

My SVN

Marincia Cătălin, grupa B1, anul II
catalinmarincia@gmail.com

Faculty of Computer Science Iași

Abstract. Acest document conține informații despre realizarea unui version control system primitiv.

Keywords: TCP · SVN · VCS · Scratch · Computer Networks

1 Introducere

Trăim în era digitalului. Invențiile în domeniul științei calculatoarelor reprezintă puternici piloni ce fac posibilă dezvoltarea societății în care trăim. Odată cu necesitatea dezvoltării în timp util a unor produse software diverse cu o calitate și complexitate ridicate, au fost dezvoltate diferite aplicații(unelte) menite să faciliteze munca programatorilor.

Un tip de aplicație ce se încadrează cu ușurință în categoria descrisă este cel de control de versionare("version control system"). O astfel de aplicație pune la dispoziție programatorilor posibilitatea de a versiona un anumit proiect oferind facilități precum întoarcerea la o versiune anterioară pentru a studia existența unor bug-uri(reproducere, regressions, fixes, etc.), minimizarea lucrului manual cu fișiere(back-up-ul iresponsabil) și a dezorganizării.

Pe lângă cele menționate, o astfel de platformă trebuie să poată oferi accesul mai multor persoane la sursele unui proiect întrucât produsele software complexe sunt realizate, preponderent, în echipe. Trebuie să ofere o modalitate eficientă și sigură atât din punct de vedere financiar, cât și din punct de vedere al stocării datelor.

Având în vedere cele de mai sus, voi încerca să rezolv această problemă dezvoltând o aplicație al cărei arhitecturi va fi de tip client/server și va permite următoarele funcționalități:

- înregistrarea și conectarea în sistem a utilizatorilor;
- descărcarea(clonarea), încărcarea și ștergerea de fișiere din acest sistem;
- posibilitatea de a salva modificările dorite("commit") cu diferite mesaje("commit messages");
- posibilitatea de întoarcere la versiuni anterioare("revert").

Aplicația va putea fi utilizată cu mai multe produse software. Vor exista două tipuri de clienți:

- contributor - va avea drepturi de modificare a conținutului fișierelor(cu commit/add);

- normal-user - va avea doar drepturi de acces asupra datelor din repository de tip read-only, neavând posibilitatea de a le modifica.

2 Tehnologii utilizate

2.1 TCP(Transmission Control Protocol)

TCP este un protocol orientat conexiune ce acționează la nivelul transport și este destinat transferului de date fiabil în rețea. TCP asigură o comunicare sigură a partenerilor implicați, transferul de date este verificat de erori, se asigură că ordinea în care sunt transmise pachetele este una corectă, previne congestiunea rețelei, asigură disponibilitatea datelor, etc. Datorită faptului că un sistem de versionare nu își poate permite transferuri de date nesigure și incomplete, voi utiliza protocolul TCP.

Aplicația va avea ca bază următoarele caracteristici:

- datorită faptului că voi folosi TCP, conexiunea între server/client se va realiza folosind metoda **three way handshaking** utilizând API-ul BSD pentru lucrul cu sockets, primitivile `accept()`, `listen()`, `connect()`;

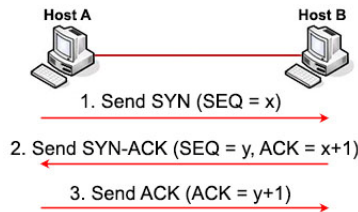


Fig. 1. Metoda three way handshaking

- la nivelul rețea voi folosi protocol IP(astfel, combinația TCP/IP). Fiecare nod va fi identificat printr-o adresă formată din 32 de biți, împărțită în 4 secțiuni(adresă IPv4). De asemenea, voi utiliza porturi pentru a identifica în mod unic procesele ce rulează pe sistem din rețea;
- transmiterea informațiilor(datelor) va fi realizată prin socket-uri. Construite să poate fi folosite ca niște descriptori de fișiere, aceștia au asociate o adresă IP și un port.
- transmiterea de secvențe de bytes se va face utilizând primitivile din familia `read()` și `write()` fiind învelite de funcții proprii cu anumite funcționalități precum citire și decriptare, criptare și scriere, etc.

2.2 MySVN protocol

Tipologia acestei aplicații determină crearea unui protocol particular ce va facilita diferite comenzi de versioning systems precum adaugare de cod, trimitere de comenzi, descărcare de fișiere.

Deoarece nu am găsit o librărie de criptografie care să aibă o licență permisivă și o metodă de criptare implementată manual ar ridica probleme de performanță, am ales ca mesajele să fie trimise necriptat între client și server.

