Progres raport my_ssh tema 2

Rotariu Dragos-Irinel

January 15, 2025

1 Introducere proiect

Proiectul "my_ssh" se bazeaza pe pereche client/server care este capabila de autentificare si comunicare encriptate. Clientul va putea trimite comenzi (login, comanda terminal,quit) serverului, iar serverul or proceseaza logarea (verifica o baza de date) or executa comanda (daca e logat) si trasmite rezultatul executii comenzi clientului (la fel ca terminalul de linux).
Fiecare client (pot fi mai multi de unul) poate trasmite comenzi serverului prin tastarea lor la consola urmat de un *enter* . Comenzile pot fi de tipul *login* ,*quit* sau o comanda de terminal (linux). Serverul primeste linia scrisa de client, si,daca are drepturi (este logat), atunci el trebuie sa execute comanda si sa raporteze rezultatele comenzi la client.
Pentru a se loga, clientul trebuie sa scrie *login user_name password* iar serverul verifica daca perechea de argumente date este intr-o baza de date . La confirmare ca este, serverul ii va confirma clientului ca este logat (la neexistenta combinatiei, serverul il va avertiza pe client).
In final clientul poate iesi din server la tastarea comenzi *quit*.

2 Tehnologii Aplicate

I. Socket-uri *BSD*
Comunicarea intre client si server s-a realizat cu ajutorul unui socket. Socketul ofera o conexiune directa intre cele doua programe si permite comunicarea bidirectionala intre client si server (care este necesara incat clientul ii scrie serverului comanda, serverul citeste comanda apoi ii trimite output-ul generat inapoi la client) crearea a mai multe cai de comunicare one way (exemplu socket pair or fifo file) sunt mai grele de gestionat. (O sa fie refolosit socket-ul deci o sa avem nevoie de reuse addr)
II. server *TCP* concurent
Serverul este de tip *TCP* concurent, motivul implementarii unui server *TCP* (si nu *UDP*) este simplu: 1. fluxul de informatii input output intre server si client nu este atat de ridicat comparat cu aplicatii de genul retele de socializare, in plus comenzile trimise si raspunsurile primite pot fi de o natura importanta (afisezi informatii cu cat, modifici drepturile unui document) iar

tcp imi asigura faptul ca informatiile care sunt trasmise intre server si client sunt corecte si exista o conexiune sigura intre cele doua. Totusi, pentru a asigura mai ca mai multi clienti pot sa fie conectate in acelasi timp si sa poata trimite serverului comenzi, din aceasta cauza, serverul este concurent.
III. thread-uri (pthreads)
Pentru fiecare client este creat un thread, care ofera intr-un timp rezonabil (mai rapid decat orice preforked server) outputul comenzilor trimise din client.
IV. fork() si execvp()
Pentru a executa comanda, o sa am nevoie de execvp() specific incat nu stiu cate argumente o sa aiba fiecare comanda si folosind un vector in afara de o lista este mai potrivit. O sa creeam un proces copil cu fork() pentru fiecare (sub)comanda unde vom folosii execvp.
V. sqlite
Datele privind utilizatorii si parolele lor vor fi valabile intr-o baza de date. Pentru a interoga tabelul o sa avem nevoie de sqlite (baza de date, login si logout vor fi implementate mai tarziu).
VI. mutex?
Incat exista posibilitate ca mai multi clienti sa incearca sa modifice un document in acelasi timp, este luat in considerare folosirea a mutual exclusive locks (mutex), momentan nu a fost implemenant mutex pe threaduri incat exista unele comenzi (login,logout) care nu depind in mod direct de computer (nu trebuie mutex).

