Documentatie proiect Froxy

Similachi Andrei Felix , grupa A3

Facultatea de Informatica, Str. General Berthelot Nr. 16, 700483, Iaşi, Romania
sivfhd@yahoo.com

Abstract. In urmatoarele sectiuni voi vorbi despre proiectul Froxy , generalitati si diferite tehnologii aplicate in realizarea acestuia, precum si viziunea mea finala asupra proiectului.

Keywords: FTP · Proxy · Concurenta.

1 Introducere

Proiectul Froxy are ca scop implementarea unui server proxy, capabil să gestioneze conexiuni concurente de la clienți și să ofere funcționalități precum autentificare, si descarcare de fisiere de la un server FTP. De asemenea proxy-ul trebuie sa poata interzice conectarea clientului la anumite site-uri pe baza unor politici din fisierul de configurare ,si sa filtreze fisierele descarcate , conform fisierului de configurare. In plus , in functie de credentialele utilizatorului , acesta poate fi chiar un administrator ce va avea in plus dreptul de a modifica fisierul de configurare.

2 Tehnologii Aplicate

2.1 Utilizarea TCP

Pentru implementarea concurenta a comunicatiei dintre proxy si client , am ales utilizarea protocolului TCP pentru a asigura o conexiune fiabilă și ordonată între client și server, esențială în transferul de date sensibile precum fișiere.

2.2 Utilizarea bazelor de date XML

Folosirea XML pentru configurarea proxy-ului prin libraria libxml , permite o gestionare flexibilă si organizata a setărilor , precum si a bazei de date , permițând administratorului să configureze timpul de pastrare a fisierelor dupa ultima logare a unui utilizator , tipurile de fișiere permise, dimensiunea maximă a fișierelor și politicile de acces . De asemenea fisierele obtinute de catre proxy prin utilizarea protocolului FTP vor fi stocate temporar in baza de date XML , de unde vor putea fi trimise catre client.

2.3 Concurenta

Pentru implementarea concurentei am ales utilizarea de thread-uri , datorita avantajelor aduse de acestea , cum ar fi : utilizarea intr-o masura mai mica a resurselor calculatorului , accesul facil la date si rapiditate .

3 Structura Aplicației

Aplicatia este împărțită în două module principale: partea de proxy - server.c și partea de client - client.c. Clientul trebui sa comunice prin intermediul proxy-ului cu server-ul FTP.

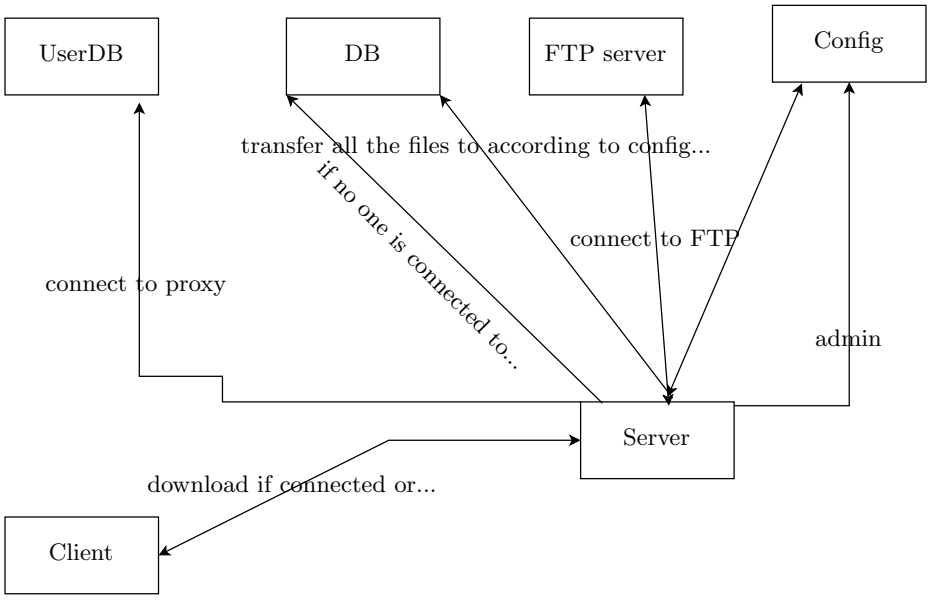

Fig. 1. Froxy Diagram

A more detailed diagram after proxy connection:

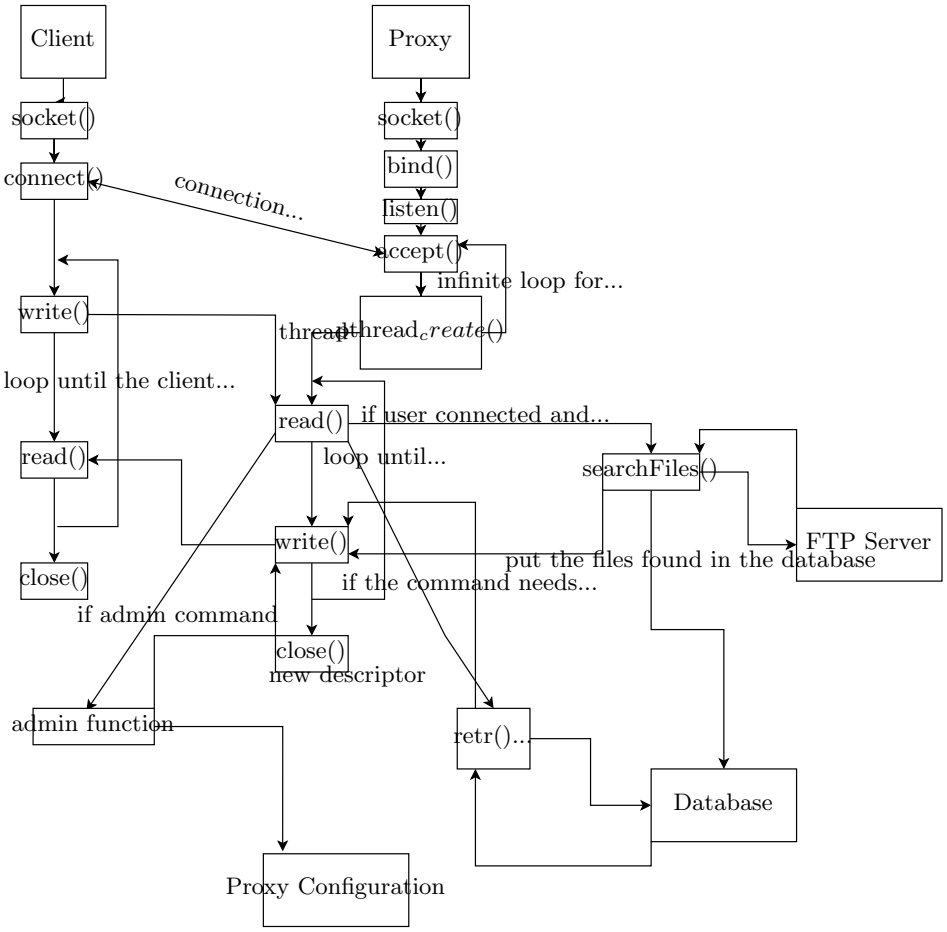

Fig. 2. Diagram

4 Aspecte de Implementare

Structuri folosite pentru stocarea datelor:

```
struct ProxyConfiguration
{
    int maxFileSize;
    char toKeep[50];
    struct
    {
        char fileTypes[MAX_FILE_TYPE_COUNT][10];
        int fileTypeCount;
    } allowedFileTypes;
```

```
    struct
    {
        char url[MAX_URL_LENGTH];
        int forbiddenSitesCount;
        struct
        {
            char dayRange[MAX_DAY_RANGE_LENGTH];
            char timeRange[MAX_TIME_RANGE_LENGTH];
            char clientDomain[MAX_CLIENT_DOMAIN_LENGTH];
        } accessPolicy;
    } forbiddenSites[MAX_FILE_TYPE_COUNT];
} p;
//in aceasta structura se pastraza configuratia proxy-ului , care este preluata prin parsar
```

