My SVN
Proiect Retele

Luca Andrei, IIA5

Facultatea de Informatica Iasi

luca.andrei96@gmail.com

Keywords: TCP, version control, diff-files, concurent client/server

1 Introducere

Datorita impresionantei cresteri a utilizarii internetului si a calcultoarelor din ultimii ani, se impune crearea de produse software din ce in ce mai diverse si mai complexe menite sa ne satisfaca nevoile. In acest context, de-a lungul vremii s-au creat diferite aplicatii care au fost menite sa ajute programatorii sa conceapa produse mai bune, mai repede si mai usor.

Un tip de aplicatie ce se preteaza acestor cerinte este cel de control de versionare("version control"). O astfel de aplicatie ofera programatorilor posibilitatea de a-si versiona aplicatia cu posibilitatea de a se intoarce la o versiune anterioara, avand astfel controlul asupra anumitor probleme ce ar putea aparea de-a lungul realizarii aplicatiilor.

De asemenea, este nevoie ca acesta platforma sa poata oferi accesul mai multor persoane (produsele software complexe sunt realizate in general de echipe) si sa ofere o modalitate eficienta si sigura de stocare a datelor.

Astfel, voi incerca sa rezolv aceasta problema creand o aplicatie in retea care se foloseste de paradigma client/server si care va permite urmatoarele functionalitati:

- inregistrarea/logarea utilizatorilor in sistem
- descarcarea/incarcarea/stergerea de fisiere din sistem
- operatiuni de salvare a modificarilor("commit") cu diferite mesaje
- operatiuni de intoarcere la versiuni anterioare("revert")

Aplicatia va fi utilizata doar pentru un produs software, deoarece contine un singur repository. Vor exista doua tipuri de clienti:

- contributor - va avea toate drepturile asupra fisierelor(cu commit/add)
- normal-user - va avea doar drepturi de acces asupra datelor nu se de modificare(fara commit/revert/upload)

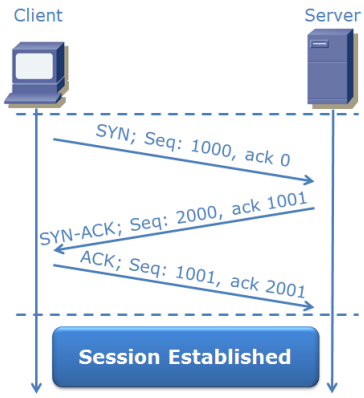
2 Tehnologii utilizate

2.1 TCP - Transimission Control Protocol

TCP reprezinta un protocol al nivelului transport orientat conexiune destinat transferului fiabil de date in retea. TCP asigura o legatura sigura din punct de vedere a transferului de date, adica datele sunt verificate de erori, se asigura ca pachetele sunt trimise in ordine, previne congestionarea retelei, etc. Din cauza necesitatii in aplicatia My SVN de o modalitate de transmitere a datelor cat mai sigura(fara pierderi de date) voi folosi protocolul TCP.

Astfel, aplicatia va avea urmatoarele caracteristici:

- conexiunea intre server/client se va realiza folosind operatiunea metoda "three way handshaking" utilizand primitivele accept(), listen() si connect()



(Fig 1.) Three way handshaking method.

- pentru adresare voi folosi la nivelul retea protocolul IP $\Rightarrow$ fiecare nod din retea va fi identificat printr-o serie de 32 biti, iar la nivelul transport voi utiliza porturile(metoda de a identifica in mod unic procesele ce ruleaza pe un sistem din retea).
- transmiterea de informatii o voi realiza prin socket-uri, o abstractiune a descriptorilor de fisiere din programarea ANSI-C careia i se pot atasa adrese $IP + port$
- intre socket-uri se vor transmite secvente de bytes utilizand primitive din familia read() si write()

