Collaborative Notepad

Raport tehnic

Maria Munteanu

Universitatea *Alexandru Ioan Cuza* Iasi, Facultatea de informatica

Abstract. In acest raport este prezentat modul de functionare al aplicatiei **Collaborative Notepad** si cele mai importante aspecte ce tin de implementarea acesteia.

Keywords: Notepad · ma ami gandesc · si aici ma mai gandesc.

1 Introducere

Proiectul reprezinta o implementare a unei aplicatii de tipul *notepad*, ce permite editarea unui document simultan de catre doi utilizatori. Fisierele sunt stocate pe server, clientul avand posibilitatea sa descarce fisierele.

2 Tehnologiile utilizate

2.1 TCP

Interactiunea dintre client si server se bazeaza pe o conexiune de tipul *TCP*. In alegerea protocolului am tinut cont de faptul ca in acest tip de aplicatie corectitudinea datelor reprezinta cel mai important aspect; *TCP* asigura faptul ca datele ajung corect, atat de la server la client cat si invers.

Implementarea serverului este concurenta, astfel serverul poate procesa simultan mai multe cereri de la clienti diferiti.

2.2 QT

Pentru crearea interfatei grafice a aplicatiei am folosit framework-ul *QT*. Cu ajutorul acestui framework, am implementat atat interfata grafica pentru utilizator(client) cat si o interfata grafica minimala pentru server. Am ales sa folosesc acest framework deoarece este usor de utilizat si ofera o multitudine de tool-uri ce faciliteaza implementarea aplicatiilor.

3 Arhitectura aplicatiei

3.1 Diagrama aplicatiei

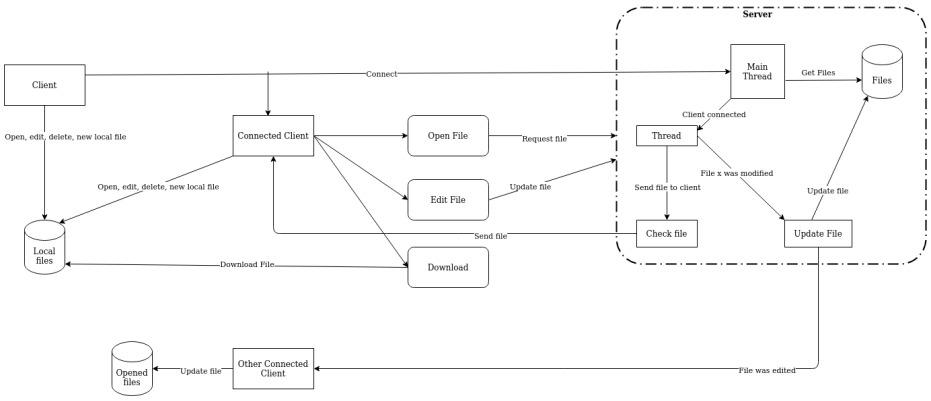

Fig. 1. Diagrama aplicatiei

3.2 Conceptele implicate

Serverul creaza un thread pentru fiecare client si foloseste mutex-uri pentru protejarea functiei *accept()*. Clientul poate functiona atat conectat la server, cat si in lipsa unei conexiuni. Prin intermediul interfatei grafice implementate, utilizatorul se poate conecta si deconecta de la server. Datele de conectare(adresa si portul) sunt citite din fisierul *config*. Dupa realizarea conectarii la server, un nou thread este creat pentru a asculta mesajele primite de la server fara a bloca interfata grafica. Utilizatorul are posibilitatea de a deschide fisiere stocate atat local cat si pe server. Pentru fiecare fisier, serverul retine utilizatorii ce il editeaza. Atunci cand se incearca deschiderea unui fisier stocat pe server, este trimisa o cerere serverului iar acesta verifica daca a fost atins numarul maxim de utilizatori(2) ce pot edita un fisier simultan si trimite clientului raspuns. De fiecare data cand se produce o modificare a unui fisier de pe server, clientul trimite serverului detaliile modificarii. Serverul trimite catre ceilalti clienti o notificare cu privire la modificarile facute fisierului.