Implementarea concurentei folosind thread-uri

```
static void *treat(void *);

int handleClient(void *);

void *continuousExecution(void *arg);

int main()
{
    /*pthread_t tid;
    if (pthread_create(&tid, NULL, continuousExecution, NULL) != 0)
    {
        fprintf(stderr, "Error creating thread\n");
        return 1;
    }*/
    pthread_t th[100]; // Identificatorii thread-urilor care se vor crea
    int i = 0;
    initialiseConfig(&p);
    struct sockaddr_in server; // structura folosita de server
    struct sockaddr_in from;
    bzero(&from, sizeof(from));
    int sd = initServCltCommunication(&server);
    initialise_database();
    /* servim in mod concurent clientii... */
    while (1)
    {
        int client;
        int length = sizeof(from);
        thData *td; // parametru functia executata de thread
```

```
    printf("[server]Asteptam la portul %d...\n", PORT);
    fflush(stdout);

    /* acceptam un client (stare blocanta pina la realizarea conexiunii) */
    client = accept(sd, (struct sockaddr *)&from, &length);

    /* eroare la acceptarea conexiunii de la un client */
    if (client < 0)
    {
        perror("[server]Eroare la accept().\n");
        return errno;
    }
    td = (struct thData *)malloc(sizeof(struct thData));
    td->idThread = i;
    td->cl = client;
    td->connected = 0;
    td->username = (char *)malloc(50 * sizeof(char));
    td->proxyUser = (char *)malloc(50 * sizeof(char));
    bzero(td->username, 50);
    td->domain = (char *)malloc(50 * sizeof(char));
    bzero(td->domain, 50);

    char h[100];
    bzero(h, 100);
    socklen_t addr_len = sizeof(struct sockaddr_in);
    if (getnameinfo((struct sockaddr *)&from, addr_len, h, sizeof(h), NULL, 0, 0) == 0)
    {
        strcpy(td->domain, h);
        printf("Client domain: %s\n", td->domain);
    }
    else
    {
        perror("getnameinfo\n");
    }
    pthread_create(&th[i], NULL, &treat, td);
    i++;
} /* while */
if (remove(database) == 0)
{
    printf("XML database deleted successfully.\n");
}
else
{
    perror("Error deleting XML database");
    return EXIT_FAILURE;
```

```
    }
    /*if (pthread_join(tid, NULL) != 0)
    {
        fprintf(stderr, "Error joining thread\n");
        return 1;
    }*/
}
```

