Aplicaţia de Editare Simultană a Fişierelor Text SharedPad

Denise Goldan

Facultatea de Informatică Iaşi, Universitatea "Alexandru Ioan Cuza"
denise.goldan@info.uaic.ro

Rezumat Prezenta lucrare descrie o aplicaţie de tip client/server ce pune la dispoziţia utilizatorilor o modalitate de a edita fişiere text. Un utilizator logat în aplicaţie are opţiunea de a alege să formeze o pereche alături de un alt utilizator logat, în scopul editării concurente a unui fişier text la alegere. Există şi opţiunea de a lucra pe cont propriu, chiar şi în cazul în care serverul nu poate prelua cererile clienţilor, fiind împiedicat de o eroare. Aplicaţia permite editarea fişierelor de pe staţia de lucru proprie şi salvarea modificărilor. Este implementat un protocol de comunicare pentru operarea concurentă asupra fişierelor, protocol ce asigură consistenţa datelor.

Cuvinte cheie TCP, editare concurentă, interfaţă grafică, client/server, fişiere text, informaţii

1 Introducere

Editarea concurentă a unui fişier de către mai mulţi utilizatori este o problemă des întâlnită, pentru care s-au propus multe soluţii.

Una dintre acestea este dată de utilizarea unui algoritm bazat pe token-uri: clientul cere un token de editare serverului, aşteaptă până când primeşte token-ul, editează documentul până când token-ul expiră. Acest mecanism de lock pe fişier poate afecta, însă, experienţa celorlalţi participanţi la editare, care nu au token-ul respectiv.

Altă variantă de rezolvare a conflictelor ridicate de această problemă este dată de transformările operaţionale [1]. Reprezintă o tehnologie implementată, pentru prima dată, în 1989 de către C.Ellis şi S. Gibbs în cadrul GROVE (GRoup Outline Viewing Edit). Au urmat ani de cercetare şi eforturi substanţiale de îmbunătăţire a tehnologiei. Aceasta oferă mecanisme ce nu presupun blocarea pe fişier, oferind un timp de răspuns bun, neafectat de latenţa Internetului. Drept urmare, este indicat în implementarea editoarelor de fişiere în contextul Web/Internet.

În modelul client/server, serverul gestionează modificările asupra documentului comun, pe care utilizatorii le realizează pe maşinile personale. El determină cum modificările făcute de un utilizator ar trebui să afecteze versiunea deţinută de el a fişierului, pentru ca, mai apoi, să transmită utilizatorilor ce participă la editare varianta corectă a fişierului.

2 Tehnologii utilizate

2.1 TCP

O noţiune cheie în cadrul acestei aplicaţii este informaţia şi nu se doreşte pierderea acesteia sub nicio formă. Din acest motiv se foloseşte de **TCP** (*Transmission Control Protocol*).

Acest protocol de comunicaţie asigură, prin cadrul unei conexiuni sigure şi prin controlul erorilor, faptul că modificările făcute de un utilizator ce partajează cu alt utilizator un fişier text, în scopul editării, sunt văzute de cel din urmă. Necesitatea ca informaţiile să fie nealterate în cadrul comunicării dintre procese este evidentă. Spre deosebire de protocolul **UDP** (*Uşer Datagram Protocol*), ce nu presupune mecanisme de validare a corectitudinii datelor, **TCP** oferă siguranţă în transmisia datelor.

2.2 SPP

La **nivelul aplicaţie**, există un protocol de comunicaţie special definit pentru tratarea operării concurente asupra fişierelor, protocol ce poartă numele de SharedPad Protocol.

Clientul şi serverul vor folosi mesaje de tip json [4]. Acest format este caracterizat, în primul rând, de extensibilitate, întrucât se pot face modificari- cum ar fi adăugarea unor câmpuri cheie-valoare- fără a afecta codul existent. De asemenea, este uşor de folosit şi există numeroase biblioteci pentru generatoare şi parsere în C++.

Protocolul SPP defineşte un set de reguli printre care şi una privitoare la mesajele ce vor fi schimbate între client şi server. Acestea vor respecta formatul standardizat, aşa cum s-a menţionat în paragraful anterior, şi specializat. Clientul va trimite serverului mesaje de tip "cerere" iar serverul va trimite mesaje de tip "răspuns". Acestea vor fi prefixate de lungimea lor şi urmate de un delimitator.