Ajutându-mă de anumite librării precum cea de manipulare a fișierelor de tip json, voi putea face posibilă o comunicare uniformă. Astfel, comunicarea dintre client și server va respecta următoarea schema:

Protocolul minimal al aplicației MySVN va avea următoarele primitive:

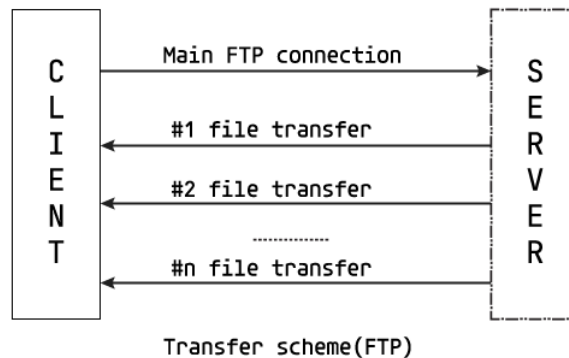


Fig. 2. Protocolul Threee-pass

- `register <username> <parola>` - permite crearea unui user, oricine își poate crea mai multe conturi chiar;
- `clone <repository_name> [<dir_path>]` - permite operația de download a unui repository de pe server;
- `init` - permite operația de creare a unui repository în folderul curent;
- `pull` - permite operația de actualizare a aplicației cu suprascrierea tuturor modificărilor curente;
- `push` - permite operația de încărcare a tuturor modificărilor curente;
- `reset` - permite resetarea directorului de lucru la versiunea locală fără modificări;
- `allow-edit <username>` - permite unui deținător de repositories să ofere drepturi de modificare unui alt utilizator;
- `block-edit <username>` - permite unui deținător de repositories să revoce drepturile de modificare ale altui utilizator;
- `allow-access <username>` - permite unui deținător de repositories să ofere drepturi de acces unui alt utilizator;

- **block-access** <username> - permite unui deținător de repositories să revoce drepturile de acces ale altui utilizator;
- **go-public** - permite unui deținător de repositories să facă un repository public;
- **go-private** - permite unui deținător de repositories să facă un repository private;
- **checkout-file** <filename> [-v <version_number>] - permite operația de download a unui fișier din repository curent de pe server cu suprascrierea versiunii locale, dacă nu e specificată versiunea va fi considerată cea mai recentă versiune;
- **message** <message> - permite appendarea de mesaje la changelog;
- **checkout** <version_number> - va permite întoarcerea la o versiune anterioară sau mai recentă a aplicației, modificând conținutul directorului de lucru. Această comandă poate fi folosită și pentru a face **revert** la o anumită versiune;
- **stage** + <filename>

filename2, ...

- permite adăugarea unui fișier în staging area.
- **unstage** . | <filename> [filename2, ...] - permite ștergerea unui fișier din staging area.
- **restore** . | <filename> [filename2, ...] - permite restaurarea unui fișier în staging area din untouched.
- **delete** . | <filename> [filename2, ...] - permite marcarea fișierelor din untouched ca fiind șterse, se va folosi un fișier **.marked_as_deleted** de tip JSON pentru a stoca acele nume ale fișierelor șterse.
- **list-staged** - permite vizualizarea numelor fișierelor care sunt monitorizate (ce urmează a fi încărcate la acțiunea de push), se află în directorul de staging;
- **list-remote** - permite vizualizarea numelor fișierelor ce sunt în repository pe server;
- **list-untouched** - permite vizualizarea numelor fișierelor ce sunt în directorul **.untouched**;
- **list-dirty** - permite vizualizarea numelor care se află în directorul de lucru al clientului;
- **status** {-v <version_number> | all} - permite vizualizarea changelog-ului atașat unei anumite versiuni;
- **diff-file** [-v <version_number>] <filename> - permite vizualizarea diferențelor dintre varianta fișierului din directorul de lucru și cea de pe server de la o anumită versiune, dacă nu e specificată versiunea va fi considerată cea mai recentă versiune;
- **diff-version** {-v <version_number> | latest} - permite vizualizarea diferențelor dintre o versiune și versiunea imediat precedentă ei;
- **get-changelog** [all | -v <version>] - permite vizualizarea changelog-ului pentru repository curent;

Am decis ca fiecare repository să aibă limitarea de a nu conține directoare (să aibă un singur nivel) întrucât operațiile de commit, diff, revert, upload, download devin considerabil mai complexe iar timpul alocat acestui proiect nu îmi

permite documentarea suplimentară în această arie.

2.3 Sqlite3

SQLite este un sistem de gestiune a bazelor de date(SGBD) sub forma unei librării scrise în limbajul C. Spre deosebire de alte SGBD, SQLite este un non client-server database engine. În schimb, este integrat în programul final.

De ce Sqlite3?

Am ales să folosesc acest SGBD deoarece:

- este open source;
- este rapid și deoarece nu necesită administrare, acesta funcționează bine în medii lipsite de asistența umană. SQLite este un bun candidat pentru utilizarea pe telefoanele mobile, televizoare, game-consoles, roboți, ceasuri și, de asemenea, mașina pe care va rula aplicația server a SVN-ului pe care îl implementez.