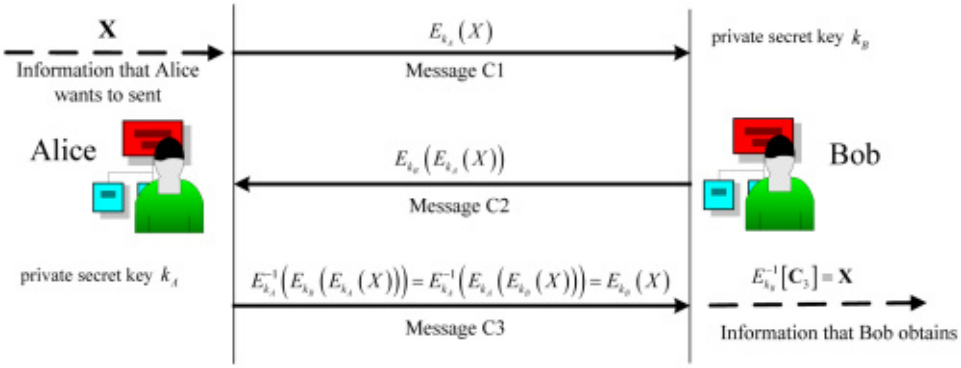
2.2 My SVNProtocol

Voi incerca sa implementez un protocol la nivelul aplicatie ce imi faciliteaza executarea comenzilor de adaugare, descarcare de fisiere similar cu FTP (File Transfer Protocol) adaugand o parte de criptare a tuturor datelor transmise. Pentru criptare, avand in vedere ca inca nu am studiat astfel de metode voi

folosi o metoda de criptare simpla: "Cifrul lui Cezar" combinata cu protocolul de comunicare "three-pass" al lui Ali Shamir, deoarece consider ca in aceasta situatie putem sacrifica viteza de transfer a datelor(deoarece sunt date relativ mici) si putem avea mai multa grija la securitatea datelor.

Protocolul minimal va avea urmatoarele primitive definite:

- *login* - permite operatia de login a unui user
- $create\_user < user > < parola > < drepturi >$ - permite crearea unui user (drept ce il au doar administratorii)
- $download < fileName >$ - permite operatia a unui fisier din repository
- $upload < fileName >$ - permite operatia de adaugare a unui fisier in repository pentru a putea fi urmarit drept ce il au doar administratorii)
- $commit < message >$ - va incarca schimbarile aduse la repository-ul curent (drept ce il au doar administratorii)
- $revert < versionNumber >$ - permite intoarcerea la o versiune anterioara a aplicatiei, modificand continutul directorului currentFiles, din care mai departe utilizatorii pot downloada fisierele (drept ce il au doar administratorii)
- $uploadedFiles$ - permite vizualizarea numelor fisierelor care sunt urmarite de aplicatie (drept ce il au doar administratorii)
- $currentFiles$ - permite vizualizarea numelor fisierelor care se afla in repository
- $workFiles$ - permite vizualizare numelor fisierelor care se afla in folderul de lucru al clientului
- $status < versionNumber/"all" >$ - permite vizualizare mesajelor care au fost atasate commit-urilor
- $showDiff < versionNumber >< fileName >$ - permite vizualizarea diferentelor realizate in timpul unui commit pentru un anumit fisier/.

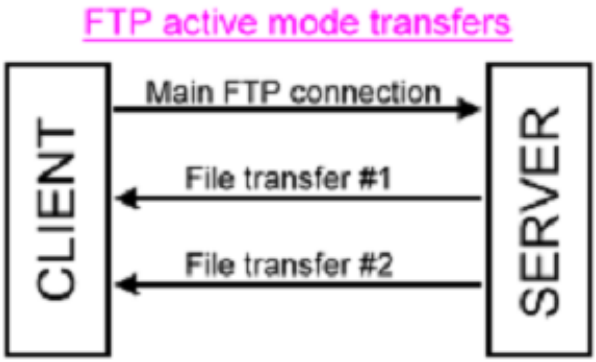


(Fig 2.) Three-pass Protocol

Astfel, fiecare instanta va avea cate o cheie privata care va fi generata in momentul nasterii acesteia care va reprezenta un contor cu care vom deplasa un caracter in tabela ascii(conform cifrului lui Cezar).