O "cerere" va conţine câmpurile "comandă" şi "argumente", iar un "răspuns" va conţine "cod", "descriere a codului", pe lângă alte câmpuri specializate pe tipul răspunsului.

2.3 Json

Mesajele pe care serverul şi clientul le vor schimba vor fi de tip json, unde json reprezintă o abordare simplă de serializare a unui obiect bazată pe sintaxa JavaScript. Aceste mesaje vor fi prefixate de lungimea lor. Această formulă de transmitere a mesajelor este legată de protocolul **SPP**.

Avantajele date de acesta sunt:

- oferă o modalitate automată de a serializa/deserializa obiecte JavaScript, folosind linii minime de cod
- oferă un format concis, datorită abordării bazate pe perechea nume-valoare
- există parsere şi generatoare pentru extragerea datelor dintr-un text în format json, respectiv pentru a produce text în format json
- API-uri simple, disponibile pentru un număr mare de limbaje de programare, printre care şi C++.

2.4 Fire de execuţie POSIX

Ca şi limbaj de programare, atât serverul, cât şi clientul sunt scrişi în **C++**. Mai mult decât atât, ambii actanţi se vor folosi fire de execuţie (POSIX threads[6]). Spre deosebire de procese, firele de execuţie au un mare avantaj în sincronizare, întrucât ele partajează în mod implicit majoritatea resurselor unui proces şi modificarea unei astfel de resurse dintr-un

fir este vizibilă instantaneu şi în celelalte fire. Prezintă şi alte plusuri, cum ar fi faptul că distrugerea/crearea unui fir de execuţie durează mai puţin decât în cazul unui proces. Se pot dovedi utile în multe situaţii, de exemplu, pentru a îmbunătăţi timpul de răspuns al aplicaţiilor cu interfeţe grafice (cum este cazul clientului din această aplicaţie, unde fiecare inserare sau ștergere a unui caracter sau a unui șir de caractere determină o comandă ce trebuie executată), unde prelucrările CPU-intensive se fac de obicei într-un fir de execuţie diferit de cel care afişează interfaţa.

2.5 Qt

Clientul va avea în plus, faţă de server, o interfaţă grafică simplă, atractivă, uşor de înţeles, implementată cu ajutorul **Qt**[5]. Astfel, interacţiunea utilizatorului cu aplicaţia va fi simplificată.

Interfața grafică va cuprinde, printre altele:

- o zonă destinată autentificării utilizatorilor (Figura 1)
- o zonă destinată editării fişierului text (Figura 2)
- opţiunea de a accesa un fişier de pe staţia proprie
- opţiunea de a salva modificările făcute în editor pe staţia proprie
- opţiunea de a invita un colaborator pentru editarea unui fişier
- opţiunea de a părăsi colaborarea
- opţiunea de părăsire a aplicaţiei cu sau fără delogare

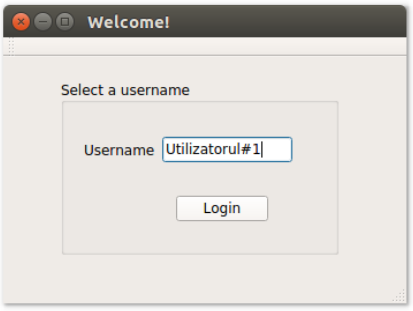

Figura 1. Utilizatorul doreşte să se autentifice

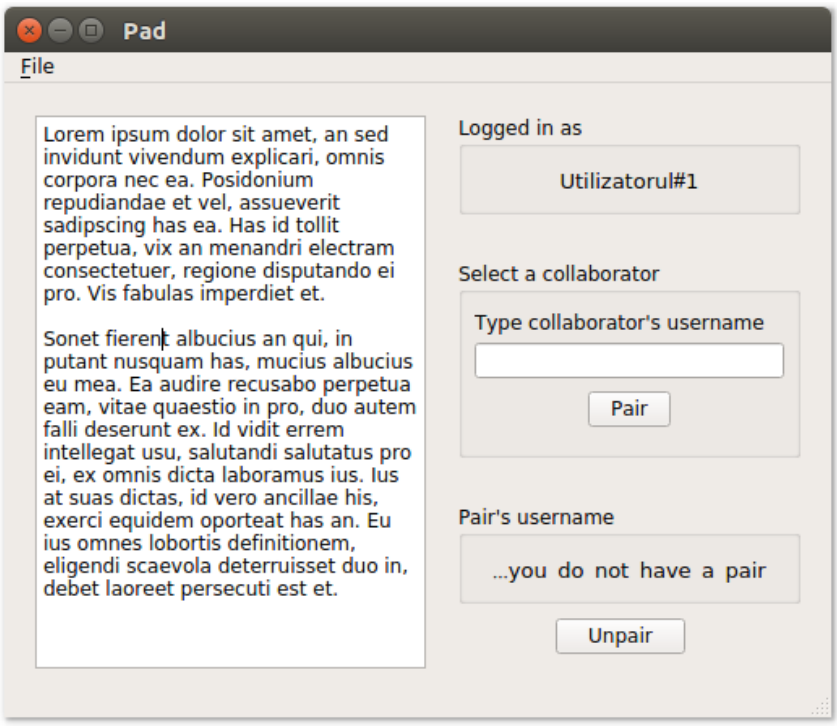

Figura 2. Fereastră dedicată editării fişierelor

3 Arhitectura aplicaţiei

3.1 Server

Serverul va rula fără oprire, pentru a permite ultilizatorilor accesul la aplicaţie şi funcţionalităţile ei. Serverul are rol în înregistrarea utilizatorilor, precum şi în gestionarea perechilor de editare colaborativă a fişierelor. De semenea, asigură accesul utilizatorilor la date şi sincronizarea datelor referitoare la editarea unui fişier. Legat de accesul la fişiere, va permite utilizatorilor să editeze şi salveze local fişiere de tip text. El va stoca temporar aceste fişiere.

3.2 Client

Clientul este reprezentat de o interfaţă grafică prin intermediul căreia, în primă fază, utilizatorul se va înregistra (Figura 3). Poate apela la funcţionalităţile aplicaţiei pe cont propriu sau poate invita un colaborator (Figura 4). Fişierele vor putea fi salvate pe maşina de lucru proprie de către utilizatori.

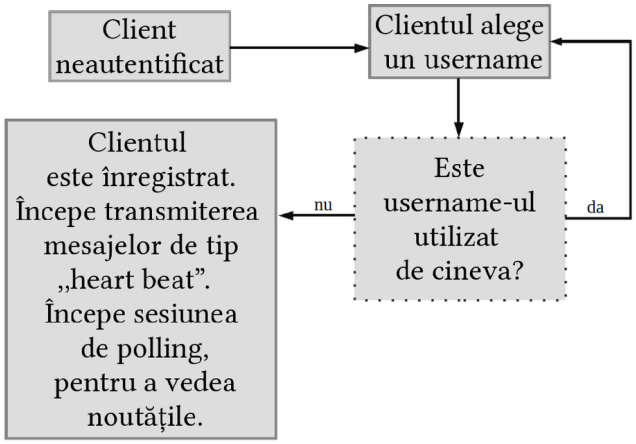

Figura 3. Un client dorește să se autentifice în cadrul aplicației

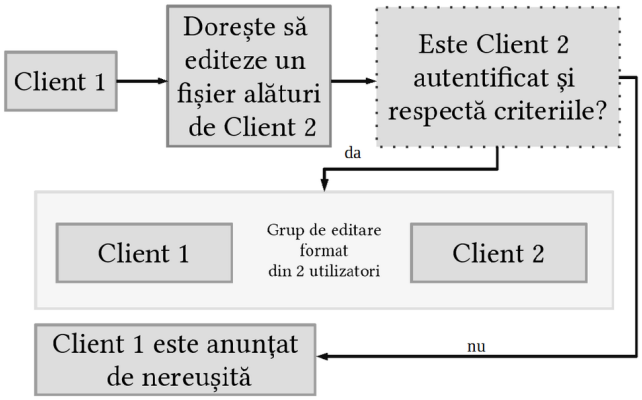

Figura 4. Client 1 dorește să editeze alături de Client 2

3.3 Accesul concurent la fişierele text

Accesul concurent la fişier duce la editarea simultană a acestuia, precum şi la menţinerea consistenţei şi corectitudinii datelor. Avantajul este dat de eliminarea timpilor îndelungaţi de aşteptare în vederea soluţionării cererilor numeroase de editare a fişierului. Se previne inconsistenţa datelor.

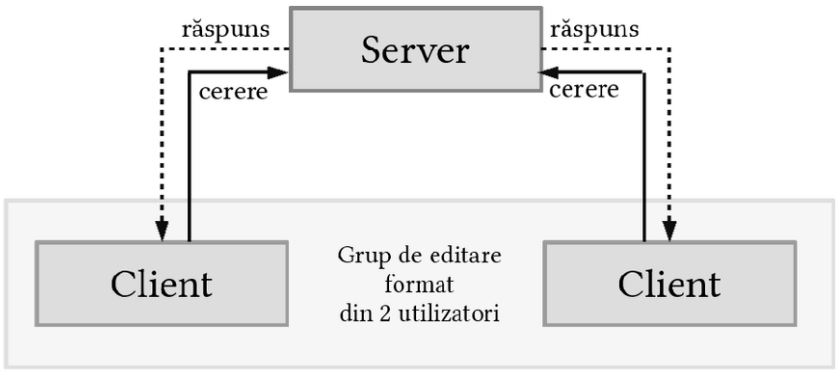

Figura 5. Gestiunea operaţiilor de editare

4 Detalii de implementare

Comunicarea dintre client şi server are la bază socket-urile BSD. Odată pornit, serverul permite clienţilor să se folosească de funcţionalităţile aplicaţiei. Acesta intră în aşteptare şi, când sosesc clienţi, îi tratează simultan, cu ajutorul thread-urilor.

Serverul are o metodă care tratează toate cererile, indiferent de tipul lor- o secţiune centralizată de soluţionare a cererilor. Acest lucru nu există şi în client, unde cererile sunt specializate şi emise din diferite puncte.

```
void *Server::handleClient(void *client)
{
    ClientInformation *currentClient = (ClientInformation *) client;

    handleClient_logger->info("Reading the request length.");
    int jsonRequestLength = readJsonRequestLength(currentClient);
    if (jsonRequestLength == -1)
    {
        handleClient_logger->warn("jsonRequestLength == -1");
        return terminateThread();
    }

    char *jsonRequest = readJsonRequest(currentClient, jsonRequestLength);
    handleClient_logger->info("Reading the request. The request is: ");
    handleClient_logger->info(jsonRequest);
    if (jsonRequest == nullptr)
    {
        handleClient_logger->warn("jsonRequest == nullptr");
        free(jsonRequest);
        return terminateThread();
    }

    handleClient_logger->info("Parsing the request.");
```

```
    Document *jsonDocument = JsonRequestParser::parseJson(jsonRequest);
    if (jsonDocument == nullptr)
    {
        handleClient_logger->warn("jsonDocument == nullptr");
        free(jsonRequest);
        free(jsonDocument);
        return terminateThread();
    }

    handleClient_logger->info("Executing the request.");
    GenericResponse *message = executeGenericRequest(jsonDocument);

    handleClient_logger->info("Sending a response back. The response is: ");
    handleClient_logger->info(message->getCodeDescription());
    sendResponse(*message, currentClient->clientSocketFD);

    return terminateThread();
}
```

Protocolul de comunicare dintre cele două entităţi stabileşte un set riguros definit de paşi. Cât timp serverul rulează fără întrerupere, urmatoriul scenariu se repetă:

- clientul formulează o cerere
- cererea este verificată în cadrul clientului şi trimisă serverului
- serverul preia cererea
- cererea este verificată în cadrul serverului
- serverul soluţionează cererea
- cererea poate produce modificări la nivelul informaţiilor pe care le gestionează serverul
- serverul formulează un răspuns corespunzător cererii efectuate de client
- serverul trimite răspunsul clientului
- clientul verifică răspunsul primit de la server
- răspunsul poate produce modificări la nivelul informaţiilor deţinute de client

Următoarea secţiune va descrie o aplicaţie a acestor paşi:

Cum este tratată o cerere de autentificare:

Un utilizator alege să folosească aplicaţia, şi introduce un username la alegere.