Schema relației USERS: (username, password)

Cu ajutorul acestei relații vom stoca informații despre utilizatorii din rețeaua acestui SVN și anume **numele de utilizator** și **parola**. Am decis ca **numele de utilizator** să fie cheie primară în această relație: nu vor exista 2 utilizatori cu același username.

Schema relației REPOSITORY: (repository_name, username, unix_date, is_public)

În această relație vom stoca informații de bază despre un anumit repository, username-ul creatorului este important deoarece doar acesta poate acorda altor utilizatori dreptul la modificarea fișierelor.

Schema relației VERSIONS: (repository_name, version, unix_date, changelog)

În această relație vom stoca informații despre o anumită versiune cum ar fi data creării, changelog-ul cu care a fost încărcată.

Schema relației STORAGE: (repository_name, version, filename, data)

În această relație vom stoca informații despre fișierele asociate unui anumit repository și unei anumite versiuni.

Schema relației PERMISSIONS: (repository_name, username)

În această relație vom stoca informații despre utilizatorii care au drepturi de scriere/modificare asupra unui anumit repository. Doar creatorul poate acorda unui utilizator drepturi asupra unui repository propriu.

Schema relației DELETIONS: (repository_name, version, filename)

În această relație vom stoca informații despre anumite fișiere șterse la o anumită versiune.

Interacțiune cu baza de date

La încercarea de conectare/înregistrare a unui utilizator credențialele acestuia vor fi căutate în tabela de date corespunzătoare(cea cu informații despre utilizatori) și în funcție de permisiuni anumite mesaje vor fi afișate clientului: în legătură cu faptul că nu este înregistrat, parola este greșită, etc.

La încercarea de upload utilizatorul va fi căutat în relația PERMISSIONS și în cazul în care acesta se află în lista de utilizatori cu drepturi de modificare acesta va putea modifica sursele remote, altfel accesul îi va fi refuzat.

La acțiune de pull, versiunea copiei locale a repository-ului va fi comparată cu cea mai recentă versiune ce se află remote și în cazul în care diferă, codul va fi actualizat local.

La acțiune de give_permissions, autorul unui repository va putea asocia altor utilizatori drepturi de modificare. Se fac interogări atât în relația USERS, cât și în relația PERMISSIONS.

La acțiunea de push, se vor insera noi intrări în tabele VERSIONS și STORAGE.

2.4 CMake

Am ales să utilizez CMake build system datorită următoarelor:

- este un build system pentru limbajele C/C++;
- are o documentație curată și menținută;
- este un build system cross-platform;
- este backward compatible;
- există integrare cu CMake pentru multe IDE-uri;
- poate crea un dependency graph;
- facilitează integrarea dependențelor în proiecte;
- facilitează crearea de static, shared și header-only librării;
- suporta native shell commands/apps execution;
- etc.

2.5 Parson

Parson este o librărie lightweight pentru manipulare fișiere în format json.

De ce Parson? Deoarece este scrisă în C, aceasta reprezintă un candidat ideal pentru proiectul în cauză.

Am ales să utilizez Parson deoarece:

- full json support(serializare, deserializare);
- API simplu;
- C89+ compatible;
- adresarea valorilor json prin dot notation(similar cu structurile C sau obiectele din limbajele de programare orientate pe obiect, e.g. "objectA.objectB.value").