Pentru partea de transfer de date, voi utiliza schema FTP adica voi avea o zona de control prin care voi analiza diferitele comenzi pe care le voi primi de la client si partea de date, unde voi realiza efectiv transferul de fisiere.

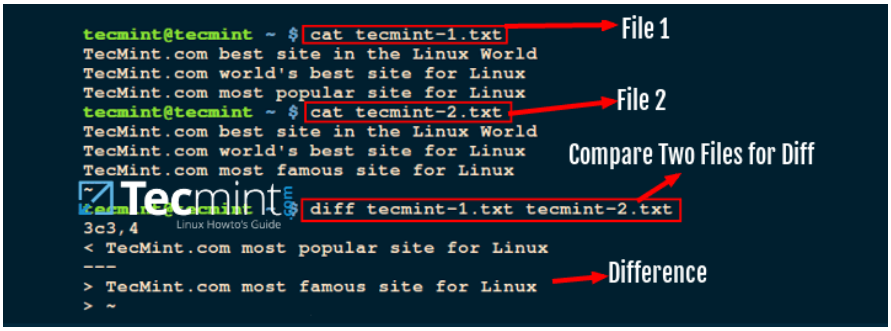
Pentru a realiza acest lucru, in momentul in care este nevoie de transfer de fisiere intre cele doua instante, voi crea o noua conexiune de la server la client prin care voi realiza efectiv transferul de date.



(Fig. 3) Schema generala a protocolului FTP

2.3 Comanda Diff din Bash

Pentru a stoca in mod eficient datele de pe server, avand in vedere ca de regula pentru o aplicatie se realizeaza o foarte multe commit-uri, este nevoie de o metoda prin care sa stocam eficient datele de la un commit la altul. Din aceasta cauza pe server voi retine doar log-uri de diferente intre versiunile anterioare si nu fisierul complet. In acest sens, voi folosi(cel mai probabil daca nu gasesc un API decent) comanda Diff din Bash care primind ca parametru 2 fisiere text va afisa diferentele intre acestea.



(Fig. 4) Exemplu de utilizare a comenzii Diff din Bash.

2.4 Comanda Patch din Bash

Pentru a folosi fisierele generate de comanda Diff intr-un mod eficient, voi folosi o alta comanda din Bash: patch. Aceasta comanda permite aplicarea unor diferente dintre 2 fisiere realizand modificarile in fisierul original. Este nevoie de redirectarea input-ului aceste comenzi catre fisierul de diferente, insa apoi se poate aplica oricarui fisier, aplicandu-i diferentele specificate la input.

2.5 SQLite3

SQLite3 este o librarie care implementeaza o modalitate de a manipula baze de date relationale. Motivul pentru care am ales aceasta librarie este pentru ca se pot executa instructiuni SQL foarte simplu si fara prea multe probleme si pentru ca datele stocate sunt criptate. Am ales sa folosesc o astfel de baza de date pentru stocarea utilizatorilor. Astfel, fiecare linie va avea trei campuri: username, parola, drepturile de acces, care vor fi verificate la logare si apoi mai departe in utilizarea aplicatiei.

3 Arhitectura aplicatiei

3.1 Concepte Implicate

Paradigma Server/Client concurent

Pentru a utiliza toate resursele disponibile pe sistemul pe calcul pe care il folosim si de asemenea pentru a avea o aplicatie mult mai rapida si eficienta este necesara utilizarea unei paradigme client/server concurente.

