Lab03. Mini-server de imagens HTTP

Faça um mini-servidor HTTP utilizando *threads* e *sockets*. Ou seja:

- seu servidor receberá requisições de diversos clientes usando *sockets*
- seu servidor usará o protocolo HTTP para recuperar uma imagem (usando o *socket* criado anteriormente)
- seu servidor inspecionará a mensagem HTTP para avaliar o método de acesso (e.g. GET) e qual a URL informada
- seu servidor procurará pelo arquivo no "lado do servidor" a partir da URL informada
- cada *thread* deverá usar, novamente, o protocolo HTTP sobre um novo *socket* para recuperar e responder a requisição ao cliente.

Entregue o código-fonte de seu programa mini-servidor.

Atividade em dupla

Alessandro Bezerra Da Silva – 41908767

Pedro Unello Neto – 41929713

Obs: As imagens utilizadas foram retiradas do software Spotify, a escolha de tais foi feita de maneira aleatória.

Obs: Foi compilado usando o gcc do Ubuntu/PopOS.

Código Fonte (Também incluso na entrega em CodigosLocal.rar, junto com as imagens utilizadas):

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>     //Tudo acima e padrao para ling C

#include <string.h>     //Manipulacao de strings

#include <sys/types.h>  //Mesma da de baixo
#include <sys/stat.h>   //Para uso do stat(), para identificar o tamanho de um arquivo (imagem)

#include <fcntl.h>      //Para uso do open() e macros do mesmo (O_RDONLY, por exemplo)

#include <sys/socket.h> //Junto com as duas abaixo, e usada na criacao de sockets
#include <netinet/in.h>
#include <arpa/inet.h>

#include <pthread.h>    //Uso das posix threads

int PartOf(char *all, char *part) //Identifica se um string "all" contem uma substring "part",
                                  //Devolvendo -1 caso nao tenha, e o indice do final da
substring, caso tenha
{
    int i = 0, j = 0;
    while (all[i] != '\0')
    {
        while (all[i] == part[j])
        {
            i++;
            j++;
            if (part[j] == '\0')
```

```
            {
                return i;
            }
        }
        j = 0;
        i++;
    }
    return -1;
}

void *ClientFunction(void *receiv) //Funcao da posix thread
{

    int* dados = (int*) receiv;     //Recebe os argumentos na criacao da thread
    int idClient = *dados;

    char* request = malloc(sizeof(char) * 2048); //Aloca espaco para requisicao que sera
recebida do client
    int bytes = recv(idClient, request, 2048, 0); //Recebe tal requisicao

    printf("\nRequisicao de %d: \n%s", idClient, request);

    if (PartOf(request, "GET")){ //Se a requisicao for do tipo GET

        int link = PartOf(request, "/"); //Procura a url desejada, colocando em "link", o indice
de tal

        if (*(request + link) == ' ') {  //Se a url for vazia (nao especificado um arquivo ou
pagina)
            printf("Mandando pagina indice para %d\n", idClient); //Manda um cabeçalho de
resposta com uma pagina indice, em texto
            send(idClient, //A pagina indice e hardcoded, com todas as imagens disponiveis
localmente no servidor
                    "HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-Length: 88\r\n\r\n"
                    "Servidor de imagens:"
                    "\n/twitter.png"
                    "\n/instagram.png"
                    "\n/img4.jpg"
                    "\n/img3.jpg"
                    "\n/img2.jpg"
                    "\n/img1.jpg"
                    , 167, 0);
        }