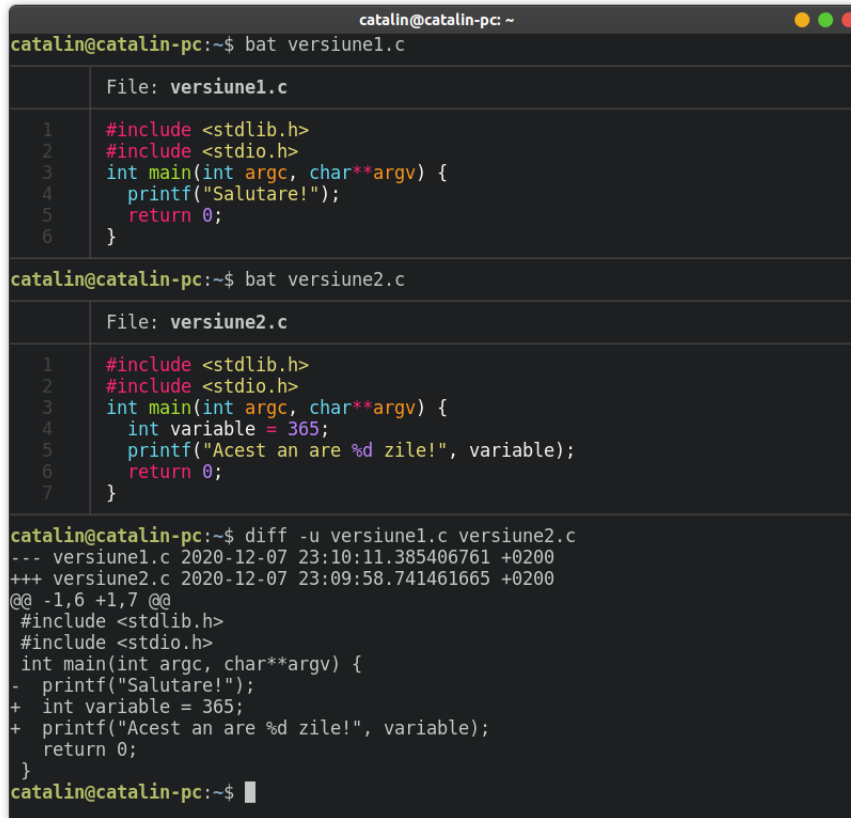
Rol în implementare

Între server și client circulă mesaje sub forma unor fișiere json care au un câmp comun și anume câmpul "message_type" al cărui valoare poate lua valori ca "commit", "commit_response", "revert", "revert_response", "add", "add_response", "login", "login_response", etc.

Astfel, mesajele la primire trec printr-un sistem ce corelează diferite funcționalități pe baza valorii câmpului "message_type".



2.6 Comanda Diff



```

catalin@catalin-pc: ~
catalin@catalin-pc:~$ bat versiune1.c
File: versiune1.c
1  #include <stdlib.h>
2  #include <stdio.h>
3  int main(int argc, char**argv) {
4      printf("Salutare!");
5      return 0;
6  }

catalin@catalin-pc:~$ bat versiune2.c
File: versiune2.c
1  #include <stdlib.h>
2  #include <stdio.h>
3  int main(int argc, char**argv) {
4      int variable = 365;
5      printf("Acest an are %d zile!", variable);
6      return 0;
7  }

catalin@catalin-pc:~$ diff -u versiune1.c versiune2.c
--- versiune1.c 2020-12-07 23:10:11.385406761 +0200
+++ versiune2.c 2020-12-07 23:09:58.741461665 +0200
@@ -1,6 +1,7 @@
 #include <stdlib.h>
 #include <stdio.h>
 int main(int argc, char**argv) {
- printf("Salutare!");
+ int variable = 365;
+ printf("Acest an are %d zile!", variable);
     return 0;
 }
catalin@catalin-pc:~$ █

```

De ce această comandă?

Deoarece, de regulă, într-o aplicație se realizează o mulțime de commit-uri este necesară o modalitate de a compara schimbările curente cu cele precedente. Datorită acestor considerente, am ales comanda Diff din Bash care primește ca parametri 2 fișiere și afișează diferențele între acestea. Voi folosi și opțiunea `-u`(unified diff) și redirectarea într-un fișier pentru a face procesul mai eficient.

3 Arhitectura aplicației

3.1 Paradigma Server/Client concurent

Am ales să folosesc această paradigma pentru a face mai eficientă aplicația din punct de vedere al utilizării resurselor de calcul. Deoarece aplicația trebuie să

fie capabilă să deservască mai mulți utilizatori simultan, am decis ca implementarea serverului să fie una concurentă.

În implementare voi folosi primitiva `fork()` pentru a asigura o stabilitate ridicată aplicației. Pentru a deservi mai mulți clienți deodată este necesară crearea unui nou proces copil separat pentru fiecare client, crearea noilor socket-uri se va face automat de către primitiva `accept`. Socket-ul inițial din server va fi preservat și utilizat de procesul tată/procesul principal pentru a face corespondența/distribuția acestor clienți diferiților fii.

Pentru client nu este absolut necesară o abordare concurentă. La trimiterea unei comenzi procesul principal client să creeze un proces fiu care să o trateze. Comanda de upload ar putea fi eficientizată folosind această arhitectură. În măsura în care timpul îmi va permite, voi face tratarea comenzii de upload în manieră concurentă.

Schema Aplicației

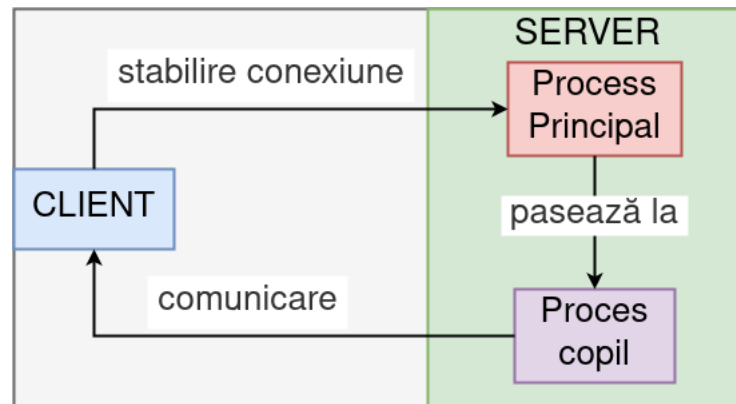


Fig. 3. Modul în care se realizează comunicarea cu clienții după realizarea conexiunii

