În fiecare caz, serverul returnează un mesaj corespunzător.

Clientul creează un socket TCP folosind aceeași funcție `socket()`, dar cu adresa IP a serverului și portul 8080, după care folosim functia `connect()` pentru a stabili o conexiune cu serverul. Clientul afișează un meniu cu opțiuni pentru încărcarea, descărcarea, ștergerea sau listarea fișierelor. Utilizatorul poate alege acțiunea dorită. În funcție de alegerea utilizatorului, clientul trimite un mesaj la server prin socket, folosind funcția `write()`. Mesajele trimise sunt de tipul:

- `UPLOAD <nume_fisier>`
- `DOWNLOAD <nume_fisier>`
- `DELETE <nume_fisier>`
- `LIST`

Comenzile implementate:

UPLOAD `<filename>`**:** Clientul trimite fișierul specificat către server iar serverul se ocupa de incarcarea in baza de date.

Listing 1: functie Upload

```
char fileName[256];
sscanf(buffer + 7, "%s", fileName);
// citesc fisierul primit de la client
// fac pathul pentru al incarca in bd
 string filePath = "./files/";
filePath += fileName;

// daca exista deja in bd ignor comanda
ifstream existFile(filePath,ios::binary);
 if(existFile.is_open())
{
     cout<<"Fisierul exista deja. Upload ignorat\n";
      existFile.close();
      return;
}
ofstream outFile(filePath, ios::binary);
if (outFile)
{
char fileBuffer[dim];
// iau secvente de file data
```

```
int bytesReceived;
 while (true)
{
     memset(fileBuffer, 0, dim);
    bytesReceived = read(client_sd, fileBuffer, dim);

    if (bytesReceived <= 0 || (bytesReceived == dim
    && memcmp(fileBuffer, "\0", dim) == 0))
        break;
//verific daca am citit ceva sau daca nu am avut o eroare
     outFile.write(fileBuffer, bytesReceived);
    if (bytesReceived < dim)
        break;
/*daca e mai mic decat dim
inseamna ca am primit ultima parte a fisierului*/
}
outFile.close();
//am terminat upload pe disc
database.uploadFile(fileName, filePath);
//acum incarc in bd
cout << "fisierul " << fileName << " UPLOADAT " << endl;
```

DOWNLOAD <filename>: Clientul solicită descărcarea unui fișier de pe server.

Listing 2: functie Download

```
char fileName[256];
sscanf(buffer + 9, "%s", fileName);
//citesc fisierul primit de la client

string filePath = database.getFilePath(fileName);
//iau din bd calea unde este salvat fisierul pe disc
    if (!filePath.empty())
    {
    ifstream inFile(filePath, ios::binary);

    if (inFile)
    {
        char fileBuffer[dim];
        while (inFile.read(fileBuffer, dim))
        {
        write(client_sd, fileBuffer, inFile.gcount());
        }

         if(inFile.gcount()>0)
        write(client_sd,fileBuffer,inFile.gcount());
```

```
        //trimit un buffer gol ca sa termine transferu
        memset(fileBuffer,0,dim);
        write(client_sd,fileBuffer,dim);

        inFile.close();
        cout << "Fisierul trimis:" << fileName << endl;
    }
```

DELETE `<filename>`**:** Clientul cere ștergerea unui fișier de pe server.

Listing 3: functie Delete

```
char fileName[256];
sscanf(buffer+7,"%s",fileName);

string filePath=database.getFilePath(fileName);
if(!filePath.empty())
{

    if(remove(filePath.c_str())==0)
    //0 pt succesful -1 la probleme cu fisierul de pe disc
    {
        database.deleteFile(fileName);
        //sterge fisierul din bd
        cout << "fisier sters" << endl;
    }
    else
        cerr<<"eroare la stergerea fisierului\n";
}
else
        cerr<<"Fisierul nu apare in baza de date\n";
```

In mod similar am procedat si pentru functiile de LIST si EXIT.

LIST: Clientul solicită lista fișierelor disponibile pe server.

Listing 4: functie List

```
vector<string> files=database.listFIles();
string response;
if(files.empty())
response="Nu exista fisiere in spatiul dumneavoastra:)";
else{
```

```
        response="Fisiere disponibile:\n";
        for(auto file: files)
         response+=file+'\n';
        }
    response+="END_OF_LIST";
    /*am folosit stringul "END_OF_LIST"
    ca sa semnalez sf fisierelor in vector*/
    write(client_sd,response.c_str(),response.size());
    cout << "fisierele sunt listate" << endl;
```

EXIT: Deconectarea clientului.

Serverul folosește threaduri pentru a rezolva problema concurentei. Am folosit biblioteca "thread" din C++ pentru a creea thread-uri care procesează cererile fiecărui client. Modelul este simplu: pentru fiecare conexiune acceptată, este creat un thread nou care gestionează comunicarea cu clientul respectiv.

Listing 5: Threads

```
cout<<"Client nou: "<< client_sd <<endl;
thread(handle_client,client_sd).detach();
```

5 Concluzii

Pe langa aspectul educativ al proiectului, consider ca este un proiect cu o puternică implicație practică. Obiectivele proiectului ating printre altele, învațarea implementarii unui server TCP, manipularea bazelor de date si implementarea concurentei. Potentiale imbunatatiri aduse proiectului ar putea fi o functie de login, prin care să facilitam multiple zone independente de cloud pentru fiecare user, o metoda de compresie a fisierelor sau alte elemente de UI cum ar fi dark mode,interfata grafica sau butoane interactive.

6 Referințe Bibliografice

In realizarea diagramei am folosit:

1. https://lucid.app/

In probleme legate de concurenta/tcp am folosit:

1. https://www.geeksforgeeks.org/multithreading-in-cpp/
2. https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
3. https://stackoverflow.com/questions/5592747/bind-error-while-recreating-socket
4. https://stackoverflow.com/questions/73941101/socket-programming-issue-c
5. https://ramonnastase.ro/blog/ce-este-protocolul-tcp-si-cum-functioneaza/

In probleme cu baza de date am folosit:

1. https://stackoverflow.com/questions/26897132/how-to-save-a-file-path-at-database
2. https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm
3. https://www.tutorialspoint.com/sqlite/sqlite_installation.htm

In incercari pentru criptare am folosit:

1. https://stackoverflow.com/questions/49313677/need-assistance-with-vigenere-cipher-prog
2. https://stackoverflow.com/questions/70755464/tls-1-2-implementation-in-c-windows-appli
3. https://www.researchgate.net/figure/Encryption-and-decryption-Process-of-Vigenere-ciph fig1_287205397

Altele:

1. https://www.w3schools.com/cpp/cpp_exceptions.asp
2. https://www.geeksforgeeks.org/casting-operators-in-cpp/