Codul din thread unde se primesc mesaje de la client. Clientul se va conecta la un cont de proxy , care poate avea drepturi de administrator sau nu , dupa , va trebui introdusa o adresa ftp sau un server ftp local in vederea logarii la acesta . In cazul in care logarea este efectuata cu succes se preiau fisierele existente pe server-ul ftp in functie de fisierul de configurare si se introduc in baza de date , de unde vor fi trimise catre client daca sunt cerute.

```
static void *treat(void *arg)
{
    struct thData tdL;
    tdL = *((struct thData *)arg);
    printf("[thread]- %d - Asteptam mesajul...\n", tdL.idThread);
    fflush(stdout);
    pthread_detach(pthread_self());
    handleClient((struct thData *)arg);
    close((intptr_t)arg);
    return (NULL);
};

void *continuousExecution(void *arg)
{
    while (1)
    {
        for (int i = 0; i < nrOfOnlineUsers; i++)
        {
            if (onlineUsers[i].n == 0)
            {
                time_t currentTime = time(NULL);
                onlineUsers[i].lastConnectionTime = currentTime;

                int tokeepSeconds = atoi(p.toKeep);

                while (onlineUsers[i].n == 0)
                {
                    sleep(1);
                    currentTime = time(NULL);
```

```
                    int elapsedSeconds = currentTime - onlineUsers[i].lastConnectionTime;

                    if (elapsedSeconds >= tokeepSeconds && onlineUsers[i].n == 0)
                    {
                        printf("Time has passed. Performing necessary actions.\n");
                        removeFromDb(onlineUsers[i].username);
                        break;
                    }
                }
            }
        }
    }

    pthread_exit(NULL);
}

int handleClient(void *arg)
{
    struct thData tdL;
    tdL.connected = 0;
    tdL = *((struct thData *)arg);
    int client = tdL.cl;
    char msg[100];
    char msgrasp[100];
    char prUsr[50], passd[50];
    bzero(prUsr, sizeof(prUsr));
    bzero(passd, sizeof(passd));
    int ftp, step = 0;
    int admin;
    int proxyCon = 0;
    while (1)
    {
        bzero(msg, sizeof(msg));
        int nrOf;
        printf("[server]Asteptam mesajul...\n");
        fflush(stdout);

        if (read(client, msg, sizeof(msg)) < 0)
        {
            perror("[server]Eroare la read() de la client.\n");
            return errno;
        }
        printf("[server]Mesajul a fost receptionat...%s\n", msg);

        if (strncmp("modifyMaxFileSize", msg, strlen("modifyMaxFileSize")) == 0 && admin ==
```

```
{
    msg[strlen(msg) - 1] = '\0';
    char maxfilesize[10];
    bzero(maxfilesize, sizeof(maxfilesize));
    sscanf(msg, "modifyMaxFileSize %s", maxfilesize);
    printf("Mfsize: %s\n", maxfilesize);
    modifyMaxFileSize(maxfilesize, &p);
    write(client, "Max file size modified", strlen("Max file size modified"));
}
else if (strncmp("modifyToKeep", msg, strlen("modifyToKeep")) == 0 && admin == 1 &&
{
    msg[strlen(msg) - 1] = '\0';
    char mToKeep[10];
    bzero(mToKeep, sizeof(mToKeep));
    sscanf(msg, "modifyToKeep %s", mToKeep);
    printf("MtoKeep: %s\n", mToKeep);
    modifyToKeep(mToKeep, &p);
    write(client, "Max file size modified", strlen("Max file size modified"));
}
else if (strncmp("addFileType", msg, strlen("addFileType")) == 0 && admin == 1 && p
{
    msg[strlen(msg) - 1] = '\0';
    char fileType[10];
    bzero(fileType, sizeof(fileType));
    sscanf(msg, "addFileType %s", fileType);
    printf("FileType: %s\n", fileType);
    addFileType(fileType, &p, client);
}
else if (strncmp("deleteFileType", msg, strlen("deleteFileType")) == 0 && admin ==
{
    msg[strlen(msg) - 1] = '\0';
    char fileType[10];
    bzero(fileType, sizeof(fileType));
    sscanf(msg, "deleteFileType %s", fileType);
    printf("FileType: %s\n", fileType);
    deleteFileType(fileType, &p, client);
}
else if (strncmp("addForbiddenUrl", msg, strlen("addForbiddenUrl")) == 0 && admin =
{
    msg[strlen(msg) - 1] = '\0';
    char url[50];
    bzero(url, sizeof(url));
    char drange[50];
    bzero(drange, sizeof(drange));
    char trange[10];
```

```
    bzero(trange, sizeof(trange));
    char cdom[10];
    bzero(cdom, sizeof(cdom));
    sscanf(msg, "addForbiddenUrl %s %s %s %s", url, drange, trange, cdom);
