Procesarea Imaginilor

Membrii echipei:
Cincu Andrada Maria Alexandra
Chira Andreea-Marina
Kovacs Roland
Ciobotariu Gheorghe

Cuprins

Capitolul 1

Documentatia Aplicatiei

1.1 Descrierea temei

Aplicatia "Procesarea Imaginilor" este o aplicatie de tip client-server multithread care are rolul de a procesa imaginile in functie de optiunea aleasa de client, avand ca rezultat trimiterea optiunii alese clientului. Serverul va pune la dispozitia clientilor un meniu din care pot alege urmatoarele optiuni: transformarea unei imagini in grayscale, inversarea culorilor unei imagine, punerea unei imagini pe mai multe nivele(Pyramid level), rotirea imaginilor, conturarea obiectelor dintr-o imagine(Edge Detection).

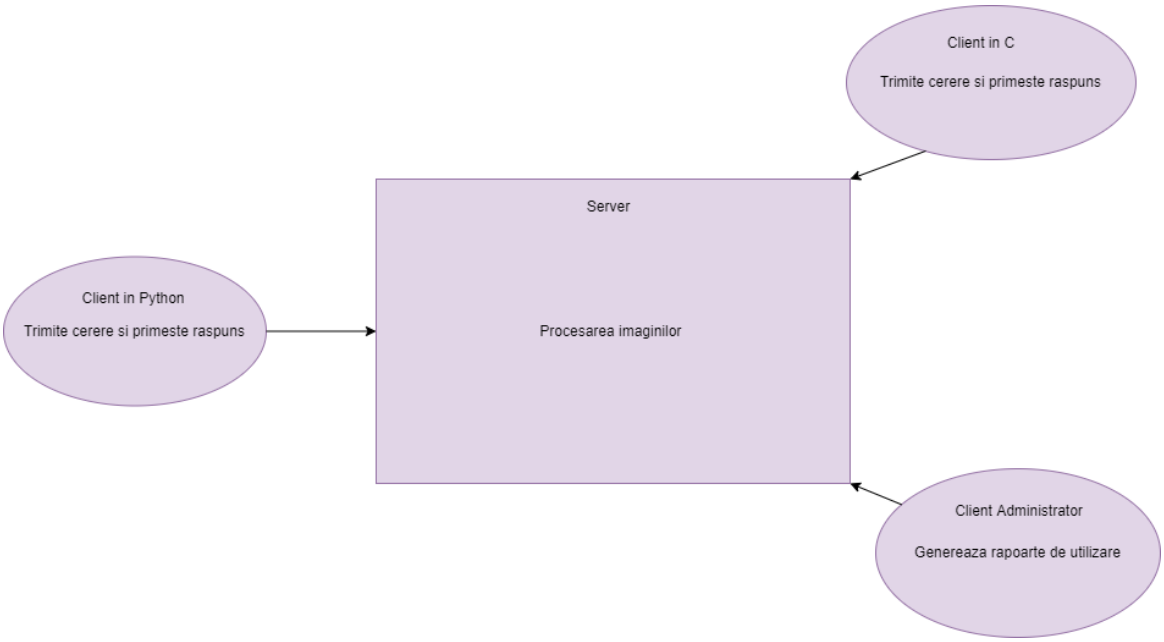

Figura 1.1: Diagrama Arhitecturii

1.2 Structura aplicatiei

Structura aplicatiei este formata dintr-un server, care va permite accesul mai multor clienti. Pentru realizarea server-ului se va folosi limbajul de programare C, utilizand diferite biblioteci, precum pthreads pentru a implementa mai multe fire de executie,cu scopul de a suporta conectarea a mai multor clienti.
In principiu, serverul va accepta 3 tipuri de clienti:

- Uploader Client - responsabil pentru incărcarea imaginilor pe server, urmand ca acestea sa fie transformate de server. De asemenea acesta va putea sa descarce si imaginile transformate de pe server.
- Client - solicita ce transformari vor fi efectuate asupra imaginilor incarcate.
- Clientul Administrator - vede cea mai recenta activitate a serverului, intrerupe serverul sau iese din sesiunea curenta.

Transformarile acceptate de server sunt: conversia în grayscale, inversarea culorilor,gaussian blur, esantionarea piramidei in sus/ in jos, rotirea imaginilor cu unghiuri arbitrare, detectarea marginilor(utilizand filtrul Canny).

In principiu, serverul presupune folosirea unui api socket din sys/socket.h pentru a se lega de masina pe care ruleaza. Acesta va astepta o nouă conexiune si va crea fire de executie pentru noile conexiuni intalnite. De asemenea, serverul va avea capacitatea de a identifica tipul de client la care se conecteaza si ofera serviciul mensionat mai sus in mod corespunzator.

1.3 Tehnologii folosite

Aplicatia "Imagini de Procesare", ruleaza pe sistemul de operare Ubuntu(Linux) si este realizata folosind limbajele de programare C si Python.
Aplicatia utilizeaza un protocol de comunicare care permite clientilor sa se conecteze la server folosind o adresa ip si un port.
Odata cu conectarea clientilor la server, clientul controller, scris in limbajul de programare C, va alege optiunea dorita, optiune pe care serverul o va aplica la transformarea imaginii. Imaginea va fi incarcata de catre clientul uploader, scris in limbajul de programare Python si va fi transmisa catre server unde vor avea loc transformarile acesteia. Dupa realizarea acestei operatii, serverul va returna imaginea procesata.

1.4 Server

Serverul multithreaded efectueaza procesarea imaginilor, Acesta are capacitatea de a servi trei tipuri de clienti:

- Client de incarcare.
- Client administrator.
- Client de control al modului de transformare a imaginii. (Controler client)

