Tema 2: Documentatie Password Manager

Petrache Andrei Liviu 2A3

Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza", Strada General Berthelot, nr. 16, Iași, România secretariat@info.uaic.ro
https://www.info.uaic.ro

1 Pagina de github a proiectului

https://github.com/liviuq/QuizzGame

- A se ignora numele proiectului
- initial am inceput sa fac QuizzGame dar Password Manager este mai interesant.
- Proiectul este licentiat sub GNU GPL v3.0

2 Introducere

Password Manager este o aplicatie de tipul client/server pentru managementul diferitelor parole ale userilor inregistati. Aceasta aplicatie ofera utilizatorilor o modalitate de logare/inregistrare pe baza de user:password, se pot adauga site-uri impreuna cu parola aferenta, se pot cere toate parolele sau se pot sterge unele campuri.

3 Tehnologii utilizate

Aplicatia se foloseste de protocolul TCP pentru a realiza schimbul de informatii intre client si server. Am ales acest protocol deoarece este sigur, primirea datelor fiind acompaniata de un acknowledge(ACK).

Fiindca se doreste ca mai multi utilizatori sa foloseasca aplicatia in acelasi timp, nu putem sa asteptam fiecare utilizator sa isi realizeze operatiile. Asadar am decis sa realizez un server concurrent, astfel incat fiecarei conexiuni noi sa ii fie oferit un fir de executie pe care se vor realiza functiile I/O.

Datorita arhitecturii alese, putem avea un numar "nelimitat" de utilizatori ce realizeaza operatii I/O cu serverul, nefiind delay-uri intre requesturile fiecaruia.