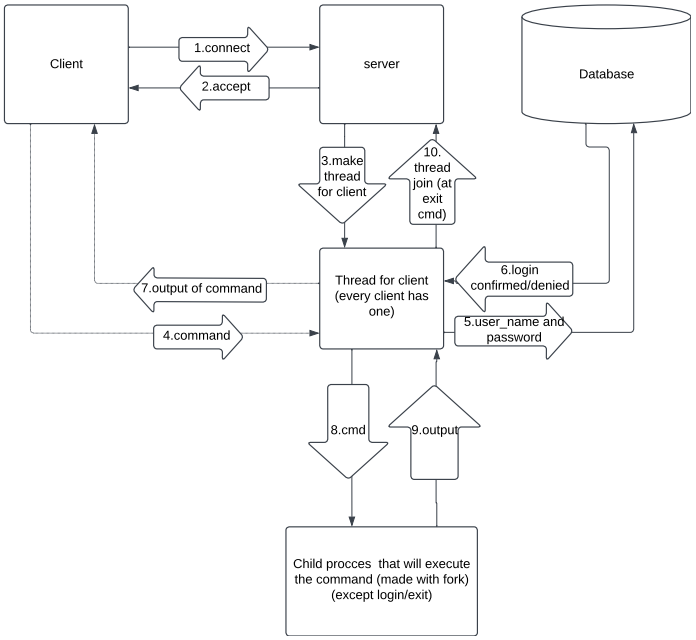


Figure 1: Diagrama proiect

3 Structura Aplicatiei

La deschiderea clientului, el va incerca sa se conecteze(1) la server. Serverul ii accepta incercarea de conectare (2) si ii va creea un thread (2) clientului. Clientul apoi trimite serverului (mai precis threadului) comnenzi (4) prin consola , o prima comanda fiind *login* care este de tip $login\ user_name\ password$. La primirea comenzii *login* impreuna cu argumenzii sai, server(threadul clientului din server) va verifica in baza de date(5) si apoi o sa-i confirme (6) daca el s-a logat sau nu (7).
La alte tipuri de comenzi (nu login,exit) , el va creea un proces copil (8) care va executa comanda trimisa si va returna inapoi outputul comenzi threadului clientul (9) care va fi trasmis clientului (7). La primirea comenzi "exit", threadul isi va termina executia si va raporta serverului printr-un thread Join(10).

4 Aspecte de Implementare

In partea de server, functiile bind, si listen care sunt necesare pentru atasarea unui socket in server si pregatirii conectarii la server sunt puse in functii unice.

```
        void prep_srv()
{
        // creeam un socket care sa primeasca informatii initiale+conectare
    if ((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
    {
        perror ("[server] Eroare la socket().\n");
        kill_srv(errno);
    }
    int on=1;
    setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));
    bzero (&server, sizeof (server));
    bzero (&from, sizeof (from));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl (INADDR_ANY);
    server.sin_port = htons (PORT);
    if (bind(sockfd,(struct sockaddr *)& server, sizeof (struct sockaddr))==-1)
    {
        perror("[server] Eroare la bind().\n");
        kill_srv(errno);
    }
    printf("Bind succesful\n");
    fflush(stdout);
    if (listen (sockfd,6)==-1)
    {
        perror("[server] Eroare la listen().\n");
        kill_srv(errno);
    }
    printf("Sever ready to accept\n");
    fflush (stdout);
}
```

Intr-o maniera asemanatoare a fost pregatit si clientul pentru conexiune.

```
        void prep_cl()
{
    if ((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
    {
        perror ("[server] Eroare la socket().\n");
        kill_cl(fatal_err_cl);
    }
    srv.sin_family = AF_INET;
    char* addr="127.0.0.1";
    srv.sin_addr.s_addr = inet_addr(addr);
    srv.sin_port = htons (PORT);
        // stabilim conexiunea cu serverul
        if (connect (sockfd, (struct sockaddr*) &srv, sizeof (struct sockaddr))==-1)
```

```
        {
            perror("[client] Eroare la connect");
            kill_cl(fatal_err_cl);
        }
    printf("Client pregatit, va rog sa tastati comanda ceruta urmata de de un ENTER\n");
    fflush(stdout);
```

Detectarea clientului este efectuata cu ajutorul unui while (1) si functiei wait_for_client.

```
        void wait_for_client()  //asteptam un client sa se conecteze
{
    //accept()
    socklen_t ln=sizeof(from);
    int client;
    if ( (client = accept (sockfd, (struct sockaddr *) &from,(socklen_t *)&from)) < 0)
     {
     perror ("[server]Eroare la accept().\n");
            return;
        }
    fflush(stdout);
    thr_cl(client);//acceptam si trecem la crearea unui thread pentru el
}
```

Vor fi implementate functiile kill_srv, kill_cl si kill_thr pentru inchiderea fortata a server, client or thread la intalnirea unei erori letale.

```
        void kill_cl(int err)
{
    printf("IDENTITY CRISIS");  //will be added later
    fflush(stdout);
    return;
}
void kill_srv (int er)
{
    //fortam inchidere eroare fatala din server
    // will be added later
}
void kill_thr (int er)
{
    //fortam inchidere eroare letala thread
    //will be added later
}
```

Impartirea programelor in functii specifice ne ajuta sa intelegem mai usor piesele din cod.

```
    main din client
        int main()
{
    prep_cl();
    while (1)
    {
       int rp=send_cmd();
       if(rp==quitt)
       {
            quit_cl();
            return 0;
       }
       else
       {
            show_rst(rp);
       }
    }
}
```

5 Concluzia initiala

In acest moment este facut doar partea de conexiune intre client si server si testarea fluxului de input=output . Mai trebuie adaugat mai multe lucruri pecrum o functie separata pentru login, mutex, encriptia, si partea de executie a comenzii care devine mai complicat cand vine vorba de ; & si | incat fiecare comanda trebuie impartite in subcomenzi delimitate de ; apoi & apoi | (care trebuie executate in mod diferit).
In plus mai trebuie functii diferite pentru cd si pwd(motivul principal in care exista adresa absoluta a clientului in thData este necesitatea lui pentru acele functii). Totusi sunt destul de confident in ducerea acestui proiect pana la capat.

6 Referinte Bibliografice

Cursurile si codurile oferite de $https://edu.info.uaic.ro/computer-networks/$

7 Bash in depth

Sunt multe intrebari referind unele interatiuni pe care terminalul le are pe care le-am avut, de exemplu ce se intampla daca stergi de pe un window/tab un working directory de la un alt tab? In ce pas ai voie sa redirectionezi inputoutputerror_messages.

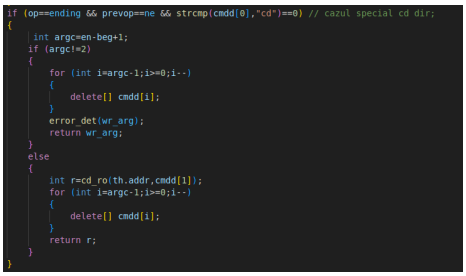
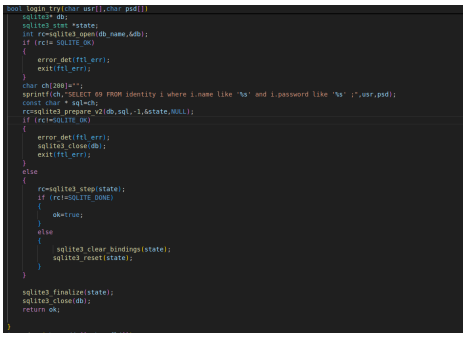


Figure 2: Enter Caption

Raspunsurile la aceste intrebari imi iduc mai multe intrebari. Raspunsul la prima intrebare este ca acum ai un window pe un folder non existent unde unele comenzi merg dar altele ca ls nu merg , raportand o eroare de director neexistent sau incorect. Terminalul nu te forteaza sa te muti la un director neexistent si (in unele cazuri) nici cd ../ nu o sa mearga daca ai sters si directorul anterior.
In final am decis sa impun o regula stricta , daca wd nu e valid , nu conteaza ce comanda ai apelat, o sa raporteze o eroare de cwd nonexistent si te duce la (sperand nu nonexistent) home directory.
A doua intrebare are un raspuns care poate duce la foarte mari probleme. Redirectionarea se poate face la nivelul de executie (dupa parsarea de pipes | &).
Pare initial o non problema....pana cand vezi interactia intre redirectionari si | ... Daca redirectionezi stdout la o comanda care se afla oriunde (in afara de capatul pipeline-ului), efectivi tai pipeline-ul incat ai stdout trebuia sa fie stdin pentru urmatoarea comanda care acum se duce la fisier (si numai la fisier), nu se dubleaza raspunsul (adica nu poti pastra pipeline-ul si sa afisezi un rezultat intermediar).
Ceva asemanator se intampla si daca redirectionezi stdin-ul la o comanda care nu e prima in pipeline. Nu o sa mai ai acces la rezultat si ai irosit primele comenzi incat | si produc un proces fiu si pipeline-ul este (posibil) asigurat cu pipes si dup.
Acest lucru e complet inutil si as fi dorit sa fi pus o restrictie sa nu poti face asta dar, din pacate, este posibil in codul meu sa redirectionezi stdout si stderr incat pipes sunt create la nivel de analizare, nu executie.
Destul de amuzant, faptul ca la proccess command (zona de parsare | si) trebuie sa apelezi un proces fiu creeaza o interactie speciala cu login si cu cd. Directorul curent al clientului impreuna cu starea clientului (logged in or not logged in) nu este salvata in procesul parinte (Daca va intrebati, DA, terminalul face exact acelasi lucru cand pui intr-un pipeline cd, o sa raporteze erori daca exista dar nu salveaza modificarile facute) deci trebuia sa fac un salt special pentru cd si login try .
Mai sunt si alte lucruri care au facut proiectul mai putin potrivit (un alt exemplu este comanda false care ar trebui sa simuleze o comanda esuata dar



```
bool login_try(char usr[],char psd[])
{
    sqlite3* db;
    sqlite3_stmt *state;
    int rc=sqlite3_open(db_name,&db);
    if (rc!= SQLITE_OK)
    {
        error_det(ftl_err);
        exit(ftl_err);
    }
    char ch[200]="";
    sprintf(ch,"SELECT 69 FROM identity i where i.name like '%s' and i.password like '%s' ;",usr,psd);
    const char * sql=ch;
    rc=sqlite3_prepare_v2(db,sql,-1,&state,NULL);
    if (rc!=SQLITE_OK)
    {
        error_det(ftl_err);
        sqlite3_close(db);
        exit(ftl_err);
    }
    else
    {
        rc=sqlite3_step(state);
        if (rc!=SQLITE_DONE)
        {
            ok=true;
        }
        else
        {
            sqlite3_clear_bindings(state);
            sqlite3_reset(state);
        }
    }

    sqlite3_finalize(state);
    sqlite3_close(db);
    return ok;
}
```