Executabilul (aplicația server) care rezultă, din compilarea codului sursa, are o singura linie de comanda care este numarul portului, in care serverul ar trebui sa asculte pentru incomming.
Pentru a efectua procesarea imaginilor, serverul utilizeaza interpretorul Python pentru a rula un script care va efectua sarcina de procesare a imaginii necesara pe imaginea pe care serverul a primit-o de la client. Imaginea transformată este trimisă înapoi la client.
Aceasta functie se ocupa de uploader client, aceasta functie primeste imaginea de la acesta si transforma imaginea in functie setarea curenta a modului de transformare a serverului. In cele din urma, functia returneaza imaginea transformata catre clientul solicitant.

```
 void handle_uploader_client(const THREAD_DATA *th_data)
{
        int cl_sock_fd = th_data->client_sock;
        int ret = 0;
        char *okay = "OK";
        char *success = "SUCCESS";
        char *error = "ERROR";

        write(cl_sock_fd, okay, strlen(okay) + 1);

        char file_name[1024];
        char file_size[64];
        char msg[32];
        memset(msg, 0x00, 32);
        char path[128];
        memset(path, 0x00, 128);

    char transformed_img_file_path[128];
```

```
memset(transformed_img_file_path, 0x00, 128);
memset(file_size, 0x00, 64);
memset(file_name, 0x00, 1024);

ret = read(cl_sock_fd, file_name, 1024);

if (ret != -1)
{       pthread_mutex_lock(&mux);
        printf("\n[*]Uploader Client: Filename: %s\n", file_name);
        pthread_mutex_unlock(&mux);
}
else
{
        return;
}
write(cl_sock_fd, okay, strlen(okay) + 1);

read(cl_sock_fd, file_size, 64);

write(cl_sock_fd, okay, strlen(okay) + 1);

unsigned char *buff = (unsigned char *)malloc(4096);

ssize_t n = 0; // Number of bytes read.

sprintf(path, "./Uploads/%s", file_name);

FILE *fp = fopen(path, "wb");

ssize_t sz = 0;
ssize_t fSize = atol(file_size);

while (sz < fSize)
{
        n = read(cl_sock_fd, buff, 4096);
        fwrite(buff, 1, n, fp);
        sz += n;
}

fclose(fp);
pthread_mutex_lock(&mux);
printf("\n[*]Successfully received file: %s \n", file_name);
memset(serv_info.log_data, 0x00, 1024);
sprintf(serv_info.log_data, "Connection (%s, %d)
uploaded '%s'; %d bytes. ",
inet_ntoa(th_data->client_addr.sin_addr), ntohs(th_data->client_addr.sin_port
file_name, fSize);

pthread_mutex_unlock(&mux);

sprintf(transformed_img_file_path, "./Transformed/%s", file_name);

pthread_mutex_lock(&mux);
int code = serv_info.transformation_mode;
float rotangle = serv_info.rotation;
pthread_mutex_unlock(&mux);
```

```
ret = transform_image(path, transformed_img_file_path, code, rotangle);

if (ret == 0)
{
        write(cl_sock_fd, success, strlen(success) + 1);
        printf("\n[*] '%s' successfully transformed! \n", file_name);
}
else
{
        printf("\n[!]ERROR occured in the transformation process :
        ( \n");
        write(cl_sock_fd, error, strlen(error) + 1);
        return;
}
read(cl_sock_fd, msg, 32);

if (strcmp(msg, "NEWSIZE") != 0)
{
        pthread_mutex_lock(&mux);
        printf("[!] Client violated communication protocols. \n
        Stopped communicating with them. \n");

        pthread_mutex_unlock(&mux);
        return;
}
struct stat file_status;
stat(transformed_img_file_path, &file_status);

memset(file_size, 0x00, 64);
sprintf(file_size, "%lld", file_status.st_size);

write(cl_sock_fd, file_size, strlen(file_size) + 1);
memset(msg, 0x00, 32);
read(cl_sock_fd, msg, 32);

if (strcmp(msg, "TRANSFER") != 0)
{
        pthread_mutex_lock(&mux);
        printf("[!] Client not ready for file transfer. \n
        Stopped communicating with them. \n");
        pthread_mutex_unlock(&mux);
        return;
}
fp = fopen(transformed_img_file_path, "rb");

while ((n = fread(buff, 1, 4096, fp)) > 0)
{
        write(cl_sock_fd, buff, n);
}

pthread_mutex_lock(&mux);
printf("\n[*] Successfully sent transformed image,
of size %lld bytes. \n", file_status.st_size);
memset(serv_info.log_data, 0x00, 1024);
sprintf(serv_info.log_data, "Transformed image of size %lld bytes,
sent to connection (%s, %d). ",
file_status.st_size, inet_ntoa
(th_data->client_addr.sin_addr),
```

```
        ntohs(th_data->client_addr.sin_port));
        pthread_mutex_unlock(&mux);
        fclose(fp);
        free(buff);
}
```

1.5 Uploader Client

Clientul este realizat in Python si repreoduce un script care incaraca o imagine pentru a fi procesata pe serverul ostru de procesare a imaginilor. Acesta asteapta ca imaginea sa fie procesata de server si de asemena descarca imaginea transformata.
Dupa ce primim ca raspuns de la server ca, conexiunea a avut loc cu succes, clientul va incarca imaginea pe server.

```
while True:
        filepath = input("[*]Please provide path
        to image file to upload, to our server: ")
        if os.path.exists(filepath):
                        break
        else:
                print("\n[!]Please provide correct path! \n")

                filename = filepath.split("/")[-1]
                sock.send(filename.encode("ascii") + b"\x00")
                reply = sock.recv(1024)
                if reply == b"OK\x00":

                sz = os.stat(filepath).st_size

                sock.send(str(sz).encode("ascii")+b"\x00")
                reply = sock.recv(1024)

                if reply == b"OK\x00":
                with open(filepath, "rb") as fObj:
                        while True:
                        data = fObj.read(4096)
        if len(data) == 0:
                print("\n[*]Successfully done uploading the file!\n")
                                break
```