4 Arhitectura aplicatiei

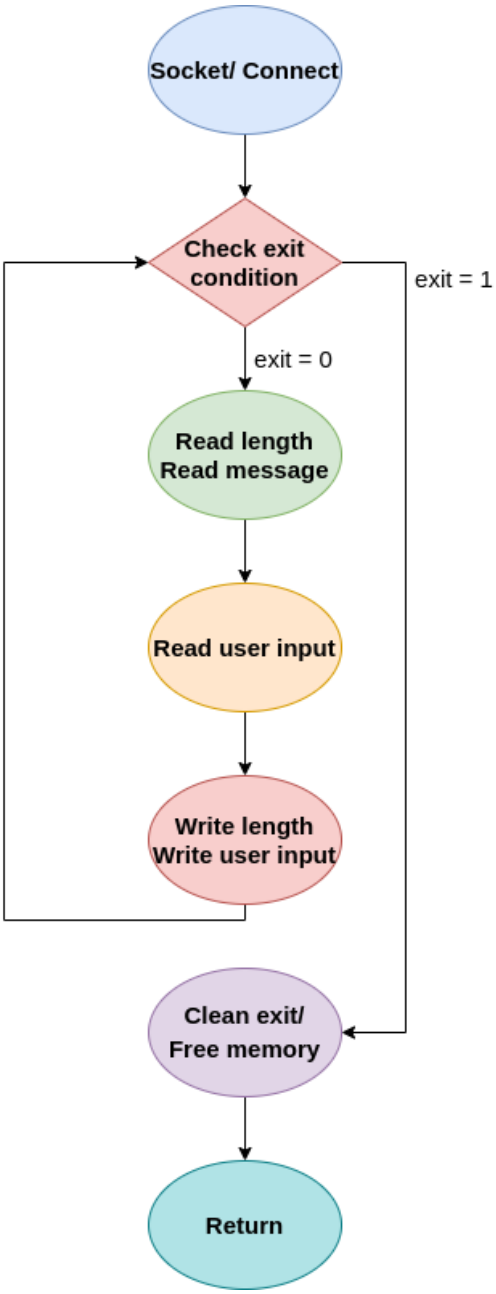

Fig. 1. Flowchart pentru client

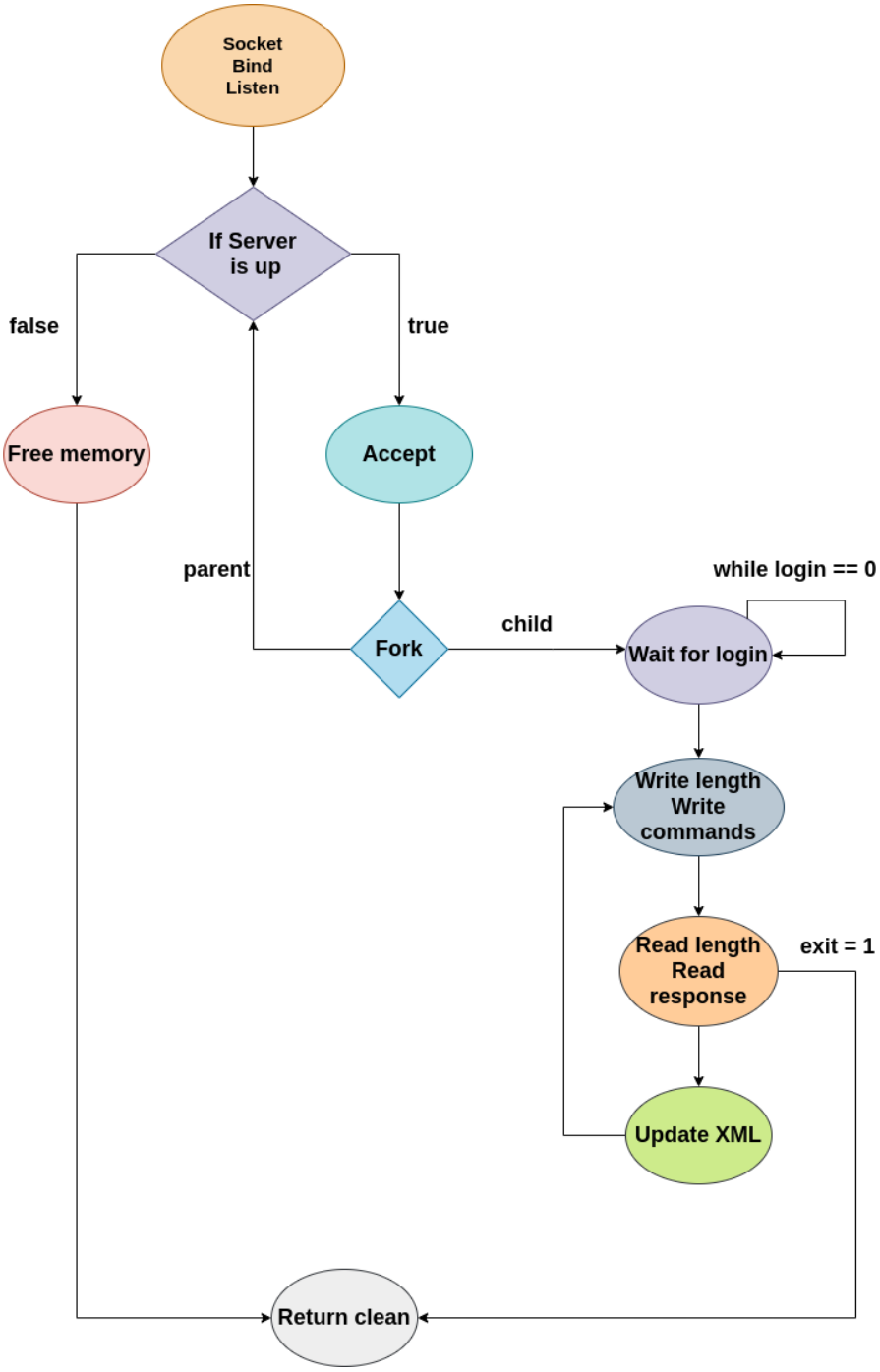

Fig. 2. Flowchart pentru server

5 Concepte implicate

Password Manager contine doua executabile: un client si un server, ambele fiind compilate cu ajutorul comenzii make si a unui fisier Makefile. Scheletul proiectului este compus din mai multe foldere:

- bin: contine binarele returnate dupa compilare + link
- documentation: poze + aceasta documentatie
- include: toate headerele pentru hasing, encryption, alte definitii de functii
- source: toate fisierele sursa
- xml: contine codul sursa si headerele necesare pentru parsarea fisierelor XML

Prin client vom interactiona cu serverul, citind de pe socket 4 octeti ce reprezinta lungimea mesajului de la server, urmand sa citim mesajul intreg. Pe baza informatiilor din mesajul primit, vom citi de la tastatura optiunea dorita si vom trimite lungimea raspunsului si raspunsul actual.

Serverul va realiza toate optiunile de modificare a fisierului cu datele userului. Username-urile vor fi hashed cu ajutorul functiei djb2. Parolele vor fi criptate cu AES, unde key-ul privat va fi Master Password-ul.

Pentru stocarea eficienta a acestor date, voi folosi libraria libxml2 pe care am compilat-o pe unitatea mea si am facut link static pentru ca serverul sa nu aiba dependente externe. Libxml2 este o librarie pentru parsarea fisierelor .xml, facand extragerea si modificarea campurilor mult mai simple si eficiente.