4 Detalii de implementare

4.1 Setarea serverului

```
int serverMain::startServer ( ) {
  if ( ( serverSocketDescriptor = socket ( AF_INET, SOCK_STREAM, 0
       ) ) == -1 ) {
    //eroare la crearea socket-ului
  }
  int on = 1;
  setsockopt ( serverSocketDescriptor, SOL_SOCKET, SO_REUSEADDR, &
      on, sizeof ( on ) );
  bzero ( &serverSocket, sizeof ( serverSocket ) );
  serverSocket.sin_family = AF_INET;
  serverSocket.sin_addr.s_addr = htonl ( INADDR_ANY );
  serverSocket.sin_port = htons ( PORT );
  if ( bind ( serverSocketDescriptor, ( struct sockaddr * ) &
      serverSocket,sizeof ( struct sockaddr ) ) == -1 ) {
    //eroare la bind
  }
  if ( listen ( serverSocketDescriptor, 2 ) == -1 ) {
    //eroare la listen
  }
  return 0;
}
```

4.2 Crearea thredurilor in server

Functia *spawnThread* creaza un thread nou pentru fiecare client. Fiecare thread proceseaza requesturile primite de la client pana cand acesta se deconecteaza sau pana la aparitia unei erori.

Functia *getAvailable()* returneaza un ID ce nu este deja utilizat de alt client.

```
int serverMain::threadCallback ( int clientId ) {
  int error = 0;
  while ( ! error ) {
    error = processRequest ( clientId );
  }
  return 0;
}

void serverMain::spawnThread ( ) {
  for ( ;; ) {
    int clientId = getAvailable ( );
    if ( ( clientSocketDescriptor[ clientId ] =
    waitForClients ( ) ) != -1 ) {
      available[ clientId ] = false;
      threadPool[ clientId ] =
      std::thread ( &serverMain::threadCallback, this, clientId );
```

```
      threadPool[ clientId ].detach ( );
    }
  }
}
```

4.3 Acceptarea clientilor

Functia *accept()* este protejata cu ajutorul mutexurilor.

```
int serverMain::waitForClients ( ) {
  socklen_t length = sizeof ( fromSocket );

  bzero ( &fromSocket, sizeof ( fromSocket ) );
  //static std::mutex m_lock; variabila m_lock este declarata ca
      variabila statica a clasei
  m_lock.lock ( );
  int client;
  if ( ( client = accept ( serverSocketDescriptor,
              ( struct sockaddr * ) &fromSocket, &length ) ) <
      0 ) {
    return -1;
  }
  m_lock.unlock ( );
  return client;
}
```

4.4 Procesarea requesturilor

Atunci cand se conecteaza la server, clientul trimite un mesaj cu username-ul acestuia. In momentul in care clientul vrea sa deschida un fisier de pe server, acesta trimite requestul "list", iar serverul ii raspunde cu lista fisierelor disponibile. Cand clientul incearca sa deschisa un fisier, trimite mesajul "open" iar serverul ii trimite inapoi raspunsul, in functie de numarul de utilizatori ce il editeaza deja. De fiecare data cand clientul modifica un fisier, ii trimite serverului detaliile modificarii, iar acesta trimite notificare celorlalti clienti. Mesajul "new" este trimis atunci cand se creaza un fisier nou, iar "delete" atunci cand se sterge.