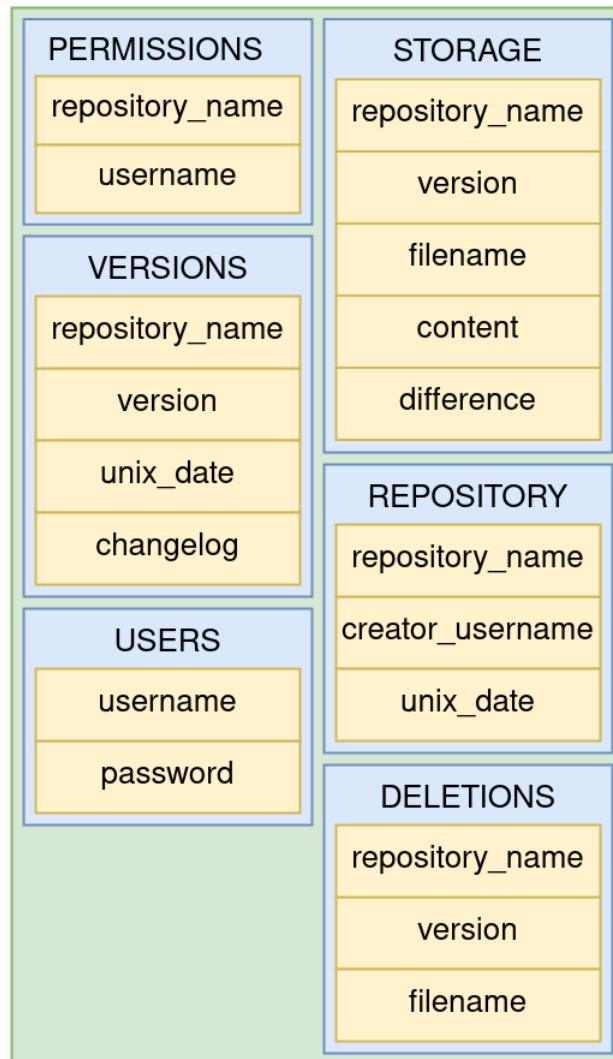


Fig. 4. Schema bazei de date

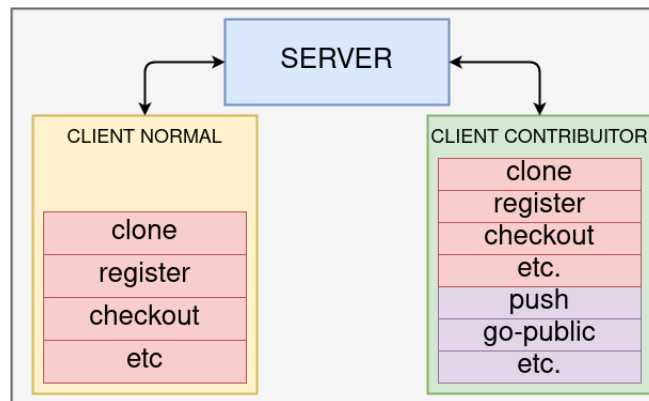


Fig. 5. Permisunile clienților

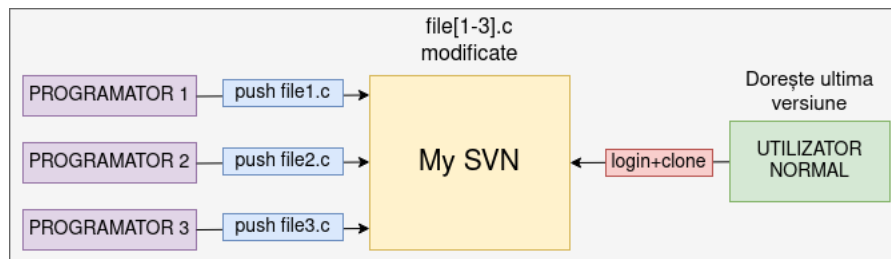


Fig. 6. Modul în care interacționează utilizatorii cu aplicația

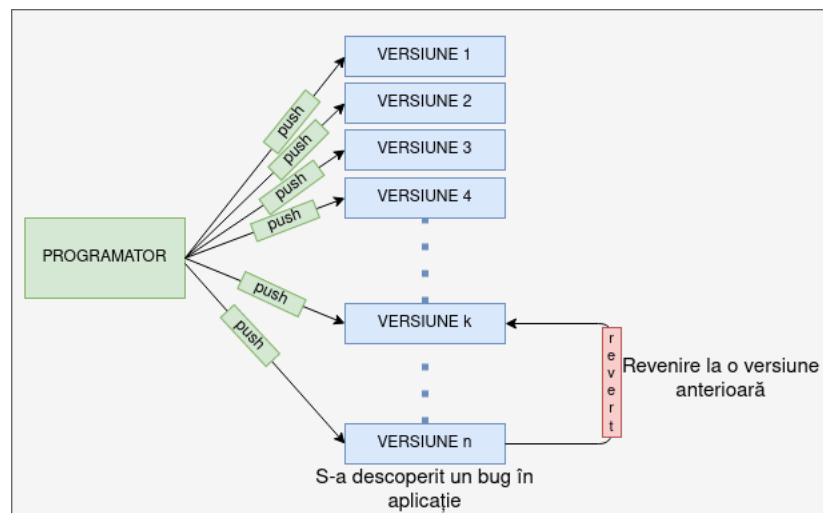


Fig. 7. Modul în care erorile sunt rezolvate într-o aplicație

4 Detalii de implementare

4.1 Descriere generală

Am descris în secțiunea **Tehnologii utilizate** dependențele aplicației. Pe partea de server va fi necesară, la inițializare, rularea scriptului `createDataBase.sh` care va compila fișierul C responsabil de crearea bazei de date. După rularea executabilului `server` utilizatorii vor avea posibilitatea de a executa comenzi prin intermediul clientului dacă au permisiuni, desigur.

În continuare voi descrie flow-ul general al aplicației. Este important de precizat că realizarea funcționalităților descrise este dependentă de crearea unor directoare și fișiere de server/client după cum urmează:

- `.repo_config` - fișier ce va deține informații despre repository, nume, versiune locală, ultimul commit hash, vârsta în unix time, etc;
- directorul curent al clientului este directorul `dirty` în care se vor putea descărca fișiere și din care se vor muta fișierele spre staging area;
- `.untouched` - director în directorul curent al clientului în care se va afla versiunea descărcată fără nicio modificare;
- `.staging` - director în directorul curent al clientului care inițial va conține sursele versiunii din `.untouched`, apoi va suferi modificări prin add/remove. Tot în acest folder, există un fișier numit `.changelog` la care se dă append de commit messages;

Comunicarea este criptată folosind protocolul `three pass protocol`.

Detalii de implementare a comenzilor

login(indirectă): Utilizatorul introduce credențialele. Acestea sunt trimise serverului care verifică baza de date: doar în cazul în care sunt corecte utilizatorul va putea realiza acțiuni de write pe repository public sau read and write pe repository private.

register: Utilizatorul introduce noile detalii de conectare. Acestea sunt trimise serverului care verifică baza de date: dacă există deja un utilizator cu username-ul dat atunci o eroare va fi trimisă clientului altfel operația se va termina cu succes iar clientul se va putea conecta.

