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/PasswordManager

– 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

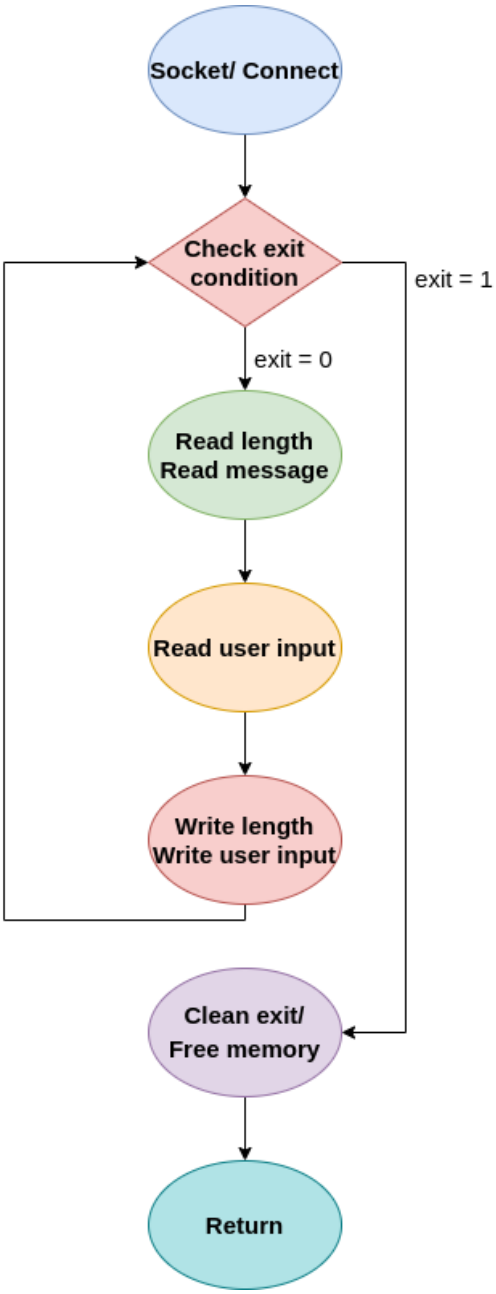

Fig. 1. Flowchart pentru client

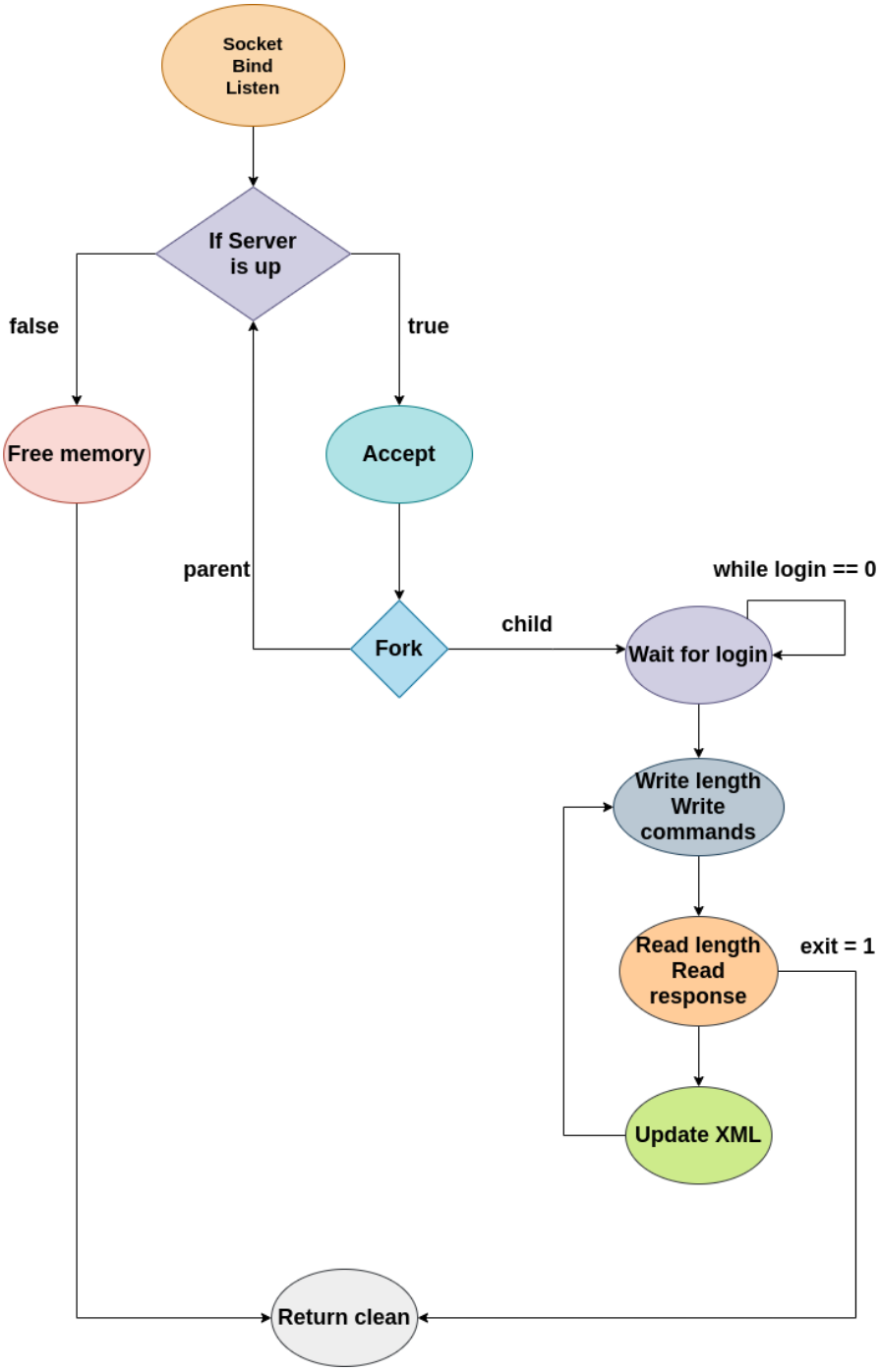

Fig. 2. Flowchart pentru server

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.

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:

```
family@andr3wport:~/PasswordManager/bin$ cat 6953319446904.xml
<?xml version="1.0"?>
<user>
  <masterpass>andrei</masterpass>
  <login>0</login>
  <category>
    <title>Redoot_mod</title>
    <username>username_for_redoot</username>
    <password>papaswoord_for_redoot</password>
    <url>www.redoot.com</url>
    <notes>Reddot is just reddit but dankyer</notes>
  </category>
</user>
family@andr3wport:~/PasswordManager/bin$
```

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, se va face un fork pentru a indeplini capabitatea de concurenta

Pentru a obtine hash-ul din numele clientului, vom folosi functia djb2 ce ne returneaza un long, pe care il vom transforma in char* astfel:

Verificarea existentei userilor se face prin verificarea existentei fisierului hashed_user.xml astfel:

```
// 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

```
// the main loop
while (!exit_server)
{
        int client;
        socklen_t length = sizeof(from);

        printf("[SERVER] Listening on port %d\n", port);
        fflush(stdout);

        // waiting in here untill a client comes
        client = accept(sd, (struct sockaddr *)&from, &length);

        if (client == -1)
        {
                printf("%s on line %d\n", strerror(errno), __LINE__);
                continue;
        }
        else // forking the execution, a child for every connection
        {
                int pid;
                if ((pid = fork()) == -1)
                {
                        printf("%s on line %d\n", strerror(errno), __LINE__);
                        close(client);
                        continue;
                }
                else
                {
                        if (pid != 0) // parent
                        {
                                close(client);
                                while (waitpid(-1, NULL, WNOHANG))
                                        ;
                        }
                        else // child
                        {
                                // closing the socket where we accepted the connection
                                // because we no longer need it
                                close(sd);

                                printf(GREEN "[SERVER] Connection established.\n" DEF);
                                fflush(stdout);
```

Fig. 7. Fork-ul

