Proiect CastDoc

Popoiu Claudiu-Daniel — A4

Dec 2022

1 Introducere

Proiectul CastDoc ofera utilizatilor posibilitatea de a converti diferite fisiere (pdf, txt, tex, etc.). Utilizatorii vor trimite serverului prin intermediul clientului *(care va oferi si o interfata grafica prin care utilizatorii isi pot selecta fisierul si tipul de convertire dorit)* un fisier, care va fi convertit de catre server si va fi trimis inapoi la client pentru a fi salvat la o locatie aleasa de utilizator. Serverul va oferi si functionalitatea de caching pentru o performanta mai buna.

2 Tehnologii utilizate

Protocolul **TCP** este un protocol din stiva de protocoale **TCP/IP** aflat in nivelul transport ce asigura livrarea ordonata a unui flux de octeti de la client la server si stabileste conexiunea dintre client si server printr-un **3-way handshake**.

Vom folosi protocolul **TCP**, datorita faptului ca ne dorim ca mesajele sa ajunga in intregime de la client la server si inapoi. Clientul si serverul vor schimba fisiere intre ele, deci nu ne permitem sa pierdem informatii pe parcurs pentru ca astfel va fi afectata integritatea fisierului.

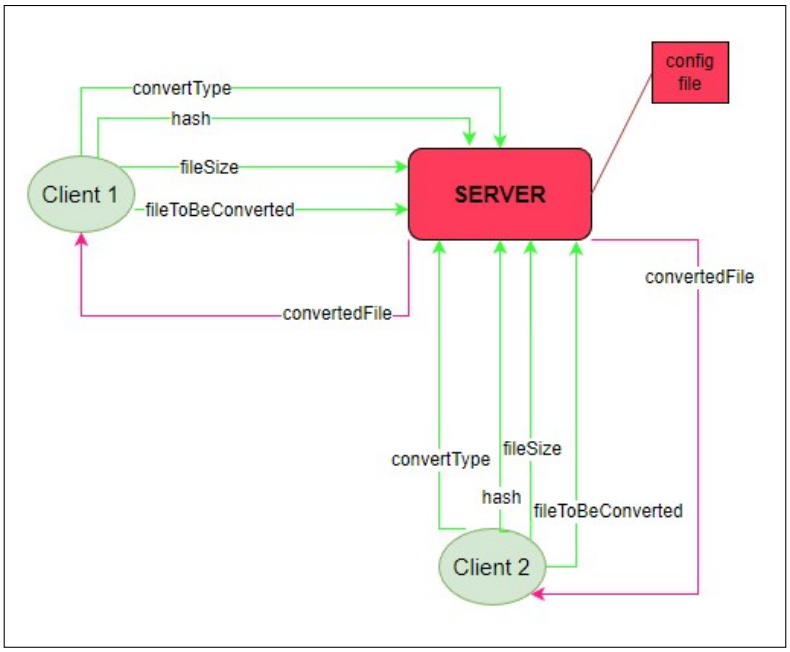
TCP este un protocol *orientat-conexiune*, deci este mai potrivit pentru acest proiect decat protocolul **UDP** deoarece ne permite sa realizam comunicatii full-duplex fara pierdere de informatie. Desi UDP este mai rapid decat protocolul TCP, acesta nu ne garanteaza ca tot continutul fisierul trimis va ajunge la destinatie. De asemenea, nu ne dorim sa primim informatia din fisiere intr-o ordine gresita, iar TCP asigura faptul ca ordinea acestora este corecta.

Pentru implementarea unui server concurent care sa raspunda tuturor cererilor clientilor ne vom folosi de **threaduri**. Atunci cand un client se conecteaza la server, un thread este creat pentru a-l servi pe clientul respectiv.

3 Arhitectura aplicatiei

Clientii vor trimite serverului tipul fisierelor care vor fi convertite si tipul in care se doreste sa fie convertite (ex: ps2pdf, asp2php, etc.). Odata ajunse la server, acesta va cauta tipurile in fisierul de configurare unde va prelua utilitarul necesar pentru convertire. Pentru a implementa functionalitatea de caching, clientul va trimite si hashul fisierului ce urmeaza sa fie convertit. Pe computerul pe care este rulat procesul server vor exista foldere pentru fiecare tip acceptat (pdf, asp, ps, txt, etc.), iar in aceste foldere vor fi salvate fisierele convertite cu numele hashului respectiv. Astfel, atunci cand clientul va trimite hash-ul, serverul va verifica daca acesta a fost deja convertit si il va trimite inapoi (daca este gasit in folderelul respectiv tipului final). Daca acest fisier nu este gasit de catre server, clientul va trimite la server dimensiunea fisierului (pentru a verifica daca acesta a fost livrat in totalitate in server) si fisierul ce urmeaza sa fie convertit. Avand in vedere ca ne folosim de protocolul TCP, transportul de mesaje intre procese se va face prin fluxuri de octeti. Asadar, vom deschide fisierele ce urmeaza sa fie convertite si vom transmite secvente de octeti pana cand ajungem la EOF. Procesul server va primi aceste secvente si le va scrie intr-un fisier nou, urmand sa converteasca acest fisier cu utilitarul necesar gasit in config file si va trimite fisierul convertit inapoi la client.

.



4 Detalii de implementare

UI-ul va fi implementat cu ajutorul frameworkului Qt. Prin aceasta interfata userul se va putea conecta la server si va putea alege ce fisier doreste sa converteasca, respectiv tipul convertirii.
In *server.c* vom pregati socketul si structurile de date necesare si vom astepta ca un client sa se conecteze. Cand acesta incearca sa se conecteze, va fi creat un nou thread care va servi clientul respectiv.

```
int main()
{
    struct sockaddr_in server; // structura folosita de server
    struct sockaddr_in from;
    int nr; // mesajul primit de trimis la client
    int sd; // descriptorul de socket
    int pid;
    pthread_t th[200]; // Identificatorii thread-urilor care se vor crea
    int i = 0;

    /* crearea unui socket */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("[server]Eroare la socket().\n");
        return errno;
    }
    /* utilizarea optiunii SO_REUSEADDR */
    int on = 1;
    setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

    /* pregatirea structurilor de date */
    bzero(&server, sizeof(server));
    bzero(&from, sizeof(from));

    /* umplem structura folosita de server */
    /* stabilirea familiei de socket-uri */
    server.sin_family = AF_INET;
    /* acceptam orice adresa */
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    /* utilizam un port utilizator */
    server.sin_port = htons(PORT);

    /* atasam socketul */
    if (bind(sd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
    {
        perror("[server]Eroare la bind().\n");
        return errno;
    }

    /* punem serverul sa asculte daca vin clienti sa se conecteze */
    if (listen(sd, 2) == -1)
    {
        perror("[server]Eroare la listen().\n");
        return errno;
```

```
    }
    /* servim in mod concurent clientii...folosind thread-uri */
    while (1)
    {
        int client;
        threadData *td; // parametru functia executata de thread
        int length = sizeof(from);

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

        // client= malloc(sizeof(int));
        /* acceptam un client (stare blocanta pana la realizarea conexiunii) */
        if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0)
        {
            perror("[server]Eroare la accept().\n");
            continue;
        }

        /* s-a realizat conexiunea, se astepta mesajul */

        // int idThread; //id-ul threadului
        // int cl; //descriptorul intors de accept

        td = (struct threadData *)malloc(sizeof(struct threadData));
        td->idThread = i++;
        td->cl = client;

        pthread_create(&th[i], NULL, &treat, td);

    }
}
```