init: Utilizatorul introduce un nume de repository pentru a fi creat. Se va crea un fișier JSON numit `.repo_config` ce va avea doar un câmp: `repository_name` cu valoarea dată.

clone: Utilizatorul introduce numele repository-ului pe care dorește să îl cloneze. Dacă există pe server în baza de date acesta va fi transmis clientului și mai apoi se vor crea directoarele `.staging` și `.untouched` și fișierul `.repo_config`.

pull: Utilizatorul apelând această comandă va verifica existența unor versiuni noi. În cazul în care există versiuni noi acestea vor fi descărcate, directoarele `.untouched` va fi resetat iar conținutul directorului curent va fi copiat într-un director `old`, conținutul său va fi resetat mai apoi și conținutul directorului `.staging` va fi copiat într-un director `staging-old`, conținutul său va fi resetat mai apoi. Va fi actualizat și fișierul `.repo.config`.

push: Utilizatorul apelând această comandă va marca încercarea de a încărca o versiune nouă pe server. Serverul îi va transmite numele versiunii disponibile(cea de pe server+1) și va încărca fișierele în tabela de date corespunzătoare, va introduce o nouă înregistrare în tabela versiunilor și va introduce mesajele de commit corespunzător.

reset: Utilizatorul va reseta atât folderul curent(`dirty`), cât și cel de `.staging` cu ajutorul folderului `.untouched`. Nu se fac cereri la server, nu se face conexiune cu el, nu este nevoie.

checkout-file: Utilizatorul selectează un fișier pe care dorește să-l descarce. Se trimite numele fișierului și numele repository-ului la server(eventual și versiunea) și se face query în baza de date, dacă există acel fișier în repository-ul respectiv și, eventual, la versiunea respectivă se va face transfer de la server la client pentru acel fișier. Dacă nu este specificată versiunea va fi considerată versiunea cea mai recentă de pe server pentru repository-ul respectiv.

message: Utilizatorul dă append de mesaje de commit în fișierul `.changelog` după bunul plac. Nu se poate realiza fără nicio modificare făcută asupra directorului `.staging`.

