Proiect - MySSH

Ciudin Ștefana

Universitatea "Alexandru Ioan Cuza", Facultatea de Informatică
stefanaciudin@info.uaic.ro

Abstract. Documentație pentru proiectul MySSH, conținând diverse informații relevante, precum arhitectura aplicației, tehnologiile utilizate și detalii de implementare.

1 Introducere

1.1 Proiectul ales

Proiectul pe care am ales să îl implementez este MySSH.
Cerința acestuia este : Să se implementeze o pereche client/server capabilă de autentificare și comunicare encriptate. Server-ul va executa comenzile de la client, și va returna output-ul lor clientului. Comenzile sunt executabile din path, cu oricât de multe argumente; cd și pwd vor funcționa normal. Se pot executa comenzi multiple legate între ele sau redirectate.

1.2 Motivul alegerii

Am ales să implementez acest proiect deoarece mi se pare foarte interesant și l-am considerat o provocare, deoarece nu am mai realizat nimic asemănător până acum. De asemenea, prin intermediul acestuia, am considerat că voi putea aprofunda noțiunile învățate pe parcusrul semestrului la această materie.

1.3 Funcționalitatea aplicației

Aplicația client rulează într-un terminal, scopul acesteia fiind de a primi comenzi de la utilizator, pe care le trimite serverului. Pot rula mai mulți clienți simultan. Serverul creează câte un proces copil pentru fiecare client conectat - copilul va procesa și executa comenzile primite de la client, după care va trimite răspunsul comenzii înapoi la procesul părinte, iar serverul îl va trimite înapoi la client, pentru a putea fi afișat de acesta. Serverul se ocupă și de procesul de autentificare, credențialele utilizatorilor fiind stocate într-o bază de date.
Așa cum menționează cerința, comunicarea dintre server și client/clienți se va realiza encriptat.

2 Tehnologiile utilizate

2.1 Protocolul de comunicare

Pentru comunicarea dintre clienți și server, am ales socket-uri TCP, deoarece acesta este un protocol orientat conexiune și de încredere, care garantează că informația transmisă de la client la server va fi completă și nu vor exista duplicate.
Caracteristicile protocolului TCP (retransmiterea pachetelor; nu există pachete pierdute și reasamblarea pachetelor la destinație) garantează o comunicare eficientă între client și server - nu am dori ca, de exemplu, să se piardă informație în procesul de comunicare, astfel nu s-ar mai putea executa corect comenzile introduse de utilizator sau rezultatul afișat de client nu ar fi cel real. Pentru a păstra securitatea procesului de comunicare, acesta va fi encriptat folosind funcțiile din *OpenSSL*.

2.2 Alte mențiuni

Pentru a crea procesele copil din server, se utilizează *fork()*. Pentru comunicarea dintre server și procesele copil ale acestuia vor fi utilizate *pipe-uri*. Pentru prelucrarea comenzilor introduse de utilizator se utilizează funcțiile pentru prelucrarea șirurilor de caractere din *string.h* (cum ar fi strtok, strlen, strcpy, strchr, etc).

3 Arhitectura aplicației

Aplicația este alcătuită dintr-o pereche de componente - clientul și serverul.

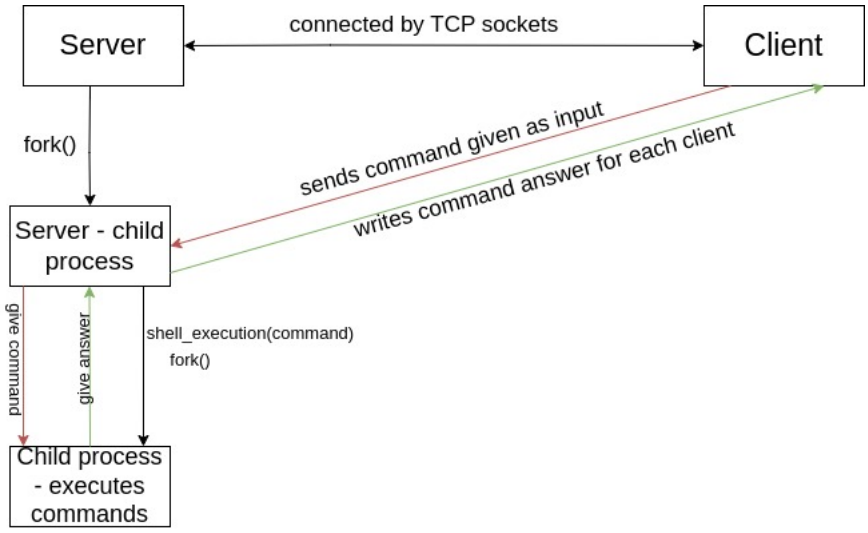
3.1 Serverul

Inițial, serverul va asculta la un anumit port, pentru a aștepta conexiuni de la clienți. Pentru fiecare client conectat se va crea câte un proces copil - aici se va realiza prelucrarea comenzilor introduse de utilizator, execuția acestora, și returnarea răspunsului fiecărei comenzi în parte înapoi la clienți. Ulterior, serverul va aștepta noi conexiuni, de la noi clienți.
Serverul are acces la o bază de date care conține două tabele: tabela *Users*, unde sunt reținute date precum utilizatorul și parola acestuia(asupra parolei se aplică o funcție hash înainte de a fi stocată, pentru a nu fi salvată în forma sa reală) și o tabelă *Login*, unde se reține pid-ul fiecărui client care s-a conectat și dacă există sau nu un utilizator logat în clientul respectiv. Cu fiecare client care se conectează, tabela *Login* este actualizată - se introduce pid-ul clientului și valoarea 0 în câmpul logged_status al tabelei, pentru a evidenția că momentan nu este conectat un utilizator.

