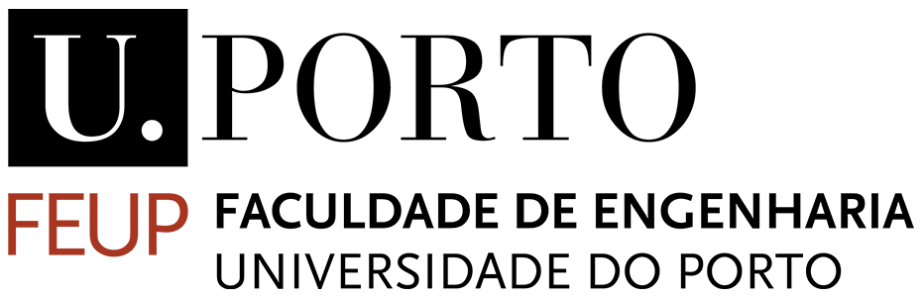
**2º Trabalho Laboratorial**

Relatório



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

**Trabalho realizado por:**

Maria Gonçalves Caldeira (up201704507)

Raul Manuel Fidalgo da Silva Teixeira Viana (up201208089)

18 de dezembro de 2020

**Índice**

[Sumário 3](#__RefHeading___Toc631_2610388941)

[Introdução 3](#__RefHeading___Toc633_2610388941)

[Parte 1: Aplicação de download 3](#__RefHeading___Toc635_2610388941)

[Arquitetura 3](#__RefHeading___Toc637_2610388941)

[Resultados 4](#__RefHeading___Toc639_2610388941)

[Parte 2: Configuração da Rede 4](#__RefHeading___Toc641_2610388941)

[Experiência 1 - Configurar um endereço de IP de rede 4](#__RefHeading___Toc643_2610388941)

[Experiência 2 – Implementação de duas LAN’s virtuais no switch 5](#__RefHeading___Toc645_2610388941)

[Experiência 3 – Configurar um router em Linux 6](#__RefHeading___Toc647_2610388941)

[Experiência 4 – Configurar um router em Linux 7](#__RefHeading___Toc649_2610388941)

[Experiência 5 – DNS 8](#__RefHeading___Toc651_2610388941)

[Experiência 6 – Ligação TCP 8](#__RefHeading___Toc653_2610388941)

[Conclusões 9](#__RefHeading___Toc655_2610388941)

[Anexo 1 11](#__RefHeading___Toc657_2610388941)

[Imagens 11](#__RefHeading___Toc659_2610388941)

[Anexo 2 17](#__RefHeading___Toc661_2610388941)

[Código da aplicação de *download* 17](#__RefHeading___Toc663_2610388941)

# **Sumário**

Este relatório foi elaborado no âmbito da unidade curricular de Redes e Computadores, do curso Mestrado Integrado em Engenharia Informática e Computação, para descrever o segundo trabalho prático, que consistiu no desenvolvimento de uma aplicação capaz de realizar o *download* de um ficheiro através da utilização do protocolo FTP (*File Transfer Protocol*) e na montagem e configuração de uma rede privada.

Assim, é possível afirmar que o trabalho foi concluído com sucesso, visto que os objetivos estabelecidos foram cumpridos, tendo sido configurada uma rede e realizado o *download* de um ficheiro através desta.

# **Introdução**

Os objetivos deste trabalho foram o desenvolvimento de uma aplicação de *download* e a configuração de uma rede. Esta rede irá permitir o funcionamento correto da aplicação a partir da criação de duas VLAN’s dentro de um *switch*.

De forma a descrever o trabalho laboratorial realizado este relatório está divido na seguinte estrutura:

**Parte 1**: Aplicação de download: arquitetura e resultados;

**Parte 2**: configuração e análise da rede: análise e descrição de cada experiência;

**Conclusão**: resumo da informação apresentada nas secções anteriores e conclusões finais.

# **Parte 1: Aplicação de download**

A primeira parte deste trabalho consistiu no desenvolvimento de uma aplicação de *download*, escrita em C. Esta aplicação aceita como argumento um *link* do tipo:

ftp://<username>:<password>@<host>/<url-path>

Através do processamento deste argumento a aplicação será capaz de qualquer ficheiro que esteja alojado num servidor FTP.

## **Arquitetura**

A aplicação está dividida em dois ficheiros principais: *main.c* e *handlers.c.* No segundo estão definidas as funções necessárias à manipulação dos *url’s*, dos *sockets* e da ligação FTP.

O processamento é iniciado pelo *parse* do *url*. É criada a estrutura “url” que contém espaço para as variáveis *user, password*, *host*, *path*, *filename* e *ip\_adress*. Seguidamente é feito o *parsing* do argumento recebido de forma a preencher estas variáveis. É corrida também a função get\_ip() que converte o nome do *host* num endereço de *ip*. A porta utilizada é a 21.

Posteriormente é criado um *socket* através do qual serão transferidos os comandos e lidos as respostas do servidor. É feito o *login* enviando os comandos “USER” e “PASS” e posteriormente é pedido ao servidor que entre em modo passivo através do envio do comando “PASV”. A resposta do servidor a este comando é interpretada pela função ftp\_passive\_mode() que calcula o endereço de ip e a porta através dos quais se liga um novo *socket*, desta feita para a transferência do ficheiro. Assim é enviado o comando “RETR” através do primeiro *socket* e recebido o ficheiro através do segundo.

Por fim são fechadas as conexões e o ficheiro criado.

## Resultados

  A aplicação foi testada com ficheiros de diferentes tamanhos e tipos. É apresentado o estado do processamento na consola para mais fácil controlo por parte do utilizador.

# **Parte 2: Configuração da Rede**

## Experiência 1 - Configurar um endereço de IP de rede

O objetivo desta experiência foi ligar o *tuxy3* ao *tuxy4* utilizando um *switch* e adquirir conhecimento necessário para responder às seguintes perguntas:

1. **O que são pacotes ARP?**

O protocolo *Address Resolution Protocol* (ARP) é um protocolo de comunicação utilizado para um determinado computador descobrir o endereço da camada de ligação associado ao endereço de IP. Serve para mapear o endereço de rede a um endereço físico, por exemplo um endereço MAC.

1. **Quais são os endereços MAC e IP dos pacotes ARP e porquê?**

Ao usar o comando ***ifconfig*** no *tuxy3*, pudemos verificar que o seu IP tinha sido bem configurado, tendo ele sido configurado como *172.16.40.1*, e que o seu endereço MAC era *00:21:5a:61:2f:98*. No *tuxy4* o procedimento foi o mesmo, e o seu IP foi configurado como *172.16.40.254* e o seu endereço MAC era *00:21:5a:c3:78:76.* Consultar *logs* da figura 2 do anexo 1.

Posteriormente o *tux3* responde, dizendo que é ele que tem aquele *IP*  enviando o seu endereço MAC.

O pacote de resposta presente na figura 3 contem o endereço IP e MAC da origem, que neste caso é o *tux3* e do destino - *tux4* .

1. **Quais pacotes são gerados pelo comando ping?**

O comando *ping* é utlizado para testar a conectividade entre o tuxy3 e o tuxy4 e gera tanto pacotes ARP (onde obtém o endereço MAC) como pacotes ICMP.

1. **Quais são os endereços MAC e IP dos pacotes ping?**

Os endereços de origem e destino dos pacotes vão ser os designados na tabela seguinte:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **MAC** | | **IP** | |
|  | Origem | Destino | Origem | Destino |
| **Pacote Pedido** | 00:21:5a:61:2f:98 | 00:21:5a:c3:78:76 | 172.16.40.1 | 172.16.40.254 |
| **Pacote Resposta** | 00:21:5a:c3:78:76 | 00:21:5a:61:2f:98 | 172.16.40.254 | 172.16.40.1 |

Tabela 1: endereços de origem e destino dos pacotes ARP no ping

1. **Como determinar se a trama Ethernet recebida é do tipo ARP, IP ou ICMP?**

É possível obter esta informação inspecionando o cabeçalho de uma trama Ethernet. Se o valor for 0x0800, representa um pacote do tipo IP, sendo que neste caso também é possível analisar o IP header. Se este tomar o valor de 1, quer dizer que se trata de um protocolo do tipo ICMP. Contudo, se o valor for 0x806, representa um pacote do tipo ARP.

1. **Como determinar o comprimento de uma trama recebida?**

Através da utilização do Wireshark é possível inspecionar a trama e observar o seu comprimento. Consultar figura 4 do anexo 1 que representa uma trama do tipo ARP com 60 bytes de comprimento.

1. **O que é a interface loopback e qual a sua importância?**

A interface loopback é uma interface de rede virtual que permite que o computador comunique com ele mesmo. É um mecanismo utilizado para testar a correta configuração da rede, permitindo a existência de um IP sempre ativo no router, o que descarta a dependência numa interface física. A figura 5 mostra uma trama *loopback.*

## **Experiência 2 – Implementação de duas LAN’s virtuais no switch**

Nesta experiência foram criadas duas LAN’s virtuais, *vlany0* e *vlany1*. Os computadores *tuxy3* e *tuxy4* foram adicionados à primeira, enquanto que o computador *tuxy2* foi adicionado à segunda.

1. **Como configurar *vlanyo*?**

A configuração física da *vlany0* passa por realizar as ligações corretas. Na régua 1 a porta *cisco->RS232* terá que ser ligada à porta do *switch* na régua 2. O *tux* que se pretende que esteja ligado ao *switch* tem de ter a sua porta S0 ligada à porta *RS232->cisco* da régua 1. De seguida são introduzidos os seguintes comandos no *gtkterm* do *tux* a configurar:

* *conf t* (ou *configure terminal*)
* *vlan y0*
* *end*
* *show vlan brief* (para verificar se a vlan foi criada)

Adicionar portas (porta do tuxy3 e do tuxy4):

* *conf t* (ou *configure terminal*)
* *interface fastethernet 0/[nº da porta]*
* *switchport mode access*
* *switch access vlan y0*
* *end*

1. **Quantos domínios de transmissão existem?**

Nesta configuração existem dois domínios de transmissão. Quando os *tuxy3 e tuxy4* fazem *ping broadcast* apenas recebem resposta um do outro e não do *tuxy2*. O *tuxy3* recebe do *tuxy4* e vice-versa. Por outro lado, quando o *tuxy2* faz *ping broadcast* não recebe qualquer resposta, consultar figura 6. Assim é possível afirmar que existem dois domínios de transmissão e ainda que os *tuxy3 e tuxy4* pertencem a um e o *tuxy2* pertence a outro.