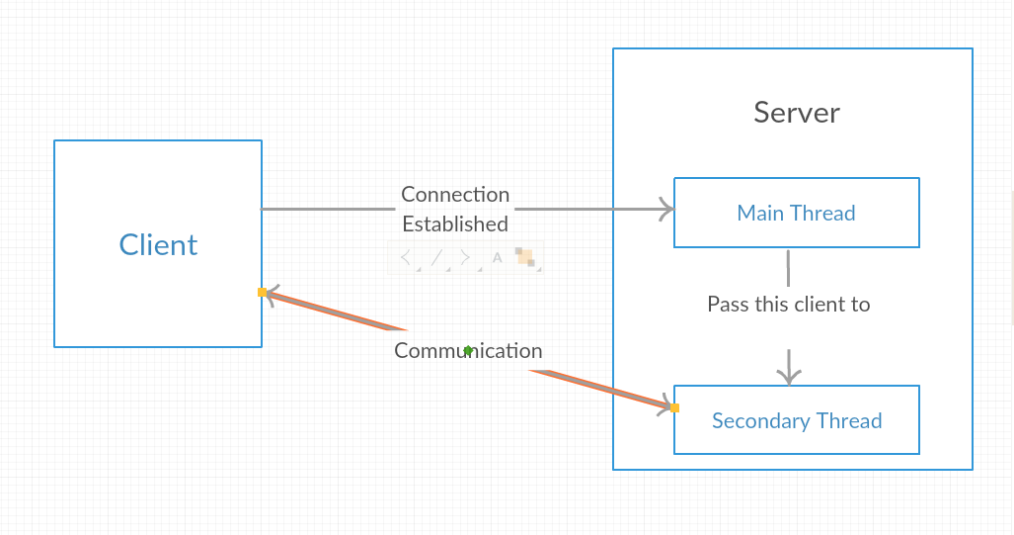
Ideea principala este de a servi fiecare client pe cate un fiu(utilizand fork()) putand astfel sa servim mai multi clienti deodata. Pentru a realiza acest lucru, este necesara crearea unui socket nou(accept-ul va realiza acest proces) pentru server prin care se va realiza comunicarea dintre fiu si client. Socket-ul initial va fi pastrat si va fi utilizat de procesul tata/firul principal de executie pentru a imparti acesti clienti diferitor fii.

Putem utiliza aceeasi paradigma si pentru clienti. In momentul in care trimitem o comanda, aceasta sa fie procesata mai departe de un fiu al procesului inital, dar in acest caz ar fi aplicabil doar in cazul upload-ului si am considerat ca nu se preteaza in cazul aplicatiei mele.

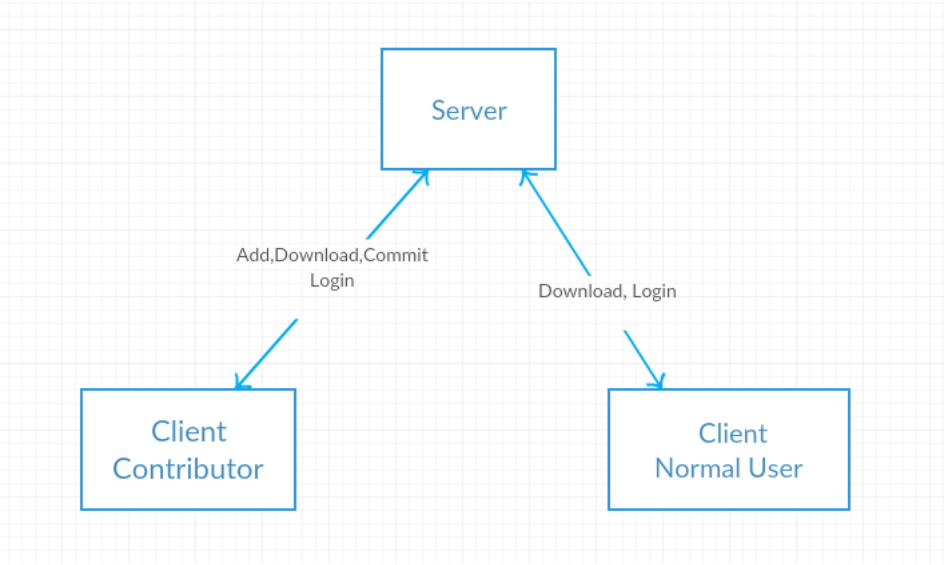
3.2 Diagrama Aplicatiei

In diagrama aplicatiei voi considera doar comenzile principale, nu si cele secundare.