    addForbiddenUrl(url, drange, trange, cdom, &p, client);
}
else if (strncmp("deleteForbiddenUrl", msg, strlen("deleteForbiddenUrl")) == 0 && ac
{
    msg[strlen(msg) - 1] = '\0';
    char url[50];
    bzero(url, sizeof(url));
    sscanf(msg, "deleteForbiddenUrl %s", url);
    deleteForbiddenUrl(url, &p, client);
}
else if ((step == 0 || tdL.connected == 0) && strcmp(msg, "quit\n") != 0 && proxyCon
{
    ftp = initConnProxyFTP(client, msg, &tdL.connected, tdL.domain);
    step++;
}
else if (strcmp(msg, "quit\n") == 0)
{
    for (int i = 0; i < nrOfProxyUsers; i++)
    {
        if (strcmp(tdL.proxyUser, ProxyUser[i].username) == 0)
        {
            printf("%s\n", tdL.proxyUser);
            for (int j = i; j < nrOfProxyUsers; j++)
            {
                strcpy(ProxyUser[j].username, ProxyUser[j + 1].username);
            }
        }
    }
    free((void *)tdL.proxyUser);
    for (int i = 0; i < nrOfOnlineUsers; i++)
    {
        if (strcmp(tdL.username, onlineUsers[i].username) == 0)
        {
            printf("%d\n", onlineUsers[i].n);
            onlineUsers[i].n--;
            printf("%d\n", onlineUsers[i].n);
            if (onlineUsers[i].n == 0)
            {
                time_t currentTime = time(NULL);
                onlineUsers[i].lastConnectionTime = currentTime;
```

```
                int tokeepSeconds = atoi(p.toKeep);

                while (onlineUsers[i].n == 0)
                {
                    sleep(1);
                    currentTime = time(NULL);
                    int elapsedSeconds = currentTime - onlineUsers[i].lastConnectio

                    if (elapsedSeconds >= tokeepSeconds && onlineUsers[i].n == 0)
                    {
                        removeFromDb(tdL.username);
                        break;
                    }
                }
            }
        }
        free((void *)tdL.username);
        break;
    }
    else if (tdL.connected == 1 && proxyCon == 1)
    {
        if (strncmp(msg, "USER", 4) == 0)
        {
            user(ftp, msg + 5, client);
            strcpy(tdL.username, msg + 5);
            tdL.username[strlen(tdL.username) - 1] = '\0';
            printf("%s\n", tdL.username);
        }
        else if (strncmp(msg, "PASS", 4) == 0)
        {
            pass(ftp, msg + 5, client, tdL.username);
            if (strcmp(tdL.username, "\0") != 0)
            {
                printf("User connected: %s\n", tdL.username);
                if (verify_if_user_exists(tdL.username) == 0)
                {
                    add_user(tdL.username);
                    searchFiles(ftp, tdL.username, ".txt", "\0");
                    onlineUsers[nrOfOnlineUsers].n = 1;
                    strcpy(onlineUsers[nrOfOnlineUsers++].username, tdL.username);
                }
                else
                {
                    for (int i = 0; i < nrOfOnlineUsers; i++)
```

```
                {
                    if (strcmp(tdL.username, onlineUsers[i].username) == 0)
                    {
                        onlineUsers[i].n++;
                        break;
                    }
                }
            }
        }
        else
        {
            printf("No user connected\n");
        }
    }
    else if (strncmp(msg, "RETR", 4) == 0)
    {
        printf("%s\n", tdL.username);
        msg[strlen(msg) - 1] = '\0';
        printf("%s\n", msg + 5);
        retr(client, msg + 5, tdL.username);
    }
    else if (strcmp(msg, "LIST\n") == 0)
    {
        printf("%s\n", tdL.username);
        msg[strlen(msg) - 1] = '\0';
        printf("%s\n", msg + 5);
        list(client, tdL.username);
    }
    else
    {
        printf("Unknown command: %s", msg);
        write(client, "Unknown command", strlen("Unknown command"));
    }
}
else if (proxyCon == 0 && (nrOf = sscanf(msg, "user %49s pass %49s\n", prUsr, passd
{
    proxyCon = connToProxy(prUsr, passd, tdL.cl, &admin, tdL.proxyUser);
    printf("%d\n", proxyCon);
}
else if (proxyCon == 0 && tdL.connected == 0)
{
    printf("You are not connected");
    write(client, "You are not connected", strlen("You are not connected"));
}
```

```
        bzero(msg, sizeof(msg));
    }

    close(client);
    close(ftp);
}
```