Fiecare fisier XML va avea urmatoarea structura:

```xml
<?xml version="1.0"?>
<user>
   <username>"username_from_client_hashed"</username>
   <masterpass>"master_pass_encrypted"</masterpass>
   <login>"0-logged off/ 1-logged in"</login>
   <title>"website title"</title>
   <url>"url"</url>
   <notes>"miscelanious"</notes>
</user>
```

Fig. 3. Campurile continute intr-un fisier XML

System-urile folosite de catre ambele aplicatii sunt:

- socket() – creeaza un punct terminal pentru comunicare
- bind() – ataseaza un IP unui socket
- connect() – stabilirea conexiunii pe un socket

– listen() – marcheaza socketul ca asteptand conexiuni pana la o anumita limita

– accept() – syscall blocant pana cand se conecteaza un client la server, returnand un file descriptor si IP:PORT

– read()/write() - pentru citirea/scrierea datelor pe socketuri

– close() – inchiderea descriptorilor ramasi deschisi

– free() – eliberarea memoriei de pe heap

6 Detalii de implementare

Clientul va citi lungimea mesajului si mesajul, dupa se va citi de la tastatura raspunsul clientului si se vor trimite lungimea raspunsului si raspunsul:

```
// reading the number of bytes from the server
int32_t length_in = 0;
if (read(sd, &length_in, sizeof(length_in)) == -1)
{
        printf("%s on line %d\n", strerror(errno), __LINE__);
        exit(EXIT_FAILURE);
}
printf("[CLIENT] Length of message is %d\n", length_in);

// reading the whole message
char *incoming_message = malloc(sizeof(char) * length_in);
memset(incoming_message, 0, length_in);

if (read(sd, incoming_message, length_in) == -1)
{
        printf("%s on line %d\n", strerror(errno), __LINE__);
        exit(EXIT_FAILURE);
}

// displaying the message
printf("%s\n", incoming_message);
```

Fig. 4. Citind de la server

In server, dupa ce un client se conecteaza, va intra intr-un while pentru a se loga sau a se inregistra:

```
// reading the input from the user
char *message = malloc(sizeof(char) * LOCAL_MESSAGE_LEN);
memset(message, 0, LOCAL_MESSAGE_LEN);
if (read(STDIN_FILENO, message, LOCAL_MESSAGE_LEN) == -1)
{
        printf("%s on line %d\n", strerror(errno), __LINE__);
        exit(EXIT_FAILURE);
}
```

Fig. 5. Citind de la user

```
// writing to the server the response's length
int32_t length_out = strlen(message);
if (write(sd, &length_out, sizeof(length_out)) == -1)
{
        printf("%s on line %d\n", strerror(errno), __LINE__);
        exit(EXIT_FAILURE);
}

// writing to the server the whole response
if (write(sd, message, length_out) == -1)
{
        printf("%s on line %d\n", strerror(errno), __LINE__);
        exit(EXIT_FAILURE);
}

// freeing the memory
free(incoming_message);
free(message);
```

Fig. 6. Scriere catre server + clean up

```
        //if exists file user.xml, check password.
        //if password is ok, login = 1;
        //if password is not ok, relogin.
        //if !exists user.xml, ask user if he wants to register
        //if user register, create xml.
        //if user not register, relogin.

        //generate the name of the file to be opened
        char userxml[256];
        memset(userxml, 0, 255);
        strcat(userxml, username);
        strcat(userxml, ".xml");

        int32_t register_bit = 0;
        //check if it exists
        if(access(userxml, F_OK) != 0) //file does not exist
        {
                //ask user if he wants to register
                if(register_bit == 1)
                {
                        //create user.xml
                        //add <login> field with 1
                }
                else
                {
                        continue;
                }
        }
        else //file exists => user exists
        {
                //open xml file
                //if password is ok, login = 1, replace <login> with 1
                //else continue
        }
}
```

Fig. 7. Partea de login(pseudocod)

7 Logarea unui user

Odata ce un user s-a conectat la server, va fi necesar sa intoduca username-ul si dupa parola. Serverul va cauta fisierul xml numit “hashed_username.xml”. Daca acest fisier nu exista, serverul trimite clientului optiunea de inregistrare.

In caz ca doreste sa se inregistreze, acest fisier va fi creat si bit-ul de login va fi setat. Nu pot exista doi useri care sa editeze acelasi fisier xml in acelasi timp.

8 Concluzii

Aplicatia este inca in procesul de realizare, intrucat trebuie sa implementez metodele de extragere a datelor din fisierul XML. To do list:

- AES encryption pentru parole(salting maybe?)
- Hashing pentru username(cu ajutorul djb2.c/h)
- Modificare + extragere informatii din .xml

References

Hasing functin djb2
`http://www.cse.yorku.ca/~oz/hash.html`
Libxml2 library for parsing and modifying an XML file
`http://xmlsoft.org/FAQ.html`