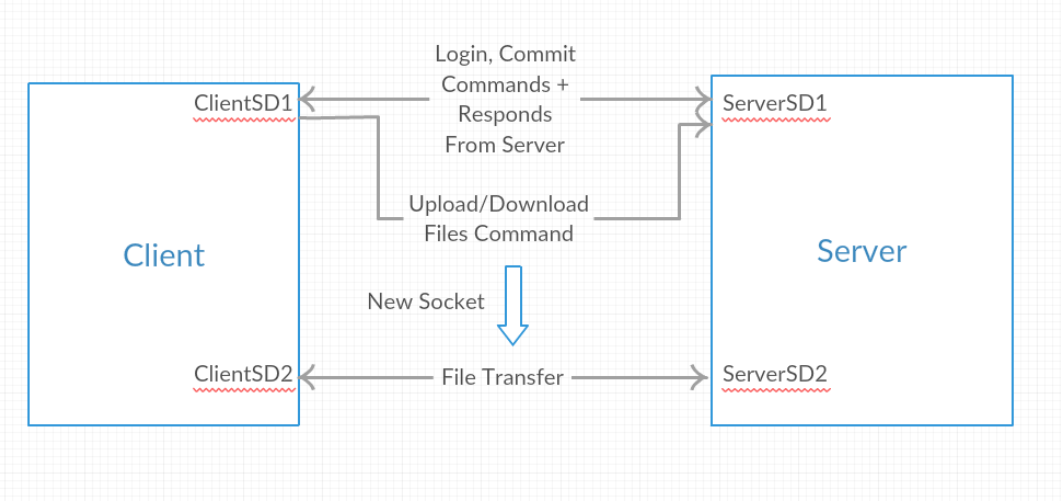
Servirea Clientilor dupa Realizarea Conexiunii(Fig. 6)



Schema Generala a Drepturilor Utilizatorilor(Fig 5.)



Schema Generala a Comunicatiei intre client si server(Fig. 7)



4 Detalii de implementare

4.1 Detalii generale

Pentru a compila aplicatia am lasat in arhiva proiectului inclusiv fisierul Makefile care contine comenzile de compilare.

Pentru o prima rulare a aplicatiei va fi nevoie de o compilare, iar apoi de o rulare a scriptului "createUsersDB" care va crea o baza de date pentru users si un user default cu username-ul 'andrei' si parola 'luca' cu drepturi de administator. Pornind de la acest cont, se pot crea alte conturi de utilizator normal/administrator.

Din acest punct, utilizatorul are dreptul la toate comenzile propuse mai sus.

Voi incerca, in cele ce urmeaza, sa explic in mare principalele comenzi si modul cum le-am implementat.

Important de mentionat este faptul ca pentru a realiza aceste comenzi am creat la executia server-ului/clientului urmatoarele directoare cu urmatoarele roluri:

- *./work* - director in directorul curent al clientului, din care se va realiza commit-ul, in care se va face descarcarea fisierelor
- *./serverFiles* - director in directorul curent al server-ului care va contine diferite directoare necesare, inclusiv versiunile de commit.
- *./serverFiles/currentFiles* - va contine fisierele curente ale repository-ului, acestea pot fi descarcate utilizand comanda download
- *./serverFiles/uploadedFiles* - va contine o prima versiune a fisierelor care pot fi urmarite. Utilizatorul va trebui sa adauge fiecare fisier care doreste sa fie urmarit in acest director utilizand comanda upload

– *./serverFiles/commitedFiles* - director auxiliar care ma va ajuta in momentul in care clientul doreste sa faca un commit (voi descarca in acest director fisierele din directorul ./work al acestuia)

De asemenea, de mentionat este si faptul ca in retea toata comunicarea care este importanta, este criptata si este utilizat protocolul "three pass prtocol".