        else { //Se a url nao for vazia, ou seja, foi pedido uma pagina ou arquivo
            char* path = malloc(sizeof(char) * 2048); //Aloca um buffer para o caminho/url
pedida
            strcpy(path, request + link); //Copia a requisicao para path
```

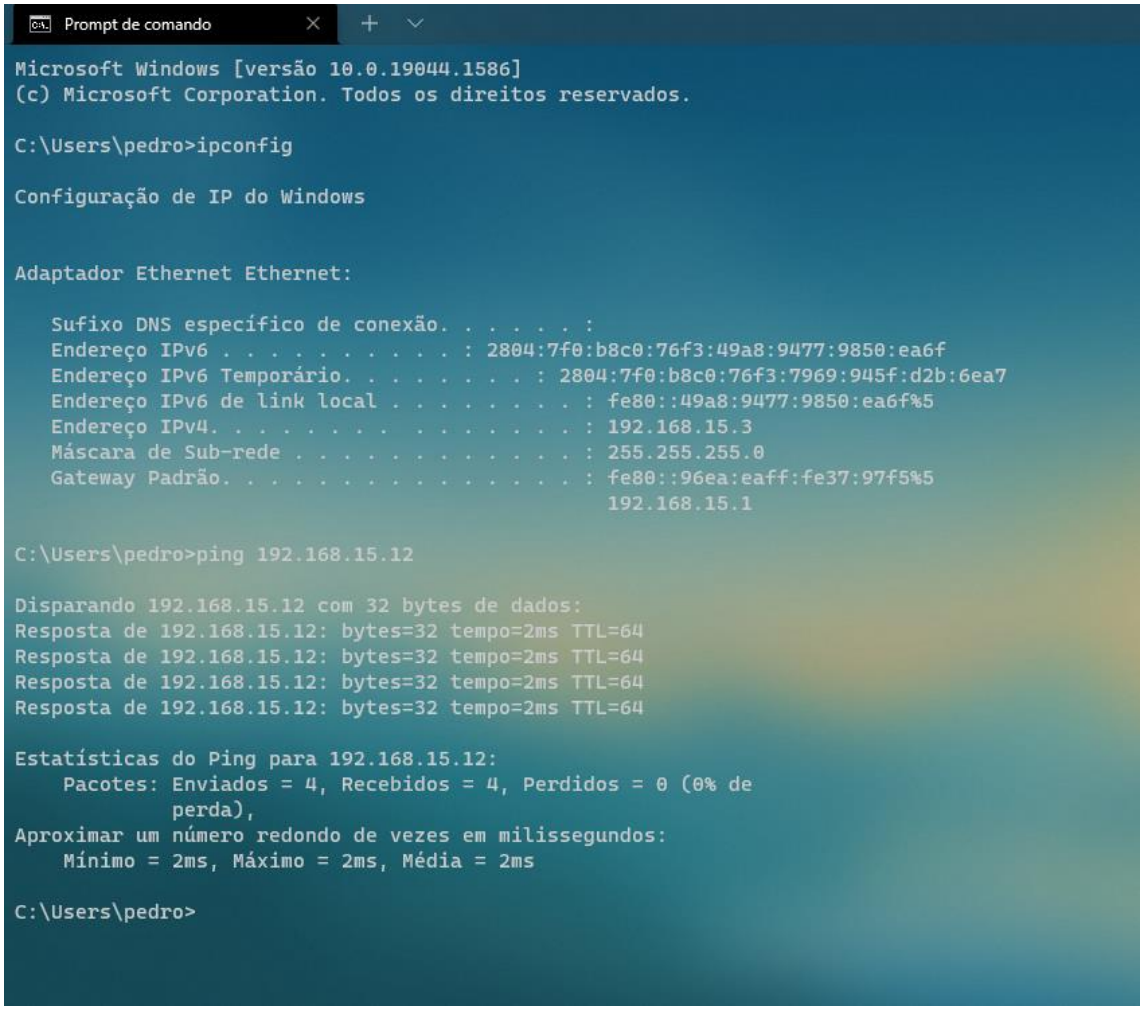
```
            *(path + PartOf(path, " ") - 1) = '\0'; //Coloca o char finalizador de string ao
final da url
            printf("Mandando %s para %d\n", path, idClient);
            FILE* image = fopen(path, "rb"); //Tenta abrir o arquivo/imagem
            struct stat filesize;
            int fileStatus = fstat(open(path, O_RDONLY), &filesize); //Tenta ler o tamanho do
arquivo/imagem
            free(path);

            if (image == NULL || fileStatus < 0) //Se nao conseguir ler a imagem ou seu tamanho,
manda resposta como BAD REQUEST
            {
                printf("Falha na requisicao de %d, imagem não existente \n", idClient);
                send(idClient,
                    "HTTP/1.1 404 Not Found\r\n\r\n",
                    32, 0);
                free(request);
                return NULL;
            }

            char* buffer = malloc(sizeof(char) * 4000); //Aloca um buffer para o cabecalho da
resposta
            strcpy(buffer, "HTTP/1.1 200 OK\r\nContent-Length: "); //Copia o comeco do
cabecalho, com codigo 200 OK
            char* size = malloc(sizeof(char) * 1000); //Aloca um buffer "size" para o tamanho do
arquivo, agora em char, para o Content-length
            sprintf(size, "%zd", filesize.st_size); //Preenche o buffer "size", como o tamanho,
em bytes do arquivo
            strcat(buffer, size); //Adiciona (append) o tamanho do arquivo apos o Content-length
            strcat(buffer, "\r\nConnection: keep-alive\r\n\r\n"); //Adiciona o fim do cabecalho
            send(idClient, buffer, strlen(buffer), 0); //Manda o cabecalho de resposta para o
client

            char* imageSend = malloc(sizeof(char) * filesize.st_size); //Aloca um buffer com
tamanho = contagem de bytes da imagem/arquivo
            fread(imageSend, sizeof(char), filesize.st_size + 1, image); //Usa o fread para
preenchelo com a imagem/arquivo inteiro
            send(idClient, imageSend, filesize.st_size, 0); //Manda para o client

            //Desaloca memoria
            free(buffer);
            free(size);
            free(imageSend);
        }
    }

    free(request);
    return NULL; //Fim thread.
}
```

```c
int main(int argc, char* argv[]) //O main recebe a quantidade de clients que serao atendidos simutaneamente
{
    struct sockaddr_in conn;
    int curClient = 0;
    int maxClient = strtol(argv[1], NULL, 10); //Cria estrutura de conexao para socket, e recebe atributos de qtd clients

    pthread_t* handles = malloc(sizeof(pthread_t) * maxClient); //Aloca (qtd clients simutaneos) threads

    char* ip = "192.168.15.12"; //ip e port do servidor (para exemplo foi utilizado meu ip local)
    int port = 8080;

    int server = socket(AF_INET, SOCK_STREAM, 0);
    conn.sin_addr.s_addr = inet_addr(ip);
    conn.sin_family = AF_INET;
    conn.sin_port = htons(port);

    printf("\nCriando servidor em %s:%d com espaco para %d cliente(s) simutaneos\n", ip, port, maxClient);
    //Cria estrutura, utiliza o bind do socket na estrutura, e coloca como listener
    if (bind(server, (struct sockaddr *)&conn, sizeof(conn)) < 0) { printf("Cant connect"); exit(1);  };
    if (listen(server, maxClient) < 0) { printf("Cant connect"); exit(1); };

    struct sockaddr_in client; //Cria uma estrutura para os clients (socket)
    int client_id, client_size = sizeof(client);

    while (curClient < maxClient) //
    {
        //Trava a execucao na espera de um novo cliente
        client_id = accept(server, (struct sockaddr *)&client, (socklen_t*)&client_size);

        if (client_id >= 0){ //Caso tenha conectado sem erro
            printf("\nConexao: %d\n", client_id); //exibe o descriptor do cliente conectado
            pthread_create((handles + curClient), NULL, ClientFunction, &client_id); //Cria e executa uma thread para tal
            curClient++; //Incrementa o contador
        }

        else {
            printf("\nConexao falhou\n");
        }

    }
```

```
    //Desliga o servidor/socket no fim da execucao
    shutdown(server, SHUT_RDWR);
    close(server);

    for (int i = 0; i < curClient; i++)
    {
        //Libera a memoria das threads
        pthread_join(*(handles + i), NULL);
    }

    return 0;
}
```

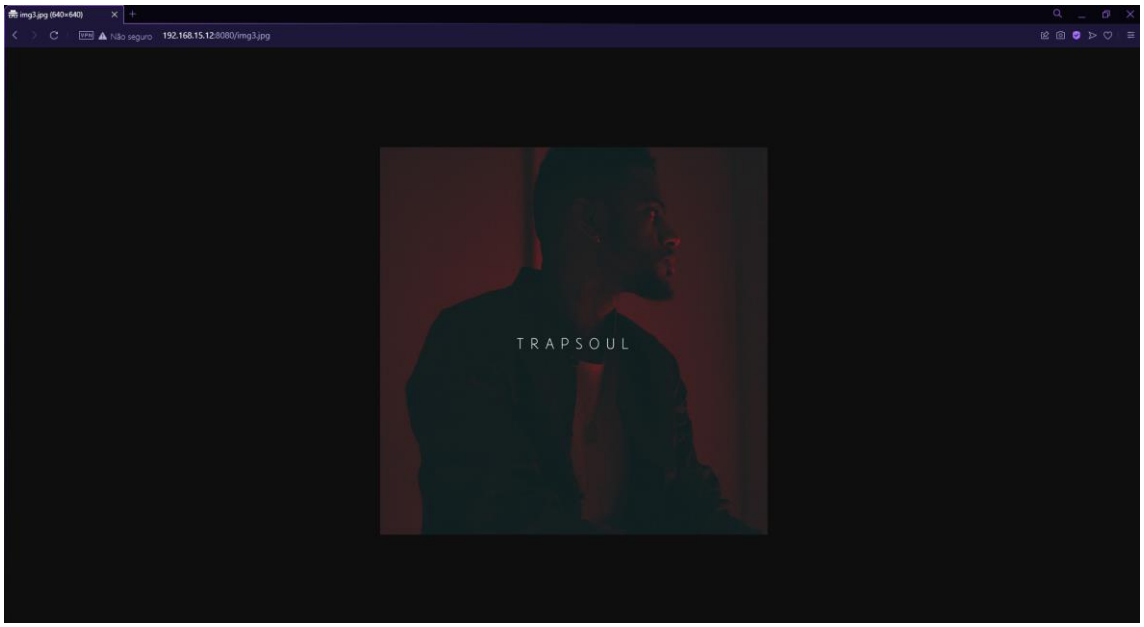
Testes:

Demonstrando a conectividade entre cliente (browser no Windows 192.168.15.3) e servidor (ServidorC.c no Linux 192.168.15.12):



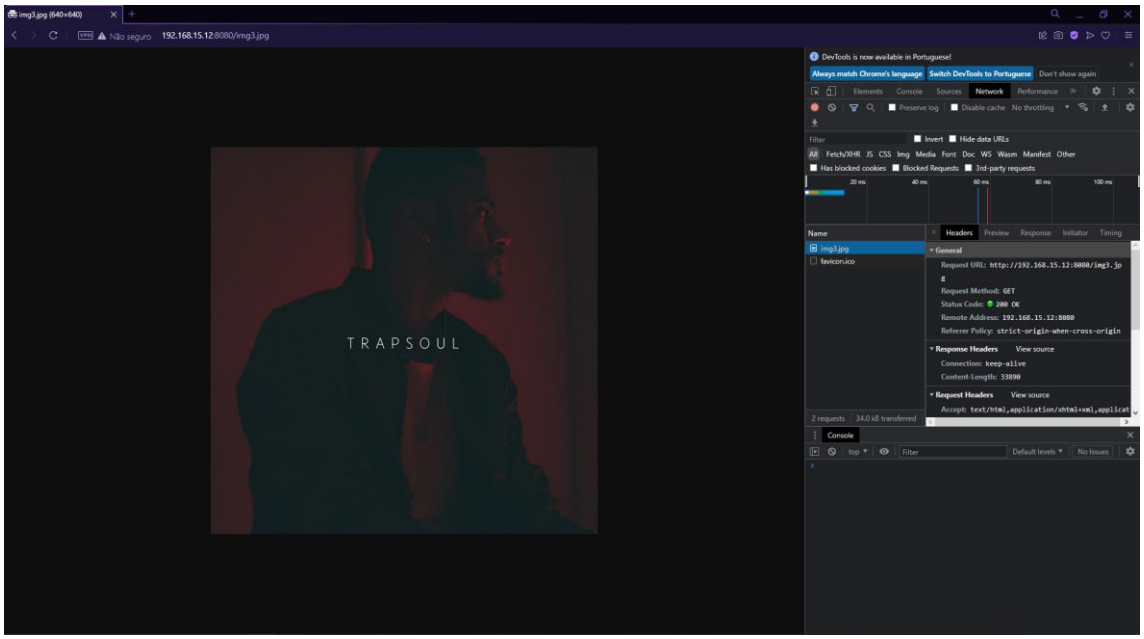
Execução e resultado (teste 1):

```
pedrou@pop-os:~/Documentos/VSCodeWS/Calebe$ gcc ServidorC.c -o ServidorC -pthread
pedrou@pop-os:~/Documentos/VSCodeWS/Calebe$ ./ServidorC 5

Criando servidor em 127.0.0.1:8080 com espaco para 5 cliente(s) simutaneos

Conexao: 4

Requisicao de 4:
GET /img3.jpg HTTP/1.1
Host: 192.168.15.12:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.52
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