Dupa ce clientul se conecteaza la server, acesta va putea selecta in interfata grafica tipul de convertire dorit si fisierul ce urmeaza sa fie convertit. Clientul va trimite tipul fisierului ce va fi convertit si tipul final si hashul.

```
void sendType(char *fileTypes, int sd)
{
    char response[TRANSFERSIZE];
    //printf("%s \n %d\n", fileTypes, sizeof(fileTypes));
    if (send(sd, fileTypes, TRANSFERSIZE, 0) == -1)
    {
        perror("[client] Error in sending file TYPE");
        exit(1);
    }
    bzero(response,TRANSFERSIZE);
}
```

```
char* getHash(FILE* fp); // WILL ADD LATER
void sendHash(char* hashCode); // WILL ADD LATER
```

Procesul server va verifica daca fisierul a mai fost convertit inainte. Daca acesta gaseste fisierul convertit il va trimite, altfel va trimite un mesaj clientului care va trimite dimensiunea fisierului si fisierul.

```
bool doesFileExist(char* hash); //WILL ADD LATER
void sendResponseToClient(); //WILL ADD LATER
// WILL ADD LATER
```

```
//client.cpp
void sendFileSize(long int size, int sd)
{
    char szString[15] = {0};
    sprintf(szString, "%ld", size);
    if (send(sd, szString, TRANSFERSIZE, 0) == -1)
    {
        perror("[client] Error in sending file SIZE");
        exit(1);
    }
}
```

```
void sendFile(FILE *fp, int sd,long int size)
{
    int checker;
    char info[TRANSFERSIZE];
    long int readBytes = 0;

    while (fread(info, TRANSFERSIZE, 1, fp)!=NULL)
    {       readBytes+=strlen(info);
            info[strlen(info)]='\0';
        if ((checker = send(sd, info, TRANSFERSIZE, 0)) == -1)
        {
            perror("[client] Error in sending file");
            exit(1);
        }

        bzero(info, TRANSFERSIZE);

    }
           if ((checker = send(sd, info, strlen(info), 0)) == -1)
        {
            perror("[client] Error in sending file");
            exit(1);
        }
    bzero(info, TRANSFERSIZE);
}
```

Serverul va primi fisierul, il va converti cu utilitarul necesar din configFile si il va trimite inapoi la client.

```
void writeRecvInfo(int sd)
{
    FILE *fp;
    char fileName[50];
    sprintf(fileName, "receivedFile%d.pdf", fileNr);
    fileNr++;
    int r;
    char info[TRANSFERSIZE] = {0};
    if ((fp = fopen(fileName, "wb")) == NULL)
    {
        printf("ERROR OPENING FILE - SERVER");
        exit(1);
    }
```

```
    r = recv(sd, info, TRANSFERSIZE, 0);
    char *remaining;
    long fileSize;
    info[r] = '\0';
    long int readBytes = 0;
    fileSize = strtol(info, &remaining, 10);
    while (readBytes < fileSize)
    {

        r = recv(sd, info, TRANSFERSIZE, 0);

        readBytes += r;
        if (r <= 0)
        {

           // printf("Received %d b\n", r);
            fflush(fp);
            fclose(fp);
            bzero(info, TRANSFERSIZE);
            return;
        }
     // printf("Received %d b\n", r);
     // printf("%s\n", info);
        fwrite(info, 1, r, fp);

        fflush(fp);
        bzero(info, TRANSFERSIZE);
        printf("%ld | %ld", readBytes, fileSize);
    }
}
```

```
void convertFile(); // WILL ADD LATER
```

Dupa ce serverul converteste fisierul si il trimite inapoi la client, il va salva in folderul tipului sau cu numele hashului, pentru a putea fi trimis imediat la client daca acesta il va solicita din nou.
Userul va putea seta prin intermediul interfetei si folderul in care doreste sa fie salvat fisierul primit de la server.

5 Concluzii

Imbunatatiri:
- O interfata grafica mai atractiva
- Adaugarea unui progress bar in interfata care sa ii arate utilizatorului progresul
- Optiunea de a trimite serverului un folder cu mai multe fisiere de acelasi tip pentru a fi convertite.

6 Bibliografie

https://ro.wikipedia.org/wiki/Transmission_Control_Protocol
https://profs.info.uaic.ro/~computernetworks/index.php

https://profs.info.uaic.ro/~ioana.bogdan/
https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/