Figure 3: the login function

execvp raporteaza ca o comanda care s-a terminat cu succes). Si probabil sunt alte interactiuni care le-am ratat.

8 sqlite3

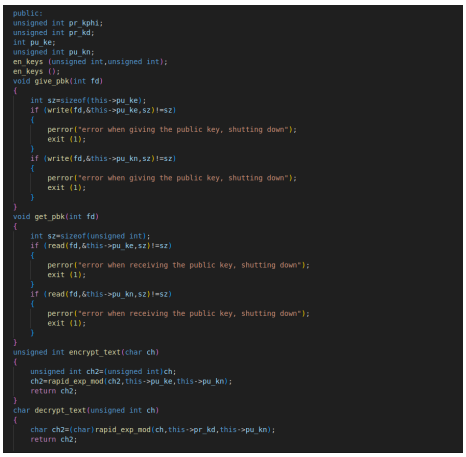
Pentru functia de logare a clientului , am folosit o data de baze db care contine mai multe usernames si parolele asociate cu ele . Functia de logare deschide baza de date, pregateste un select care ia un lucru unde numele si parola sunt la fel ca ? (variabile care o sa punem ulterior). Apoi daca a raporta ca exista o line atunci exista o persoana care are numele si parola la fel ca inputul dat.

9 encriptie

Comunicarea intre client si server trebuia sa fie secura deci am folosit creat un rsa key pentru client si pentru server si am scris de mana functiile de encriptie (cu ajutor in referire cu crearea lui d incat nu mergea codul clasic de euclid extins).
Structul en_keys creeaza elementele de encriptie (e si n) precum si elementul de decriptie (d) pe baza a doua numere prime.
Incat aplicatia deja este destul de straining din punct de vedere a timpului pentru a inregistra si procesa comenzi, am optat sa folosesc unsigned int in loc de ceva mai mare iar numerele prime alese sunt mici incat nu am vrut ca n (care e produsul nr prime) sa depaseasca limita pentru unsigned int.

10 concluzie finala

Proiectul parea usor dar unele lucruri s-au dovedit foarte complicate, totusi, apreciez foarte mult faptul ca am invatat mai in detaliu cum merge terminalul



```
public:
unsigned int pr_kphi;
unsigned int pr_kd;
int pu_ke;
unsigned int pu_kn;
en_keys (unsigned int,unsigned int);
en_keys ();
void give_pbk(int fd)
{
    int sz=sizeof(this->pu_ke);
    if (write(fd,&this->pu_ke,sz)!=sz)
    {
        perror("error when giving the public key, shutting down");
        exit (1);
    }
    if (write(fd,&this->pu_kn,sz)!=sz)
    {
        perror("error when giving the public key, shutting down");
        exit (1);
    }
}
void get_pbk(int fd)
{
    int sz=sizeof(unsigned int);
    if (read(fd,&this->pu_ke,sz)!=sz)
    {
        perror("error when receiving the public key, shutting down");
        exit (1);
    }
    if (read(fd,&this->pu_kn,sz)!=sz)
    {
        perror("error when receiving the public key, shutting down");
        exit (1);
    }
}
unsigned int encrypt_text(char ch)
{
    unsigned int ch2=(unsigned int)ch;
    ch2=rapid_exp_mod(ch2,this->pu_ke,this->pu_kn);
    return ch2;
}
char decrypt_text(unsigned int ch)
{
    char ch2=(char)rapid_exp_mod(ch,this->pr_kd,this->pu_kn);
    return ch2;
```

Figure 4: Encryption

la linux si am incercat sa fac stilul meu de programare mai curat. sper sa fie bun.