checkout: Utilizatorul apelând această comandă va verifica existența unei anumite versiuni. În cazul în care există versiunea specificată aceasta va fi descărcată, directorul `.untouched` va fi resetat iar conținutul directorului curent va fi copiat într-un director `old`, conținutul său va fi resetat mai apoi iar conținutul directorului `.staging` va fi copiat într-un director `staging-old`, conținutul său va fi resetat mai apoi. Va fi actualizat și fișierul `.repo.config`.

stage: Utilizator lucrează în directorul curent și când e sigur de o schimbare apelează comanda `stage` care adaugă fișierul în directorul `.staging`(cu suprascriere).

unstage: Utilizatorul a decis să marcheze ștergerea unui fișier. Apelând această comandă se va șterge fișierul din directorul `.staging`.

delete: Utilizatorul a decis să marcheze ștergerea unui fișier din folderul `.untouched`. Apelând această comandă se va șterge fișierul din directorul `.staging` și marchează fișierul ca șters în fișierul `.marked_as_deleted`.

restore: Utilizatorul a decis că schimbările asupra unui fișier nu sunt pe placul său. Apelând această comandă fișierul va reveni la starea copiei din directorul `.untouched` și în cazul în care a fost șters acesta va fi, de asemenea, scos din fișierul `.marked.as.deleted`.

list-staged: Se vor afișa numele fișierelor care au suferit modificări prin listarea directorului `.staging`.

list-remote: Se va face request la server pentru lista de fișiere curentă. Se va citi fișierul `.repo_config` pentru detalii despre repository-ul în cauză.

list-untouched: Se vor afișa numele fișierelor din directorul `.untouched`.

list-dirty: Se vor afișa numele fișierelor care au suferit modificări prin compararea directoarelor curent(fără a lua în calcul directoarele/fișierele ajutătoare) și `.staging`.

diff-file: Se va trimite serverului numele fișierului și versiunea repository-ului. Dacă versiunea nu este specificată va fi considerată cea mai recentă versiune a repository-ului de pe server. Se va face query în baza de date și în cazul în care există fișierul cu condițiile specificate va fi trimis conținutul său la client care va crea un fișier auxiliar de care se va ajuta pentru a face `diff` între fișierul curent și cel de pe server.

diff-version: Se va trimite serverului numele fișierului și versiunea repository-ului. Se va face query în baza de date și în cazul în care repository-ul și versiunea există va fi trimis un JSON ce deține schimbările față de versiunea imediat precedentă.

go-public: Se va trimite serverului numele utilizatorului, parola și numele repository-ului. Se va face query în baza de date și în cazul în care repository-ul există și aparține utilizatorului dat iar credențialele sunt corecte repository-ul va fi făcut public.

go-private: Se va trimite serverului numele utilizatorului, parola și numele repository-ului. Se va face query în baza de date și în cazul în care repository-ul există și aparține utilizatorului dat iar credențialele sunt corecte repository-ul va fi făcut private.

block-edit: Se va trimite serverului username-ul căruia utilizatorul dorește să îi înlăture permisiunile pentru repository-ul curent. Dacă utilizatorul deține repository-ul curent atunci operația se va termina cu succes iar utilizatorul cu username-ul specificat va pierde drepturile de modificare.

allow-access: Se va trimite serverului username-ul căruia utilizatorul dorește să îi dea acces pentru repository-ul curent. Dacă utilizatorul deține repository-ul curent atunci operația se va termina cu succes iar utilizatorul cu username-ul specificat va avea drepturi de acces.

block-access: Se va trimite serverului username-ul căruia utilizatorul dorește să îi înlăture accesul pentru repository-ul curent. Dacă utilizatorul deține repository-ul curent atunci operația se va termina cu succes iar utilizatorul cu username-ul specificat va pierde drepturile de acces.

get-changelog: Se va trimite serverului numele repository-ului și numele versiunii. În cazul în care cele 2 există se va trimite clientului changelog-ul atașat acelei versiuni pentru repository-ul specificat.

Un detaliu semnificativ implementării descrise mai sus este că multe informații sunt luate din fișierul `.repo_config` și din contextul repository-ului. De aceea, au mai existat descrieri unde am omis precizarea numelui repository-ului, spre exemplu.

Conectarea este condiționată de vizibilitatea repository-ului, dacă acesta este private se vor cere credențialele la orice comandă ce necesită conexiune, altfel numai pentru push, go-public, go-private, allow-access, allow-edit, block-access, block-edit.

4.2 Pseudocod sugestiv

În continuare, voi descrie în pseudocod secțiunile cele mai folosite din cod, cele omise fiind observate în cod cu comentariile de rigoare alături(dacă e cazul).

