Part2Part (A)

Dorneanu Dragoş-Andrei, grupa 2B2
dragos.a.dorneanu@gmail.com

December 23, 2016

1 Abstractizare

Să se conceapă o aplicaţie de tipul peer-to-peer pentru partajarea fişierelor.

Programele vor folosi socket-uri pentru comunicaţie, putând fi rulate pe maşini diferite astfel : programul client poate fi rulat de către orice utilizator, iar programul server poate fi rulat doar de autor, care este şi responsabil cu pornirea/oprirea serviciului. Programul server trebuie să fie capabil să serveasca simultan mai multe cereri de transfer provenite de la "colegii" săi din retea. Fişierele dorite vor putea fi căutate - eventual folosind expresii regulate - conform unor criterii (nume, tip, lungime etc.) în cadrul "reţelei".

2 Cuvinte Cheie

Server, Client, TCP, UDP, Bază de Date, Comunicare, Interogare, Peer, Share, Download, Fişier, User, Seeder, Tabel, Concurent, Thread, Socket, Hash, Criptare, Servent, IP, Port, Peer-2-Peer

3 Introducere

Tema proiectului ales constă în implementarea unui program asemănător unui Client Torrent. Aplicaţia trebuie să suporte opţiunile de căutare de fişiere după nume, tip şi dimensiune, transfer de fişiere între utilizatori, descărcarea unui fişier de la mai mulţi utilizatori în acelaşi timp(dacă există utilizatori care dispun de acelaşi fişier) şi posibilitatea ca toţi user-ii să aibă acces concomitent la toate serviciile disponibile.

Arhitectura cerută este una ce implică o aplicaţie client, a cărei executabil este diponibil tuturor utilizatorilor, şi o aplicaţie server, disponibilă doar programatorilor ce au lucrat la dezvoltarea aplicaţiei.

4 Tehnologii Utilizate

4.1 TCP

TCP(Transmission Control Protocol) este unul dintre principalele protocoale ale Internet-ului. Este un protocol folosit în aplicaţii ce au nevoie de confirmarea primirii datelor, păstrarea ordinii la primire şi a integrităţii acestora.

Aceste caracteristici ale protocolului TCP sunt folosite pentru comunicarea între Server şi Client. Păstrarea integrităţii şi a ordinii informaţiilor trimise de Server către Client este de o importanţă enormă în cadrul acestei aplicaţii. Dacă arhitectura nu ar consta într-o comunicare sigură între Server şi Client, atunci informaţiile care trebuie să fie primite şi apoi folosite de Client pot fi pierdute sau alterate, lucru ce va împiedica buna funcţionare a paşilor următori din procesul de file sharing.

Comunicarea propriu-zisă dintre Client şi Server se realizează cu ajutorul primitivelor read şi write(din limbajul C) aplicate asupra unor socket-uri.

4.2 UDP

UDP(User Datagram Protocol) este un protocol de comunicare pentru calculatoare ce aparţine nivelului Transport al modelului standard OSI(Open Systems Interconnection). Principalele caracteristici ale acestui protocol sunt lipsa conexiunii dintre participanţii la transferul de date, viteza şi lipsa siguranţei primirii şi a siguranţei integrităţii informaţiilor.

Viteza de transfer a informaţiei este proprietatea care a făcut protocolul UDP să fie eligibil pentru acest tip de aplicaţie. Informaţiile trimise de la un client la altul trebuie să ajungă într-un timp cât mai scurt, iar în cazul în care nu sunt primite sau sunt alterate se vor cere dinnou. Datorită vitezei de transfer oferite de UDP, alegerea acestuia în implementarea transferului de fişiere de la mai mulţi utilizatori este mai eficientă decat cea în care transferul este realizat prin TCP(mai încet datorită verificărilor făcute la primirea informaţiei şi a stabilirii conexiunii).

Comunicarea propriu-zisă dintre Clienţi se va face cu ajutorul primitivelor sendto şi recvfrom(din limbajul C) care vor trimite/primi datagrame ce conţin părţi din informaţia dorită.

4.3 MySQL Database

MySQL este un sistem de gestiune a bazelor de date relaţionale multi user şi multithreaded. Poate fi utilizat împreuna cu limbaje de programare precum C/C++, Java, PHP etc.

Posibila cantitate mare de informaţii la care trebuie să aibă acces Serverul, nevoia unui timp de răspuns bun şi necesitatea de tratare a cererilor fiecărui client într-o manieră concurentă face o bază de date MySQL un candidat bun pentru a fi desemnat structură de date pentru memorarea informaţiilor necesare Serverului. În plus, în cazul unei erori de server şi închiderii acestuia, informaţia va fi păstrata intactă în baza de date. În cazul altor structuri de date, cum ar fi tabelele de dispersie(hashtable), informaţiile ar fi pierdute.

Baza de date va fi stocată local, pe aceeşi maşină pe care ruleaza şi aplicaţia Server.