Atunci cand clientul se deconecteaza sau inchide aplicatia, mesajul "quit" este trimis.

```
const std::unordered_map< std::string, int > requestsNumbers {
      { "quit", -1 }, { "username", 1 }, { "list", 2 }, , { "open
          ", 3 }, { "update", 4 }, { "new", 5 }, { "delete", 6 }
          };
```

```
     ...
switch ( requestsNumbers.at ( mesaj primit de la client ) )
```

4.5 Setarea clientului

In client se creaza un socket pentru comunicarea cu serverul si se conecteaza. Este trimis un mesaj catre server cu username-ul acestuia.

```
 if ( ( socketDescriptor = socket ( AF_INET, SOCK_STREAM, 0 ) ) ==
      -1 ) {
    //eroare la crearea socket-ului
  }
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = inet_addr ( address );
  server.sin_port = htons ( port );
  if ( ::connect ( socketDescriptor, ( struct sockaddr * ) &server
      ,
          sizeof ( struct sockaddr ) ) == -1 ) {
    //eroare la conectare
  }

  sendRequest ( { "username", ( char * ) this->username.data ( ) }
       );
  spawnListeningThread ( );
```

4.6 Crearea threadului in client

Dupa conectarea la server se creaza un nou thread care asculta mesajele venite de la server.

```
void ClientMain::spawnListeningThread ( ) {
  listeningThread = new std::thread ( &ClientMain::threadCallback,
       this );
  listeningThread->detach ( );
}

int ClientMain::threadCallback ( ) {
  int error = 0;
  while ( ! error ) {
    error = listen ( );
  }
}
```

4.7 Comunicarea cu serverul

```
void ClientMain::sendRequest ( std::initializer_list< char * >
    msgs ) {
  for ( char *msg : msgs ) {
    int requestLength = strlen ( msg );
    if ( write ( socketDescriptor, &requestLength, sizeof ( int )
        ) <= 0 ) {
      //eroare la scriere
    }
    if ( write ( socketDescriptor, msg, requestLength ) <= 0 ) {
      //eroare la scriere
    }
  }
}
```

```
int ClientMain::listen ( ) {
  int length;
  int closed;
  if ( closed =
       ( read ( socketDescriptor, &length, sizeof ( length ) ) ) <
            0 ) {
    //eroare la citire de la server
  } else {
    if ( closed == 0 )
      return -1;
  }
  char *msg;
  if ( read ( socketDescriptor, msg, length ) < 0 ) {
    //eroare la citire de la server
  }
}
```

```
void ClientWindow::on_actionOpen_file_from_server_triggered ( ) {
  clientMain->sendRequest ( { "list" } );
}
```

4.8 Scenarii de utilizare

1. Un utilizator incearca sa stearga un fisier pe care il mai editeaza alt utilizator

In momentul in care un client incearca sa sterga un fisier, in caz ca mai este un utilizator care editeaza fisierul respectiv, cea de-al doilea utilizator este

intrebat daca este de acord ca fisierul sa fie sters. In caz contrat, stergerea nu este executata.

2. Doi utilizatori editeaza acelasi fisier in acelasi loc

Pentru a preveni editarea incorecta a fisierelor, aplicatia utilizeaza semafoare, pentru a permite accesul simultan la aceeasi resursa dar evita actiunile opuse ale utilizatorilor(un utilizator scrie iar celalalt sterge).

5 Concluzii

Modurile prin care aplicatia ar putea fi imbunatatita:

Interfata grafica a utilizatorului ar putea fi imbunatitia prin adaugarea optinilor de tipul: schimbare culoare fundal, schimbare culoare text, schimbare font text, text bold, italic.

De asemenea, utilizatorul ar putea sa primeasca in timp real notificari despre clientii conectati si fisierele adaugate.

O alta imbunatatire ar fi ca fisierele create de un utilizator sa fie private, iar ceilalti utilizatori sa solicite permisiunea acestuia daca vor sa le editeze.

References

1. https://man7.org/linux/
2. https://doc.qt.io/
3. https://stackoverflow.com/
4. https://thispointer.com
5. https://wiki.qt.io
6. https://en.cppreference.com
7. https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
8. https://www.qtcentre.org/
9. https://www.youtube.com/watch?v=EkjaiDsiM-Qlist=PLS1QulWo1RIZiBcTr5urECberTITj7gjA
10. https://www.youtube.com/watch?v=I96uPDifZ1wlist=PLGLfVvz$_L VvQrqLpBB4Sfz7gxMN9shP6v$