Aici se initializeaza conexiunea pentru comenzi intre proxy si FTP ,server-ul FTP putand fi local , sau putand fi specificat printr-o adresa url.

```
int initConnProxyFTP(int client, char *host, int *con, char *userDomain)
{
    int ftp;
    struct sockaddr_in ftpAddr, dataAddr;
    char buff[MAX_BUFFER_SIZE];

    if ((ftp = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket creation error");
        return errno;
    }

    bzero(&ftpAddr, sizeof(ftpAddr));
    if (strcmp(host, "127.0.0.1") != 0 && strcmp(host, "0") != 0)
    {
        if (verifyHostByProxy(host, userDomain) == 1)
        {
            struct hostent *serverHost;
            host[strlen(host) - 1] = '\0';
            printf("%s\n", host);
            serverHost = gethostbyname(host);
            if (serverHost == NULL)
            {
                perror("Error resolving host name");
                write(client, "Error resolving host name!", strlen("Error resolving host na
                return errno;
            }
            else
                *con = 1;
            ftpAddr.sin_family = AF_INET;
            ftpAddr.sin_port = htons(FTP_PORT);
            memcpy(&ftpAddr.sin_addr, serverHost->h_addr_list[0], serverHost->h_length);
        }
        else
        {
```

```
            write(client, "Acces denied due to proxy rules\n", strlen("Acces denied due to p
            return errno;
        }
    }
    else
    {
        ftpAddr.sin_family = AF_INET;
        ftpAddr.sin_port = htons(FTP_PORT);
        ftpAddr.sin_addr.s_addr = inet_addr(FTP_SERVER);
    }
    if (connect(ftp, (struct sockaddr *)&ftpAddr, sizeof(ftpAddr)) < 0)
    {
        perror("Connection failed");
        *con = 0;
        write(client, "Connection failed", strlen("Connection failed"));
        return errno;
    }

    bzero(buff, MAX_BUFFER_SIZE);

    read(ftp, buff, sizeof(buff));

    printf("Server: %s", buff);
    write(client, buff, strlen(buff));
    return ftp;
}
```