1.6 Controller Client

Acest client controleaza modul de transforamare a imaginii de catre serverul de procesare a imaginilor. Dupa ce primim ca raspuns de la server ca, conexiunea a avut loc cu succes, clientului i se va afisa un meniu de unde va avea posibilitatea de a alege una dintre optiunile prezentate pentru transformarea imaginii.

```
printf("** CONTROL OPTIONS **\n");

        printf("\t1. Grayscale \n");
        printf("\t2. Color Inversion\n");
        printf("\t3. Arbitrary rotation\n");
        printf("\t4. Pyramid Upsampling\n");

        printf("\t5. Pyramid Downsampling\n");
        printf("\t6. Edge detection. \n");
```

```
printf("\t7. Gaussian Blur. \n");

printf("Option >>> ");

int op = 0; // User specified option.
float rotangle = 0.0;

scanf("%d", &op);

if (op == 3)
{
        printf("\nPlease input a rotation angle
        in degrees (counterclockwise): ");
        scanf("%f", &rotangle);

        sprintf(buff, "%d:%2.2f", op, rotangle);
}
else
{
        sprintf(buff, "%d", op);
}
write(sock_fd, buff, 16);
```

1.7 Administrator Client

Acest client vede cea mai recenta activitate a serverului, intrerupe serverul sau iese din sesiunea curenta.
Dupa ce primim ca raspuns de la server ca, conexiunea a avut loc cu succes, una dintre functionalitatile pe care administratorul le poate realiza este orpirea serverului.

```
else if (op == 2)
        {
        sprintf(buff, "STOP");

        if (write(sock_fd, buff, 32) < 0)
        {
                printf("[!]ERROR in write() \n");
                break;
        }


        if (read(sock_fd, msg, 32) < 0)
        {
                printf("[!]ERROR in write() \n");
                break;
        }

        f (strcmp(msg, "BYE") == 0)
        {
                printf("\n[*]Server SHUTDOWN initiated, successfully...\n");
        }
        else
        {
        printf("\n[!]An ERROR occured in initiating SHUTDOWN !\n");
        }
                break;
                }
```

1.8 Concluzie

Printre principalele caracteristici ale codului se regaseste folosirea socket-uri pentru a face posibibila comunicarea intre server si clienti, iar de asemenea pentru crearea protocoalelor de transmisie folosim INET.

1.9 Rularea aplicatiei

Server: pentru conectarea la server se parcurg urmatorii pasi:

- introducem calea catre fisier : cd Server
- compilam fisierul server: gcc -o server server.c -lpthread
- rulam fisierul si introducem portul: .\server 8080

Controller Client: pentru conectarea clientului la server se parcurg urmatorii pasi:

- introducem calea catre fisier : cd Clients
- compilam fisierul server: gcc -o controlle controller.c
- rulam fisierul: .\controller
- introducem portul si adresa ip pentru conectarea la server: 8080, 127.0.0.1

Uploader Client: pentru conectarea clientului la server se parcurg urmatorii pasi:

- introducem calea catre fisier : cd Clients
- compilam fisierul server: python3 uploader.py
- introducem portul si adresa ip pentru conectarea la server: 8080, 127.0.0.1

Administrator Client: pentru conectarea clientului la server se parcurg urmatorii pasi:

- introducem calea catre fisier : cd Clients
- compilam fisierul server: gcc -o administrator_client administrator_client.c
- rulam fisierul: .\administrator_client
- introducem portul si adresa ip pentru conectarea la server: 8080, 127.0.0.1

In urma conectarii clientilor la server, Controller Client va putea alege una din optiunile afisate in meniu, dupa care Uploader Client va incarca imaginea catre server. Serverul va aplica filtrul ales de catre Controller Client, iar Uploader Client ofera optiunea de a descarca imaginea. In urma unei operatii complete Administrator Client va putea vedea jurnalul recent de activitate, opri serverul si iesirea din sesiune.