Algorithm 1 Funcția main client

```

1: procedure MAIN
2:   clientSD = createSocket();
3:   connect(...);
4:   parse_received_command();
5:   treat_received_command();
6:   get_server_response();
7:   apply_actions();
8: end procedure

```

Algorithm 2 Funcția main server

```

1: procedure MAIN
2:   serverSD = createSocket();
3:   listen(serverSD, NOMAXCLIENTS);
4:   while (1) do
5:     clientSD = accept(serverSD, ...);
6:     serveClient(clientSD);
7:     close(clientSD);
8:   end while
9: end procedure

```

Algorithm 3 Funcția de tratare a clientului

```

1: procedure SERVECLIENT( )
2:   pid = fork();
3:   if pid == 0 then
4:     command = receiveData(serverSD, serverKey);
5:     check_permissions_and_ask_for_login_if_needed();
6:     if requires_data_transfer then
7:       newSocket = create_new_socket();
8:       sendDataUsingNewSocket(data, newSocket);
9:       close(serverSD);
10:    else
11:      data = compute_response(command);
12:      sendData(response);
13:    end if
14:  end if
15: end procedure

```

5 Concluzii

Aplicațiile de tip SVN sunt indispensabile programatorilor. Aplicația descrisă anterior asigură strictul necesar pentru a putea versiona anumite proiecte. Aceasta poate fi extinsă pentru a putea avea **branches**, posibilitatea de a aproba/refuza schimbările care se doresc a fi încărcate de anumiți utilizatori, etc.

De asemenea, se poate implementa o variantă a comunicării între server și client criptată și construi o interfață grafică prietenoasă care să minimizeze lucrul la linia de comandă.

În concluzie, aplicația îndeplinește funcționalitățile minimale ale unei aplicații din categoria version control system. Principiile de securitate și eficiență sunt satisfăcute la un nivel minimal.

References

1. Cursul de "Rețele de calculatoare" de la Facultatea de Informatică Iași, materie predată de Alboae Lenuța și Panu Andrei, <https://profs.info.uaic.ro/~computernetworks/>
2. Cursul de "Sisteme de operare" de la Facultatea de Informatică Iași, materie predată de Vidrașcu Cristian, <https://profs.info.uaic.ro/~vidrascu/SO/>
3. Cursul de "Baze de date" de la Facultatea de Informatică Iași, materie predată de Vârlan Cosmin și Breabăn Mihaela, <https://profs.info.uaic.ro/~vcosmin/bd>
4. Cursul de "Structuri de date" de la Facultatea de Informatică Iași, materie predată de Gațu Cristian și Răschip Mădălina, <https://profs.info.uaic.ro/~sd/>
5. LNCS Homepage, <http://www.springer.com/lncs>
6. LNCS Latex templates, <https://www.overleaf.com/project/5fd4d0875e1b6900e83d1f07>
7. How to add indentation in Latex?, <https://tex.stackexchange.com/questions/45501/how-to-add-indentation>
8. How to not use random() in C?, <https://stackoverflow.com/questions/8624997/random-number-generation-in-c-using-mersenne-twister>
9. Diff Bash man page, <https://man7.org/linux/man-pages/man1/diff.1.html>
10. Wiki Caesar cipher, https://en.wikipedia.org/wiki/Caesar_cipher
11. Image Three way handshaking, <https://www.9tut.com/tcp-and-udp-tutorial>
12. Inserting images in latex, https://www.overleaf.com/learn/latex/Inserting_Images
13. Insert algo in latex, <https://codeyarns.com/tech/2012-06-29-inserting-an-algorithm-in-latex.html>
14. SQLite browser, <https://sqlitebrowser.org/>
15. Use SQLite in C, <http://zetcode.com/db/sqlitec/>
16. Draw diagrams online, <https://app.diagrams.net/>
17. Inkscape vector graphics, <https://inkscape.org/>
18. JetBrainsMono font, <https://github.com/JetBrains/JetBrainsMono>
19. Agave font, <https://github.com/blobject/agave>
20. Wiki Three pass protocol, https://en.wikipedia.org/wiki/Three-pass_protocol
21. Git homepage, <https://git-scm.com/>
22. Mercurial project mirror, <https://github.com/indygreg/hg>

- 23. Turtle homepage, <https://tortoisesvn.net/>
- 24. Turtle project, <https://gitlab.com/tortoisegit/tortoisegit/>
- 25. Overleaf, <https://www.overleaf.com>
- 26. Tectonic system, <https://tectonic-typesetting.github.io/en-US/>
- 27. MySVN Andrei Luca cu mențiunea că a fost folosit ca research only,
<https://github.com/andreiluca96/MySVN---Computer-Networks-Final-Project>
- 28. CMake documentation, <https://cmake.org/documentation/>
- 29. Parson project, <https://github.com/kgabis/parson>
- 30. Awesome C database, <https://github.com/oz123/awesome-c#database>
- 31. JSON docs, <https://www.json.org/json-en.html>
- 32. Apache subversion, <https://subversion.apache.org/>
- 33. Wiki Databases, <https://en.wikipedia.org/wiki/Database>