3.2 Clientul

Clientul comunică cu serverul printr-un socket de tip TCP.
După conectarea la server, clientul îi va trimite acestuia comenzile primite ca input de la utilizator. De asemenea, clientul are rolul și de a afișa răspunsul comenzilor executate.
Atât clientul cât și serverul vor comunica între ele encriptat - acest procedeu va fi asigurat prin utilizarea funcțiilor din *OpenSSL*.

3.3 Schema aplicației



4 Detalii de implementare

Prin introducerea comenzii pwd, se va afișa calea directorului de lucru.

În cazul serverului, pentru a putea permite conectarea mai multor clienți simultan, serverul va crea câte un proces copil pentru fiecare client.

```
while (run)
{

    int length = sizeof(from);
    client = accept(sd, (struct sockaddr *)&from, &length)
        ;
    if (client < 0)
    {
        perror("[server] error - accept \n");
        continue;
    }
    pid_t pid;
```

```
    //OpenSSL specific functions
    if ((pid = fork()) == -1)
    {
        perror("[server] error - fork \n");
        return 1;
    }
    if (pid == 0) // child process
    {
        printf("New client connected \n");
        close(sd);
        while (1)
        {
            int val;
            val = read(client, cmd_received, MAX_COMMAND);
            cmd_received[val] = '\0';
            shell_loop(cmd_received);
        }
    }
}
```

Funcția *shell_loop(command)* are rolul de a controla comenzile primite de la client. Prin intermediul acesteia, se realizează împărțirea în argumente a comenzilor și executarea acestora. Comenzile vor fi executate cât timp, o variabilă de tip *int*, denumită *status*, are valoare nenulă - variabila status dobândește valoarea 0 atunci când utilizatorul introduce comanda *exit*, care întrerupe automat activitatea clientului.
Serverul stochează informații într-o bază de date cu două tabele - *Users* și *Login*. În tabela *Users*, vor fi stocate numele utilizatorului, parola acestuia (asupra căreia se aplică o funcție hash), și salt-ul utilizat pentru obținerea hash-ului parolei. Pentru a obține hash-ul parolei, se folosește funcția crypt(), care ia ca argument parola și salt-ul. De fiecare dată când se introduce un nou user și o nouă parolă în baza de date, de va genera un nou salt random de către funcția *generate_salt()*:

```
char *generate_salt()
{
    char *salt = malloc(SALT_LENGTH + 1);
    if (salt == NULL)
    {
        perror("malloc");
        return NULL;
    }
    srand(time(NULL));
    salt[0] = '$';
    salt[1] = '5';
    salt[2] = '$';
    for (int i = 3; i < SALT_LENGTH - 2; i++)
        salt[i] = 48 + rand() % 10; //generate a number
```

```
    salt[SALT_LENGTH - 2] = '/';
    salt[SALT_LENGTH - 1] = '$';
    salt[SALT_LENGTH] = '\0';

    return salt;
}
```

Când un utilizator deja existent se va loga în aplicație, asupra parolei introduse de acesta de la tastatura se aplică funcția hash cu salt-ul stocat în baza de date pentru utilizatorul respectiv - dacă hash-ul obținut coincide cu cel stocat, se realizează conectarea. La conectarea unui client nou la server, se introduce un nou rând în tabela *Login* - primul câmp reprezintă pid-ul procesului, iar al doilea marchează dacă este conectat sau nu un utilizator în aplicație. Când un utilizator se va deconecta folosind funcția *logout*, pid-ul procesului curent este căutat în baza de date iar câmpul logged_status este setat pe 0, ceea ce marchează deconectarea utilizatorului.
Comunicarea dintre client și server este securizată folosind funcții din API-ul *OpenSSL* - la baza procedeului stă un fișier de tip certificat cu numele *mycert.pem*; în absența acestuia securizarea nu poate fi realizată și serverul nu va intra în execuție, afișând un mesaj de eroare.

5 Concluzii

Aplicația actuală oferă toate funcționalitățile cerute în enunț, însă poate fi îmbunătățită.
O posibilă îmbunătățire ar fi eliminarea pid-urilor clienților din baza de date, după ce aceștia se deconectează de la server folosind funcția *exit*, pentru a salva memorie. Așa cum a fost menționat anterior, pentru a putea permite conectarea simultană a mai multor clienți la server, pentru fiecare client conectat se va crea câte un proces copil în server printr-un apel *fork()*. Acest procedeu este unul destul de costisitor și ar putea fi îmbunătățit prin utilizarea firelor de execuție - fiecărui client să îi corespundă câte un fir de execuție.

References

1. Shell în C: https://brennan.io/2015/01/16/write-a-shell-in-c/
2. Shell în C: https://www.geeksforgeeks.org/making-linux-shell-c/
3. Informații despre proiectarea modelului de client/server TCP: https://profs.info.uaic.ro/ computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf
4. Informații generale despre protocolul TCP: https://en.wikipedia.org/wiki/Transmission_Control_Protocol
5. Informații generale despre SSH: https://en.wikipedia.org/wiki/Secure_Shell
6. Informații despre OpenSSL: https://opensource.com/article/19/6/cryptography-basics-openssl-part-1
7. SSL server-client: https://aticleworld.com/ssl-server-client-using-openssl-in-c/
8. SQLite: https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm