Image Processing Server(C): Server pentru procesarea imaginilor.

- Documentație proiect final Programare Concurentă și Distribuită-

Mercea Alex-Ovidiu, Lobonțiu Bogdan

Universitatea de Vest din Timișoara

1 Introducere:

Proiectul denumit “Image Processing Server” presupune implementarea unei aplicații compusă dintr-o componentă server și o componentă client care are rolul de a primi imaginile și de a le procesa în funcție de opțiunea aleasă de utilizator, urmând ca apoi să o trimită înapoi clientului în versiunea cerută. În cadrul aplicației utilizatorul se conectează cu un un username și o parolă, create de către un administrator. După conectare, utilizatorul are mai multe opțiuni de procesare a imaginilor. Acesta introduce numele imaginii urmate de extensie (ex. jpeg).

2 Tehnologii utilizate

Sisteme de operare compatibile și limbaje de programare utilizate. Aplicația va rula pe sistemele de operare Linux (de preferat, **Ubuntu**) și va fi realizată utilizând cod scris in **C/C++** și **Python** (librăria OpenCV).

Comunicarea în rețea. Relativ la comunicarea in rețea, aplicația va utiliza, la nivelul acesteia, un protocol de comunicare ce va permite verificarea utilizatorilor folosindu-se de userul și parola acestora. Apoi utilizatorii vor alege imaginea pe care vor să o modifice, această imagine urmând să fie transmisă mai departe către server prin protocolul FTP (File Transfer Protocol) .Acest protocol se va ajuta de o conexiune bazată pe TCP. Serverul trimite imaginea mai departe spre clientul Python care are rolul de a procesa imaginile cu ajutorul librăriei OpenCV și apoi trimite imaginea modificată înapoi serverului. În final serverul trimite imaginea modificată clientului.

3 Arhitectura

Aplicația ” Image Processing Server” va fi alcatuită din mai multe componente ce vor interacționa spre a-și îndeplini scopul primar.

3.1 Client-ul

Client-ul este modulul disponibil utilizatorilor finali, pe care îl vor putea utiliza pe stațiile de lucru. El reprezintă un program conceput pentru Linux prin care clienții se pot conecta la server folosindu-se de datele de conectare create de către administrator. Prin intermediul acestui client utilizatorii interacționează cu serverul (trimit imagini și aleg tipul de procesare de imagine dorit).

3.2 Server-ul

Server-ul este componenta care asigură verificările de rigoare pentru utilizatori(în caz că datele lor de conectare sunt greșite conexiunea cu aceștia este sistată).
Serverul face conexiunea cu clientul administrator și cu clientul normal, dar și cu clientul Python și face astfel posibil transferul de fișiere.

3.3 Sistemul de login:

Comunicarea între client și server se va realiza printr-un protocol de login care permite verificarea utilizatorilor și atribuirea spațiului de lucru corespunzător acestora.
• **Distribuirea resurselor specifice fiecărui client.** Serverul se asigură că datele de login specifice clientului au fost validate și folosindu-se de acestea distribuie acces clientului la ele.

• **Persistența fișierelor.** Server-ul se asigură că datele clienților au fost salvate în caz că acesta a fost închis de administrator sau a avut o eroare și a picat.

3.4 Clientul Admin

Clientul Admin este folosit de către administrator. Acesta are opțiunea de a crea conturi pentru utilizatori (username și parolă), urmând ca serverul să le stocheze în baza de date. Administratorul are opțiunea de a vedea toate imaginile trimise către server și de a șterge imagini. Administratorul se poate folosi inclusiv de procesarea imaginilor.

3.5 Clientul Python

Clientul Python primește imaginile de la server și le modifică în funcție de cerința utilizatorului folosind funcții din librăria OpenCV. Tipurile de procesare de imagine care sunt implementate în clientul Python sunt următoarele: White Grayscale, Salt&Pepper, HSV, Image Sharpening, Gaussian Blur și Canny.

În urma aplicării oricărui algortim menționat mai sus, clientul Python trimite imaginea procesată spre Server cu denumirea AfterProcessing urmată de un index care va fi incrementat automat, evitând astfel suprascrierea, deci în consecință este evitată și pierderea imaginilor procesate anterior.

3.6 Cod relevant

- Functia de primire de imagini în server.

```
int receive_image(int socket)
{
    // Start function
    int buffersize = 0, recv_size = 0,size = 0, read_size, write_size,
packet_index =1,stat;
    char imagearray[10241],verify = '1', server_folder[512],
incremental[16];
    FILE *image;

    //Find the size of the image
    do{
        stat = read(socket, &size, sizeof(int));
    }while(stat<0);

    printf("Packet received.\n");
    printf("Packet size: %i\n",stat);
    printf("Image size: %i\n",size);
    printf("\n");
    char buffer[] = "Got it";

    //Send our verification signal
    do{
        stat = write(socket, &buffer, sizeof(int));
    }while(stat<0);
    printf("Reply sent\n");
    printf(" \n");
    bzero(incremental, sizeof(incremental));
    sprintf(incremental, "%d", k+1);
    strcpy(server_folder, "/home/alex/ProiectPCD/server_folder/capture2");
    strcat(server_folder, incremental);
    strcat(server_folder, ".jpeg");
    k++;
    printf("Directory is Given =%s\n", server_folder);
    image = fopen(server_folder, "w");

    if( image == NULL) {
        printf("Error has occurred. Image file could not be opened\n");
        return -1;
    }
```

```
    //Loop while we have not received the entire file yet
    int need_exit = 0;
    struct timeval timeout = {10,0};

    fd_set fds;
    int buffer_fd, buffer_out;
    while(recv_size < size) {
        //while(packet_index < 2){
        FD_ZERO(&fds);
        FD_SET(socket,&fds);

        buffer_fd = select(FD_SETSIZE,&fds,NULL,NULL,&timeout);
        if (buffer_fd < 0)
            printf("error: bad file descriptor set.\n");
        if (buffer_fd == 0)
            printf("error: buffer read timeout expired.\n");
        if (buffer_fd > 0)
        {
            do{
                read_size = read(socket,imagearray, 10241);
            }while(read_size <0);

            printf("Packet number received: %i\n",packet_index);
            printf("Packet size: %i\n",read_size);

            //Write the currently read data into our image file
            write_size = fwrite(imagearray,1,read_size, image);
            printf("Written image size: %i\n",write_size);

            if(read_size !=write_size) {
            printf("error in read write\n");    }

            //Increment the total number of bytes read
            recv_size += read_size;
            packet_index++;
            printf("Total received image size: %i\n",recv_size);
            printf(" \n");
            printf(" \n");
        }
    }
    fclose(image);
    printf("Image successfully Received!\n\n\n");
    return 1;
}
```

- Functia de trimitere de imagini din server.

```
      void send_image(int socket){

    FILE *picture;
    int size, read_size, stat, packet_index;
    char send_buffer[10240], read_buffer[256], server_folder[512],
incremental[16];
    packet_index = 1;

    bzero(incremental, sizeof(incremental));
    sprintf(incremental, "%d", k);
```

```
    strcpy(server_folder,
"/home/alex/ProiectPCD/server_folder/new_grayScale");
    strcat(server_folder, incremental);
    strcat(server_folder, ".jpeg");
    printf("Directory is Given =%s\n", server_folder);
    picture = fopen(server_folder, "r");
    printf("Getting Picture Size\n");

    if(picture == NULL) {
        printf("Error Opening Image File");
    }
    fseek(picture, 0, SEEK_END);
    size = ftell(picture);
    fseek(picture, 0, SEEK_SET);
    printf("Total Picture size: %i\n",size);

    //Send Picture Size
    printf("Sending Picture Size\n");
    write(socket, (void *)&size, sizeof(int));

    //Send Picture as Byte Array
    printf("Sending Picture as Byte Array\n");

    do {
        //Read while we get errors that are due to signals.
        stat=read(socket, &read_buffer , 255);
        printf("Bytes read: %i\n",stat);
    }while (stat < 0);

    printf("Received data in socket\n");
    printf("Socket data: %c\n", *read_buffer);
    while(!feof(picture)) {
        //Read from the file into our send buffer
        read_size = fread(send_buffer, 1, sizeof(send_buffer)-1, picture);

        //Send data through our socket
        do{
            stat = write(socket, send_buffer, read_size);
        }while (stat < 0);
        printf("Packet Number: %i\n",packet_index);
        printf("Packet Size Sent: %i\n",read_size);
        printf(" \n\n");

        packet_index++;
        //Zero out our send buffer
        bzero(send_buffer, sizeof(send_buffer));
    }
    fclose(picture);
}
```

4 Concluzii:

” Image Processing Server” se adresează clienților care au nevoie de o aplicație pentru procesarea imaginilor, ușor de utilizat, ce oferă multipli algortimi de procesare de imagini.

4.1 Îmbunătățiri posibile:

Ca un adaos la funcționalitatea de bază, prezentată și deja existentă a aplicației, aceasta se poate îmbunătăți prin următoarele:

- **Implementarea unei interfețe grafice folosind Qt / GTK+**. Pentru acces mai ușor la date și realizarea operațiunilor enumerate mai sus într-un mod mai user-friendly se poate folosi o interfață.

5 Bibliografie:

- https://man7.org/linux/man-pages/man2/socket.2.html
- http://stackoverflow.com/questions/11580944/client-to-server-authentication-in-c-using-sockets
- https://www.tutorialspoint.com/unix_sockets/index.htm
- https://www.ibm.com/docs/en/ztpf/2020?topic=considerations-unix-domain-sockets
- https://medium.com/swlh/getting-started-with-unix-domain-sockets-4472c0db4eb1
- http://vichargrave.com/network-programming-design-patterns-in-c/
- https://docs.opencv.org/4.5.2/d2/d96/tutorial_py_table_of_contents_imgproc.html