4.4 OpenSSL

OpenSSL este o librărie software folosită pentru aplicaţii care au nevoie să-şi securizeze comunicarea prin reţea împotriva eventualelor atacuri.

Pentru a trimite username-ul şi în special parola într-o manieră sigură pentru a fi verificate de Server cu baza de date, este nevoie de o criptare a acestor informaţii(din motive de securitate), fapt ce se poate realiza prin folosirea algoritmului RSA a cărui implementare este prezentă în biblioteca OpenSSL. În plus, în baza de date nu este recomandată memorarea parolei exacte ci a unei valori hash a acesteia. Pentru a oferi un hash bun se pot folosi algoritmi precum MD5 sau SHA256 ce se găsesc deasemenea în librăria OpenSSL.

4.5 C/C++

Atât Serverul cât şi Clientul aplicaţiei sunt scrise în limbajele de programare C şi C++. Pentru tratarea concurentă a clienţilor s-a ales utilizarea structurii pthread din limbajul C iar pentru operaţiile de input/output s-au folosit librariile iostream si fstream din C++.

5 Arhitectura Aplicaţiei

Având în vedere cerinţele aplicaţiei, s-a decis alegerea unei arhitecturi descentralizat hibride. Cu alte cuvinte, arhitectura aplicaţiei va consta într-un Super Server care va cunoaste toţi utilizatorii şi va şti fişierele disponibile pentru seeding ale acestora. Pentru memorarea unui volum ridicat de informaţii s-a ales o bază de date MySQL. Clienţii care fac cereri Super Serverului, dupa ce primesc informaţii despre utilizatorii(peer) ce au fişierul dorit, se deconectează de la acesta şi iniţiază comunicare cu toţi utilizatorii primiţi drept răspuns. Astfel se observă necesitatea ca aplicaţia client să devină şi o aplicaţie server pentru a putea primi şi rezolva cerinţe de transfer de la alţi clienţi/utilizatori. Deci Clientul va juca rol de Servent.

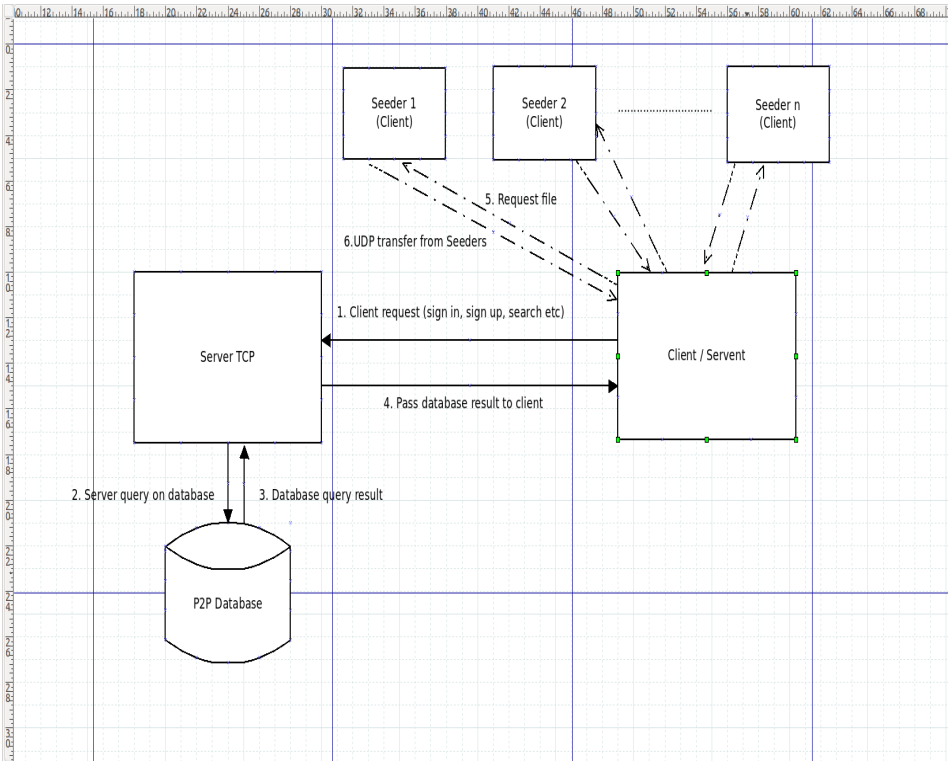

Figure 1: Application architecture

6 Detalii de Implementare

6.1 Realizarea download-ului de la mai mulţi clienţi

1) se va împărţi fişierul în bucăţi de dimensiune fileSize / peerNumber bytes

2) pentru fiecare peer găsit se lansează un thread care va cunoaste un offset de start şi un offset de sfârşit al unei părţi din fişier pe care utilizatorul vrea să o primească

3) clientul peer va trimite informaţii din fişierul căutat deschis în format binar începand de la offset-ul de start primit până la offset-ul de sfârşit primit