Commit

Pentru a realiza un commit pentru inceput voi realiza un revert la versiunea curenta(deoarece diferentele sunt valide doar de la o versiune de commit la alta) pentru ca este posibil ca utilizatorul sa fi facut un revert ulterior. Apoi clientul va trimite toate fisierele sale din directorul ./work, iar server-ul le va depozita in directorul ./serverFiles/commitedFiles. Apoi va parcurge directorul ./serverFiles/commitedFiles si va realiza diferentele(pe care le voi stoca intr-un director cu numele numarul versiunii) dintre fisierele cu acelasi nume din directorul ./serverFiles/currentFiles si directorul ./serverFiles/commitedFiles, realizand astfel diferentele de la ultimul commit.

Revert

Pentru a realiza un commit, voi copia toate fisierele din directorul ./serverFiles/uploadedFiles in directorul ./serverFiles/currentFiles si voi parcurge fiecare versiune de commit(de la 1 pana la cate versiuni am) si pentru fiecare versiune voi aplica diferentele de la acea versiune pe fiecare fisier din ./serverFiles/currentFiles.

Upload - Download este similar

Pentru a realiza un upload, pentru inceput verific daca fisierul indicat exista, iar daca nu exista erori, notific server-ul ca doresc sa fac o operatie de upload, iar acesta isi creaza un nou socket si incepe sa asculte, dupa care trimite clientului port-ul(gasit cu getsockname()) clientului pentru a sti la ce port sa se conecteze. In acest context, clientul va realiza connect la acel port si ip-ul inital, dupa care server-ul il va accepta. In acest moment se poate face transferul de date prin aceasta conexiune noua, fara a utiliza conexiunea initiala. Acest model este similar cu cel al protocolului FTP.

4.2 Cod Relevant - Pseudocod

Am considerat oportuna doar structura generala a codului a fi scrisa aici, in rest se pot vedea partile interesante comentate in cod.

Trimiterea/Primirea de date in retea

```
void send_data(data, senderSD, senderKey)
{
  data = encrypt(data, senderKey);
  write(data, senderSD);
  encrypted_data = read(senderSD);
  decrypt(encrypted_data, senderKey);
  write(data, senderSD);
}

data recieve_data(receiverSD, receiverKey)
{
  data = read(receiverSD);
  encrypted_data = encrypt(data, receiverKey);
  send(encrypted_data, receiverSD);
  encrypted_data = read(receiverSD);

  return decrypt(encrypted_data);
}
```

Functia main - Client

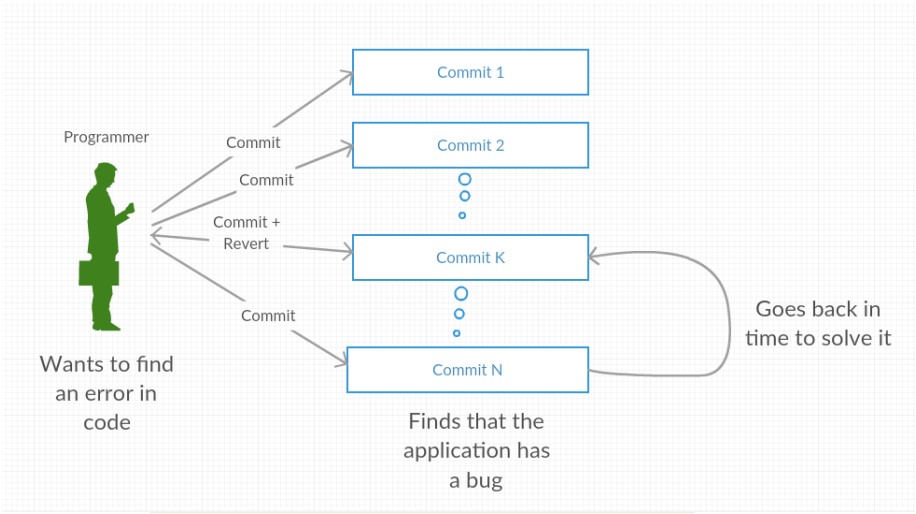
```
main()
{
  clientKey = generateRandomKey();
  clientSD = createSocket(); //fara bind
  connect(clientSD, ip, port,...); //functia reala are alta signatura
  userData = login();

  send_data(userData, clientSD);
  authToken = receive_data(clientSD);

  while (command != quit) {
    command = readCommand(); //citesc comanda
    send_data(clientSD, command, clientKey); //trimit comanda la server
    data = recieve_data(clientSD, clientKey);  // primesc datele raspuns
    processData(data); // procesez datele primite ca raspuns
  }

}
```