```
// generate the name of the file to be opened
memset(userxml, 0, 255);
uint64_t name = djb2((unsigned char*)username);
snprintf(userxml, length + 1, "%ld", name);
strcat(userxml, ".xml");
```

Fig. 8. Hashing the username

```
else // file exists => user exists
{
        fflush(stdout);
        printf(BLUE "[SERVER] User %s exists.Checking password..\n" DEF, username);
        fflush(stdout);

        // check to see if the user is already logged in
        if (!xmlCheckLogin(userxml))
        {
                if (xmlCheckPassword(userxml, password))
                {
                        // set logi n bit
                        login = 1;

                        // update login field
                        xmlReplaceLoginField(userxml, login);

                        // Logged in as user:pass
                        fflush(stdout);
                        printf(GREEN "[SERVER] Logged in as %s\n" DEF, username);
                        fflush(stdout);
                }
        }
}
```

Fig. 9. Verificarea existentei userilor

Alegerea operatiunii de catre user se va imparti in acest mod:

```
// switch based on choice
switch (choice)
{
case ADD_CATEGORY:
        xmlAddCategory(client, userxml); // done
        break;
case RM_CATEGORY:
        xmlRemoveCategory(client, userxml); // removes the category that has the title as the user entered
        break;
case MODIFY_CATEGORY:
        xmlModifyCategory(client, userxml);
        break;
case GET_CATEGORY:
        xmlGetCategory(client, userxml);
        break;
case EXIT:
        exit_client = 1;
}
```

Fig. 10. Optiuni

Parcurgerea unui fisier .xml se face astfel:

7 Adaugarea unui user nou

Daca in procesul de login, serverul intampina faptul ca nu exista vreun fisier numit astfel, va intreba userul daca doreste sa se logheze, moment in care sunt trimise doua optiuni: 1- da, 0- nu. Sa presupunem ca dorim sa ne logam. Astfel serverul apeleaza functia ”xmlCreateUser” ce ia ca parametri numele userului si parola userului pe care a introdus-o si creeaza astfel entry-uri specifice XML pentru a reusi parsarea.

Adaugarea nodurilor se face foarte usor:

8 Concluzii

Acest proiect a fost foarte interesant, mai mult am lucrat la acesta din placerea sa creez o aplicatie eficienta si folositoare, in limitele personale. As fi dorit sa introduc si o metoda de criptare a parolelor din fisier, dar ar fi luat destul de mult individual, mai ales ca ar fi trebuit sa folosesc un ”nonce” pentru fiecare criptare(number used once).

To do list:

– AES encryption pentru parole(salting maybe?) -

```
current = current->children;
while (current != NULL)
{
        if ((!xmlStrcmp(current->name, (const xmlChar *)"masterpass")))
        {
                xmlChar *key = xmlNodeListGetString(document, current->children, 1);
                if (!xmlStrcmp(key, (xmlChar *)password))
                {
                        xmlFree(key);
                        xmlFreeDoc(document);
                        return 1; // password is verified
                }
                else
                {
                        fflush(stdout);
                        // printf("key:%s", key);
                        fflush(stdout);
                        xmlFree(key);
                        xmlFreeDoc(document);
                        return 0; // password is verified, but it does not match
                }
        }
        current = current->next;
}
```

Fig. 11. Parcurgere XML

```c
void xmlCreateUser(const char *name, const char *password)
{
        xmlDocPtr new_document = NULL;
        xmlNodePtr root_node = NULL;

        // creating the document
        new_document = xmlNewDoc(BAD_CAST "1.0");
        root_node = xmlNewNode(NULL, BAD_CAST "user");
        xmlDocSetRootElement(new_document, root_node);

        // adding the child items, like password and login field
        xmlNewChild(root_node, NULL, BAD_CAST "masterpass", BAD_CAST password);
        // add <login> field with 1
        xmlNewChild(root_node, NULL, BAD_CAST "login", BAD_CAST "1");

        // saving the file to the memory
        xmlKeepBlanksDefault(0);
        xmlSaveFormatFile(name, new_document, 1);
        xmlFreeDoc(new_document);
        xmlCleanupParser();
}
```

Fig. 12. Adaugare noduri

– 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`