Urmatoarele functii au rolul de a parsa directoarele de pe server-ul ftp si de a prelua fisierele in functie de configuratia proxy-ului pentru a le stoca temporar in baza de date urmand sa fie sterse in momentul in care toti userii conectati la acelasi cont se vor deconecta.

```
int initialiseConnection(int ftp)
{
    int data_sock;
    char buffer[MAX_BUFFER_SIZE];
    struct sockaddr_in dataAddr;
    bzero(&dataAddr, sizeof(struct sockaddr_in));
    sendCommand(ftp, "PASV");

    bzero(buffer, MAX_BUFFER_SIZE);
    read(ftp, buffer, sizeof(buffer));
    printf("Server: %s", buffer);

    unsigned int ip[4];
```

```
    int port1, port2;
    sscanf(buffer, "227 Entering Passive Mode (%u,%u,%u,%u,%u,%u)",
           &ip[0], &ip[1], &ip[2], &ip[3], &port1, &port2);

    int port = port1 * 256 + port2;

    if ((data_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Data socket creation error");
        return errno;
    }

    memset(&dataAddr, '0', sizeof(dataAddr));
    dataAddr.sin_family = AF_INET;
    dataAddr.sin_port = htons(port);
    dataAddr.sin_addr.s_addr = htonl((ip[0] << 24) | (ip[1] << 16) | (ip[2] << 8) | ip[3]);

    if (connect(data_sock, (struct sockaddr *)&dataAddr, sizeof(dataAddr)) < 0)
    {
        perror("Data connection failed");
        return errno;
    }
    return data_sock;
}

int verifyHostByProxy(char *host, char *domain)
{
    for (int i = 0; i < p.forbiddenSites->forbiddenSitesCount; i++)
    {
        if (strstr(host, p.forbiddenSites[i].url) != 0)
        {
            time_t currentTime = time(NULL);
            struct tm *localTime = localtime(&currentTime);
            char dayOfWeekString[100];
            strftime(dayOfWeekString, sizeof(dayOfWeekString), "%A", localTime);
            char timeInHoursString[3];
            strftime(timeInHoursString, sizeof(timeInHoursString), "%H", localTime);
            int startHour, endHour, currHour;
            sscanf(p.forbiddenSites[i].accessPolicy.timeRange, "%d-%d", &startHour, &endHou
            sscanf(timeInHoursString, "%d", &currHour);
            printf("Day of week:%s\n", dayOfWeekString);
            if (strstr(p.forbiddenSites[i].accessPolicy.dayRange, dayOfWeekString) != 0 &&
                currHour >= startHour && currHour <= endHour)
            {
                if (strstr(domain, p.forbiddenSites[i].accessPolicy.clientDomain) != 0)
```

```
                {
                    return 0;
                }
            }
        }
    }
    return 1;
}


void add_file_to_user(const char *filename, char *content, const char *user)
{
    xmlInitParser();
    xmlDocPtr doc;
    xmlNodePtr rootNode;
    doc = xmlReadFile(database, NULL, 0);
    if (doc == NULL)
    {
        fprintf(stderr, "Failed to parse the XML file.\n");
        return;
    }
    rootNode = xmlDocGetRootElement(doc);
    for (xmlNodePtr node = rootNode->children; node; node = node->next)
    {
        if (node->type == XML_ELEMENT_NODE && strcmp(user, node->name) == 0)
        {
            xmlNodePtr newNode = xmlNewNode(NULL, BAD_CAST filename);
            xmlAddChild(node, newNode);
            xmlNodePtr newContent = xmlNewText(BAD_CAST content);
            xmlAddChild(newNode, newContent);
        }
    }
    xmlSaveFormatFile(database, doc, 1);
    xmlFreeDoc(doc);
    xmlCleanupParser();
}

void putInDb(int ftp, const char *file, const char *path, const char *user)
{
    int data_sock = initialiseConnection(ftp);
    char buffer[MAX_BUFFER_SIZE];
    bzero(buffer, MAX_BUFFER_SIZE);
    strcpy(buffer, "RETR ");
    strcat(buffer, path);
    sendCommand(ftp, buffer);
```

```
    bzero(buffer, MAX_BUFFER_SIZE);
    read(ftp, buffer, sizeof(buffer));
    printf("Server: %s", buffer);

    if (buffer[0] == '1' && buffer[1] == '5' && buffer[2] == '0')
    {
        bzero(buffer, MAX_BUFFER_SIZE);
        ssize_t bytesRead;
        while ((bytesRead = read(data_sock, buffer, sizeof(buffer))) > 0)
        {
            add_user(user);
            add_file_to_user(file, buffer, user);
            fwrite(buffer, 1, bytesRead, stdout);
            bzero(buffer, MAX_BUFFER_SIZE);
        }
    }
    else
    {
        printf("Error: Failed to retrieve file\n");
    }
    bzero(buffer, MAX_BUFFER_SIZE);
    read(ftp, buffer, sizeof(buffer));
    printf("Server: %s", buffer);
    close(data_sock);
}

void searchFiles(int ftp, const char *username, const char *property, const char *dir)
{
    int data_sock = initialiseConnection(ftp);
    char buffer[MAX_BUFFER_SIZE];
    char command[50];
    bzero(command, 50);
    strcpy(command, "LIST ");
    strcat(command, dir);
    char subdires[50][50];
    char path[50][50];
    char files[50][50];
    bzero(subdires, 50 * 50);
    bzero(path, 50 * 50);
    bzero(files, 50 * 50);
    int contor = 0, f_contor = 0;
    sendCommand(ftp, command);
    bzero(buffer, MAX_BUFFER_SIZE);
```

```
read(ftp, buffer, sizeof(buffer));
printf("Server: %s", buffer);

if (buffer[0] == '1' && buffer[1] == '5' && buffer[2] == '0')
{
    bzero(buffer, MAX_BUFFER_SIZE);
    ssize_t bytesRead;
    while ((bytesRead = read(data_sock, buffer, sizeof(buffer))) > 0)
    {
        char *token = strtok(buffer, "\n");
        while (token != NULL)
        {
            printf("File or directory: %s\n", token);

            if (strchr(token, '.') == 0)
            {
                printf("Adding subdirectory: %s\n", token);
                char directory[50];
                bzero(directory, 50);
                int ct = 0;
                for (int i = strlen(token) - 1; strncmp(token + i, " ", 1); i--)
                    ct++;
                char dire[100];
                bzero(dire, 100);
                if (dire[strlen(dire) - 1] != '/' && strcmp(dir, "\0") != 0)
                    snprintf(dire, strlen(dir) + 1 + strlen(token + strlen(token) - ct)
                else
                {
                    snprintf(dire, strlen(token + strlen(token) - ct), "%s", token + st
                }
                printf("Subdir : %s\n", token + strlen(token) - ct);

                printf("Path : %s\n", dire);

                strcpy(subdires[contor++], dire);
            }
            else
            {
                printf("Found a file with property '%s': %s\n", property, token);
                int ct = 0;
                for (int i = strlen(token) - 1; strncmp(token + i, " ", 1); i--)
                    ct++;
                int ok1 = 0, ok2 = 0;
                for (int i = 0; i < p.allowedFileTypes.fileTypeCount; i++)
                {
```

```
                        if (strstr(token + strlen(token) - ct, p.allowedFileTypes.fileTypes
                            ok1 = 1;
                    }
                    int size;
                    sscanf(token, "%*s %*s %*s %*s %d", &size);
                    printf("Size of the file: %d bytes\n", size);

                    if (ok1 == 1 && size <= p.maxFileSize)
                    {
                        strcpy(files[f_contor], token + strlen(token) - ct);
                        if (dir[strlen(dir) - 1] != '/' && strcmp(dir, "\0") != 0)
                            snprintf(path[f_contor++], strlen(dir) + 1 + strlen(token + str
                        else
                        {
                            snprintf(path[f_contor++], strlen(token + strlen(token) - ct),
                        }
                    }
                }

                token = strtok(NULL, "\n");
            }
            bzero(buffer, MAX_BUFFER_SIZE);
        }
    }
    else
    {
        printf("Error: Failed to retrieve file\n");
    }
    bzero(buffer, MAX_BUFFER_SIZE);
    read(ftp, buffer, sizeof(buffer));
    printf("Server: %s", buffer);
    close(data_sock);
    for (int i = 0; i < f_contor; i++)
    {
        putInDb(ftp, files[i], path[i], username);
    }
    for (int i = 0; i < contor; i++)
    {
        searchFiles(ftp, username, property, subdires[i]);
    }
}
```