!!!) dacă la un moment dat informaţia trimisă nu corespunde cu cea primită(au valori hash diferite) sau nu s-a mai primit nimic de la peer(şi transferul bucăţii de fişier nu se terminase) atunci se marchează într-o listă faptul că nu am primit informaţia începand de la offset-ul curent pana la offset-ul de sfârşit şi aceasta va fi cerută dinnou la o altă iteraţie a procesului de download

6.2 Server Concurent

Pentru fiecare client acceptat de server se crează un pthread cu rol de rezolvare a cerintelor acestuia. Fiecare pthread va primi în lista de argumente un pointer către baza de date a Serverului, un socket descriptor pentru Client şi structura cu informaţii despre Client.

```
bool acceptClient(int &client, int &socketDescriptor, struct sockaddr_in * from)
{
    unsigned int size = sizeof(from);
    if((client = accept(socketDescriptor, (struct sockaddr *)from, &size)) == -1)
    {
        perror("Accept error");
        return false;
    }
    return true;
}
```

```
while(true)
{
    int clientSocket;
    struct sockaddr_in from;
    pthread_t thread;
    cout << "Waiting for a client to connect..." << endl;
    if(!acceptClient(clientSocket, socketDescriptor, &from))
        continue;
    DatabaseQueryParameters threadParameters(databaseConnection, &clientSocket, from);
    pthread_create(&thread, NULL, solveRequest, (void *)&threadParameters);
}
```

Figure 2: Accept client procedure

6.3 Structura Bazei de Date

Structura unei baze de date afecteaza foarte mult rularea unei aplicaţii ce o foloseste. Aceasta trebuie gândită cât mai optim atât ca memorie utilizată cât şi ca timp de execuţie al unei interogări asupra ei. Structura bazei de date utilizate pentru stocarea informaţiilor necesare rulării corecte a programului este reprezentată în urmatoarea diagrama :

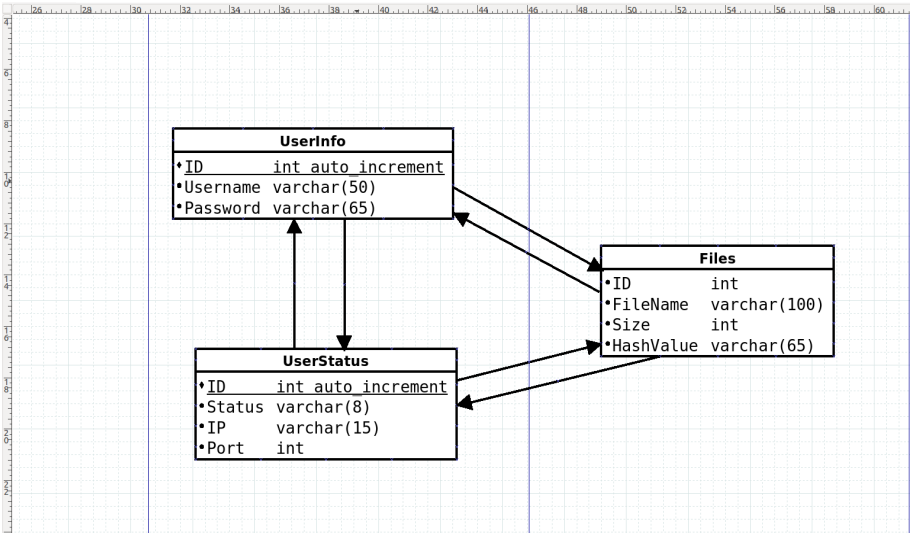

Figure 3: Database structure

6.4 Fişierul de configurare al directorului de download

Aplicaţia client funcţionează împreuna cu fişierul de configurare "path.conf". Acesta este creat în timpul operaţiei de sign up şi este important pentru buna funcţionare a programului. Conţinutul fişierului "path.conf" este o cale absolută către directorul de download/seeding al user-ului.

7 Concluzii

Aplicaţia prezentată este concepută pentru a suporta un număr relativ mare de clienţi şi pentru a satisface toate cererile acestora într-o manieră concurentă. Arhitectura a fost gândită astfel încât utilizatorul să nu trebuiască să aştepte timp îndelungat spre a primi răspunsuri la căutarea sa sau pentru a primi fişiere.

O îmbunătăţire care s-ar putea aduce structurii/implementării acestui program poate fi găsirea unei modalităţi mai bune de transfer al bucăţilor de fişier de la clienţii peer cu care comunică utilizatorul ce face download.

8 Bibliografie

1. http://profs.info.uaic.ro/ adria/teach/courses/net/cursullaboratorul.php
2. http://adambard.com/blog/3-wrong-ways-to-store-a-password/
3. http://cs.berry.edu/ nhamid/p2p/
4. ftp://ftp.springer.de/pub/tex/latex/llncs/latex2e/instruct/authors/authors.pdf