Functia main - Server

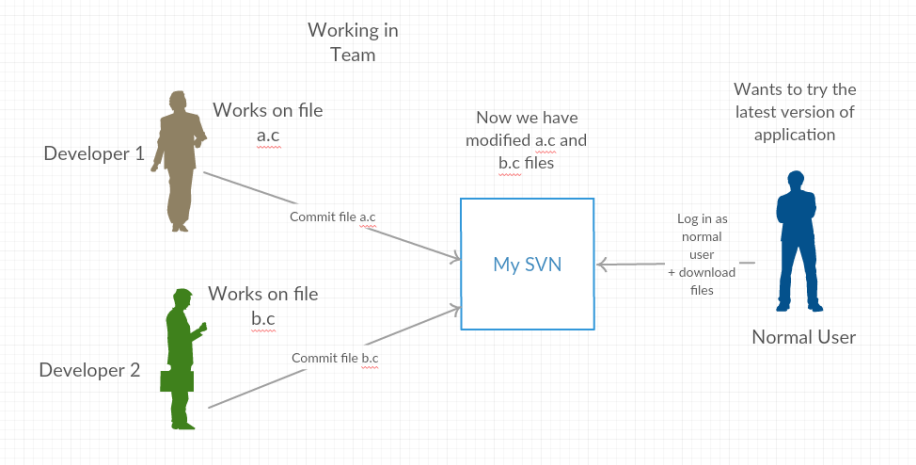
```
main()
{
  serverKey = generateRandomKey();
  serverSD = createSocket(); //aici voi realiza si bind-ul
  listen(serverSD, NMAXCLIENTS);
  while (1) {
    clientSD = accept(serverSD, ...);
    t = createThread(workForCurrentClient); // serveste clientul
    close(clientSD);
  }
}

workForCurrentClient()
{
  command = receive_data(serverSD, serverKey);
  if (command is valid and currentUser has rights to execute it) {
    if (command needs data transfer) {
      pid = fork();
      if (pid == 0) {
        close(serverSD);
        data = computeDataForTrnasfer();
        newServerSocket = createNewSocket();
        sendDataUsingTheNewSocket(data, newServerSocket);
        close(serverSD);
      } else {
        continue processing;
      }
    } else {
      data = computeResponse(command);
      send_data(response);
    }
  }
}
```

4.3 Cazuri de utilizare(use-case)

Pentru gasirea erorilor in cod(Fig. 8)



Pentru lucrul in echipa si impartasirea aplicatiei cu clienti(Fig. 9)



5 Concluzii

Aplicatia in mod evident se poate imbunatati considerabil. La structura aceasta se pot adauga mai multe repository-uri(nu un singur repository mare cum este in aces moment) diferite pentru utilizatori care sa poata fi impartasite cu alti utilizatori.

De asemenea, se poate implementa un sistem de branching care sa permita dezvoltarea de aplicatii mai complexe ce necesita adesea functionalitati noi si o interfata grafica prietenoasa astfel incat sa nu fie necesara utilizarea de la linia de comanda.

In concluzie, aplicatia ofera facilitati minimale pentru o aplicatie de tipul "version control" si satisface principiile de securitate/eficienta la un nivel minimal.

6 Bibliografie

- Cursurile de "Retele de calculatoare" de la Facultatea de Informatica Iasi, materie predata de Lenuta Alboaie.
- http://www.hackmageddon.com/2011/04/17/tcp-split-handshake-attack-explained/
- https://www.osapublishing.org/oe/abstract.cfm?uri=oe-20-3-2386
- https://enterprisedt.com/products/edtftpjssl/doc/manual/html/howtoftpthroughafilewall.html
- http://www.tecmint.com/best-linux-file-diff-tools-comparison/
- http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- https://creately.com/Draw-UML-and-Class-Diagrams-Online