Acesta este functia retr care preia un fisier specificat din baza de date si il trimite catre client.

```
void retr(int client, const char *filename, const char *username)
{
    xmlInitParser();
    xmlDocPtr doc;
    xmlNodePtr rootNode;
    doc = xmlReadFile(database, NULL, 0);
    if (doc == NULL)
    {
        fprintf(stderr, "Failed to parse the XML file.\n");
        return;
    }
    int found = 0;
    rootNode = xmlDocGetRootElement(doc);
    for (xmlNodePtr node = rootNode->children; node; node = node->next)
    {
        if (node->type == XML_ELEMENT_NODE && strcmp(username, node->name) == 0)
        {
            for (xmlNodePtr file = node->children; file; file = file->next)
            {
                if (file->type == XML_ELEMENT_NODE)
                {
                    const char *currentFilename = (char *)file->name;
                    const char *currentContent = (char *)xmlNodeGetContent(file->children);

                    if (strcmp(filename, currentFilename) == 0)
                    {
                        printf("%s\n%s\n", currentFilename, currentContent);
                        found = 1;
                        write(client, currentContent, strlen(currentContent));
                    }
                }
            }
        }
    }
    if (found == 0)
        write(client, "There is no such file", strlen("There is no such file"));
    xmlSaveFormatFile(database, doc, 1);
    xmlFreeDoc(doc);
    xmlCleanupParser();
}
```

5 Concluzii

Proiectul oferă o soluție funcțională pentru gestionarea transferului de fișiere într-un mediu securizat și configurabil. Pentru a îmbunătăți soluția propusă, pot fi luate în considerare : aspecte precum o gestionare mai detaliată a erorilor și extinderea funcționalităților pentru a acoperi mai multe comenzi FTP si eventual verificarea input-ului introdus de administrator in client.

References

1. Computer Networks UAIC Course Homepage, https://profs.info.uaic.ro/ computernetworks/index.php.
2. RFC 959, https://www.ietf.org/rfc/rfc959.txt.
3. LNCS Homepage, http://www.springer.com/lncs.
4. Andrei Scutelnicu Homepage, https://www.andreis.ro/.
5. Site-ul utilizat in realizarea diagramei(Fig. 1), https://app.diagrams.net/.