Mandando img3.jpg para 4
```



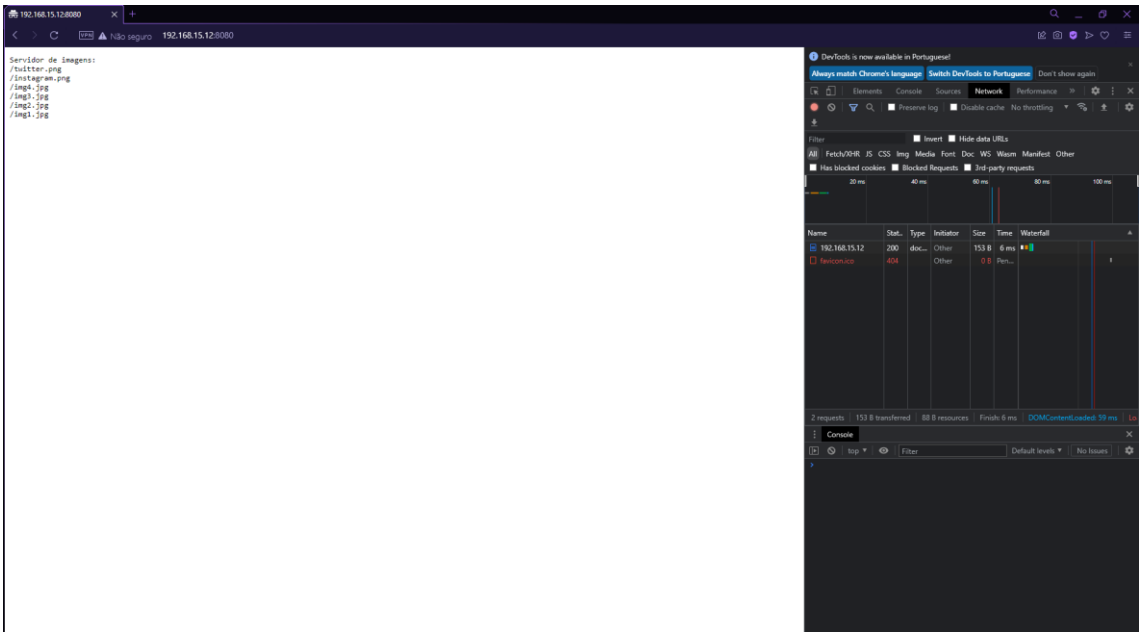
Execução e resultado (teste 2):

```
pedrou@pop-os:~/Documentos/VSCodeWS/Calebe$ gcc ServidorC.c -o ServidorC -pthread
pedrou@pop-os:~/Documentos/VSCodeWS/Calebe$ ./ServidorC 5

Criando servidor em 127.0.0.1:8080 com espaco para 5 cliente(s) simutaneos

Conexao: 4

Requisicao de 4:
GET /img3.jpg HTTP/1.1
Host: 192.168.15.12:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.52
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

Mandando img3.jpg para 4

Conexao: 5

Requisicao de 5:
GET /img3.jpg HTTP/1.1
Host: 192.168.15.12:8080
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.52
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

Mandando img3.jpg para 5
```



Execução e Resultados (teste 3):

```
Conexao: 8

Requisicao de 8:
GET / HTTP/1.1
Host: 192.168.15.12:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.52
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

Mandando pagina indice para 8

Conexao: 11

Requisicao de 11:
GET /favicon.ico HTTP/1.1
Host: 192.168.15.12:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.52
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.15.12:8080/
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

g: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

Mandando favicon.ico para 11
Falha na requisicao de 11, imagem não existente
```



Execução e Resultados (teste 4):

```
Conexao: 12

Requisicao de 12:
GET /img1.jpg HTTP/1.1
Host: 192.168.15.12:8080
User-Agent: curl/7.79.1
Accept: */*

ozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.52
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.15.12:8080/
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

g: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9

Mandando img1.jpg para 12
pedrou@pop-os:~/Documentos/VSCodeWS/Calebe$
```

```
C:\Users\pedro>curl 192.168.15.12:8080/img1.jpg
Warning: Binary output can mess up your terminal. Use "--output -" to tell
Warning: curl to output it to your terminal anyway, or consider "--output
Warning: <FILE>" to save to a file.

C:\Users\pedro>
```