Stringul introdus este verificat (în acest caz, nu se permite ca numele utilizatorului să fie necompletat). Clientul realizează o conexiune cu serverul şi îi transmite mesajul prin intermediul unui socket. Un thread al serverului preia cererea şi verifică integritatea. Se verifică formatul standardizat al cererii, aşteptându-se la un json prefixat de lungimea mesajului, urmată de un delimitator. Se verifică şi corespondenţa dintre comandă şi argumentele comenzii, precum şi câmpurile suplimentare din json. În cazul logării, este necesară existenţa câmpului "comandă", cu valoarea "login" şi a câmpului "username" aflat în blocul "arguments", cu valoarea specificată de client. Serverul verifică dacă mai există un utilizator logat cu acelaşi nume. Dacă mai există, serverul îi va trimite un mesaj prin care să anunţe clientul de faptul că numele ales este deja folosit. Dacă încă nu există un utilizator cu acel nume, serverul aprobă autentificarea clientului şi îl introduce în lista de utilizatori activi logaţi. În

acest caz, clientul primeşte un mesaj care să îi dea de înţeles că logarea a avut loc cu succes. Clientul primeşte mesajul din partea serverului şi execută o serie de verificări ale mesajului. Ca şi în cazul anterior, se aşteaptă ca mesajul să fie prefixat de lungimea sa. Răspunsul din partea serverului trebuie să conţină un cod şi o descriere a codului. Poate fi însoţit şi de alte câmpuri suplimentare, însă nu şi în cazul logării. Dacă autentificarea este o reuşită, utilizatorul va putea interacţiona cu fereastra de editare.

Secvență de cod din client, ce privește autentificarea: (Clientul are dreptul la un număr nelimitat de încercări.)

```
void LoginWindow::onLoginButtonPressed()
{
    QString username =  ui->usernameLineEdit->text();
    if (username.isEmpty())
    {
        QMessageBox::information(this, tr("Error message"),
                "Username can't be blank!");
    }
    else
    {
        Client * client = new Client();
        GenericResponse * responseFromServer =
                client->login(username.toStdString());
        switch(responseFromServer->getCode())
        {
        case LOGIN_FAILED_CODE :
        {
            QMessageBox::critical(this,"Login failed!",
                "The username you provided is already registered.");
            break;
        }
        case LOGIN_APPROVED_CODE :
        {
            notepadWindow = new NotepadWindow(this);
            notepadWindow->setUsername(username);
            notepadWindow->show();
            HeartBeatSender *sender = new HeartBeatSender();
            sender->setUsername(username);
            sender->sendUpdates();
            break;
        }
        case CONNECTION_FAILED_CODE:
        {
            QMessageBox msgBox(QMessageBox::Question,
                               tr("Server crashed"),
                               "How do you wish to proceed?",
                               QMessageBox::Yes | QMessageBox::No);

            msgBox.setButtonText(QMessageBox::Yes,tr("Edit offline"));
```

```
            msgBox.setButtonText(QMessageBox::No, tr("Exit␣application"));

            if(msgBox.exec() == QMessageBox::Yes)
            {
                notepadWindow = new NotepadWindow(this);
                notepadWindow->setUsername(username);
                notepadWindow->show();
            }
            else
            {
                exit(EXIT_SUCCESS);

            }
        }
        }
    }
    this->destroy();
}
```

Secvență de cod din server, ce privește autentificarea:

```
GenericResponse *Server::executeLoginRequest(const Document *document)
{
    string username = document->FindMember(ARGUMENTS)->
        value[USERNAME].GetString();
    if (loggedUsers->find(username) == loggedUsers->cend())
    {
        User *newUserInformation = new User();
        loggedUsers->insert
                (pair<string, User>(username, *newUserInformation));
        return SpecializedResponse::getLoginApprovedResponse();
    }
    else
    {
        return SpecializedResponse::getLoginFailedResponse();
    }
}
```

Tratarea cazului în care clientul suferă o eroare. Mesajele de HeartBeat:

În cadrul serverului este implementat un sistem de deconectare a utilizatorilor inactivi. Fiecare client are în spate un serviciu care se ocupă cu trimiterea regulată a unui mesaj, numit mesaj de HeartBeat, care cuprinde numele utilizatorului. Serverul tratează aceste cereri, actualizând timpul aferent ultimului mesaj de HeartBeat primit din partea utilizatorului respectiv. Dacă după o anumită perioadă serverul constată că un utilizator nu a mai trimis aceste mesaje, şterge din lista de utilizatori numele utilizatorului şi elimină şi perechea din care făcea parte (dacă era membru al vreunei perechi).

```
void *Server::handleDisconnecting(void *)
{
    handleDisconnecting_logger->info("Inside␣function␣handleDisconnecting.");
    while (true)
    {
```

```
        printLoggedUsers();
        printPairs();
        for (auto usersIT = loggedUsers->cbegin();
                  usersIT != loggedUsers->cend();
                  usersIT++)
        {
            timeval now; gettimeofday(&now, NULL);
            if (now.tv_sec - usersIT->second.getLastCheck().tv_sec > 10)
            {
                loggedUsers->erase(usersIT);
                removePairContainingUsername(usersIT->first.c_str());
                handleDisconnecting_logger->info
                        ("Server erased an inactive client.");
            }
        }
        handleDisconnecting_logger->info
                ("Server's disconnecting service will wait for 10 seconds.");
        sleep(10);
    }
}
```

Tratarea cazului în care serverul suferă o eroare:

În această situaţie, clientul este anunţat imediat despre faptul că serverul nu îi mai poate prelua cererile de colaborare (clientul se foloseste în acest scop de un serviciu care face polling la server pentru a afla noutăţi). Acestuia i se oferă opţiunea de a continua editarea fişierului pe cont propriu sau de a părăsi aplicaţia.

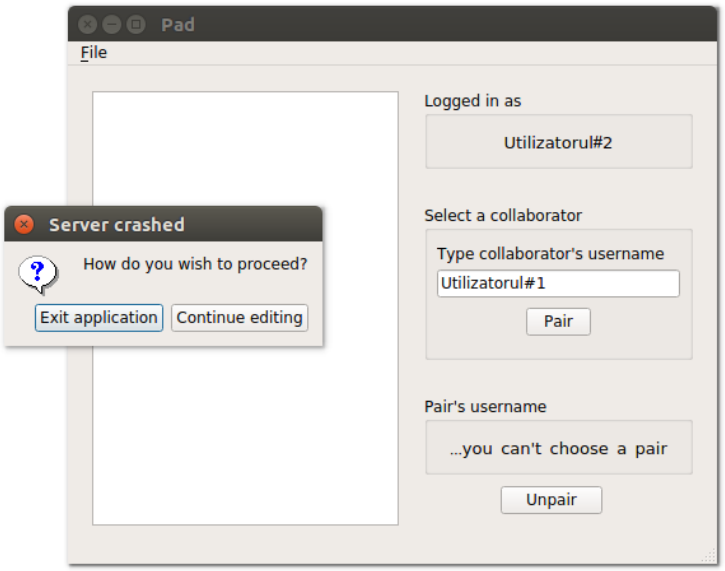

Figura 6. Clientul este informat asupra faptului că serverul a suferit o eroare

5 Concluzii

Există o serie de idei care, odată implementate, ar putea duce la îmbunătăţirea experienţei utilizatorilor şi, implicit, a eficienţei aplicaţiei.

- un chat pus la dipozitia colaboratorilor
- utilizarea bazelor de date
- păstrarea unui istoric al modificărilor
- criptarea datelor
- implementarea unui sistem de permisiuni

Bibliografie

1. `https://en.wikipedia.org/wiki/Operational_transformation`
2. Atelier de programare în reţele de calculatoare (2001), Editura Polirom Iaşi, S. Buraga, G. Ciobanu
3. Computer Networks 5th Edition (2010), Andrew S. Tanenbaum, David J.Wetherall
4. `http://rapidjson.org/`
5. `https://www.qt.io/`
6. `http://en.cppreference.com/w/cpp/thread/thread`