## **Experiência 3 – Configurar um router em Linux**

1. **Que rotas existem nos *tuxes*? Qual o seu significado?**

* No *tuxy3* há uma rota para a vlan y0 (*172.16.y0.0*) – *gateway 172.16.y0.1.* Ao longo da experiência foi criada outra rota para a vlan y1 (*172.16.y1.0*) - *gateway 172.16.y0.254.*
* No *tuxy4* há uma rota para a vlan y0 (*172.16.y0.0*) – *gateway 172.16.y0.254;* e outra rota para a vlan y1 (*172.16.y1.0*) – *gateway 172.16.y1.253.*
* No *tuxy2* há uma rota para a vlan y1 (*172.16.y1.0*) – *gateway 172.16.y1.1.* Ao longo da experiência foi criada outra rota para a vlan y0 (*172.16.y0.0*) - *gateway 172.16.y1.253.*

O destino de cada rota corresponde ao alcance de cada uma delas.

1. **Que informação é que uma entrada de uma tabela de forwarding contém?**

Estas tabelas são obtidas através do comando *“route -n”* e contém informação sobre o destino da rota (***Destination***), o ip do ponto em que a rota vai passar (***Gateway***), a máscara (***Netmask***), as ***Flags***, o “custo” de cada rota (***Metric***), o número de referências para a rota (***Ref***), o contador de pesquisas pelas rotas (***Use***) e a ***Interface***.

1. **Que mensagens ARP e endereços MAC associados são observados e porquê?**

Utilizando o comando *ping* é possível analisar a sequência existente entre mensagens ARP e endereços MAC. Caso um tux dê *ping* a outro tux e este segundo não reconhecer o endereço MAC do primeiro, irá enviar lhe uma mensagem ARP a perguntar-lhe qual é o endereço MAC que corresponde àquele ip. Esta mensagem terá o MAC do tux de origem e como o tux de destino ainda não é conhecido, terá o endereço *00:00:00:00:00:00.* Logo a seguir, é enviada outra mensagem ARP agora a partir do tux de destino com o seu endereço MAC e, como é obvio, esta mensagem também terá associado o MAC do tux de origem. O registo destas mensagens ARP e nedereços MAC associados podem ser observados nas imagens 7, 8, 9 e 10 do anexo 1.

1. **Que pacotes ICMP são observados e porquê?**

São observados pacotes ICMP de *requests* e *replies* porque foram adicionadas durante a experiência rotas que permitem todos os tux’s conseguirem alcançar todos os outros.

1. **Quais os endereços IP e MAC associados a um pacote ICMP e porquê?**

Associados a um pacote ICMP estão os endereços IP e MAC do tux de origem e do de destino. Estes são atribuídos ao ICMP quando o tux de destino é alcançável e se o MAC estiver mapeado.

## **Experiência 4 – Configurar um router em Linux**

Devido a constrangimentos inerentes à pandemia de Covid19 que atravessamos, o tempo de acesso aos laboratórios foi drasticamente reduzido. Houve também uma redução do número de aulas previsto devido a feriados o que tornou ainda mais difícil a presença física nos laboratórios. Isto levou a que não fosse possível registar *logs*  das experiências 5 e 6. Relativamente à experiência 4 apenas foi possível registar *log* do ponto 3. No entanto foram utilizados *logs* de outros grupos por sugestão do professor.

1. **Como se configura uma rota estática num router comercial?**

Primeiro tem de se ligar a porta *cisco->RS232* da régua 1 à porta do router da régua 2 (que se encontra do lado direito da porta do *switch*). O *tux* que se pretende que esteja ligado ao *router* tem de ter a sua porta S0 ligada à porta *RS232->cisco* da régua 1. Para se proceder à configuração de uma rota estática num router é necessário correr os seguintes comandos:

* conf t
* ip route 0.0.0.0 0.0.0.0 172.16.2.254
* ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
* end

1. **Quais são as rotas seguidas pelos pacotes nos testes efetuados e porquê?**

Se a rota existir, essa mesma rota será usada pelos pacotes. Se tal não acontecer, os pacotes irão até ao *router*, que é a rota por *default*, e este dá informações sobre a existência do *tux* e envia as informações pelo mesmo.

1. **Como se configura o NAT num router comercial?**

Os comandos necessários para a configuração do NAT estão presentes na *Figura 1* dos *Anexos* e tais comandos foram encontrados no guião deste projeto. Todos os comandos foram executados no *gtkterm.*

1. **O que faz o NAT?**

O NAT (*Network Address Translation*) permite que redes IP privadas, ou seja, redes com endereços IP que não estejam registados se consigam ligar a uma rede pública ou à Internet, por intermédio, neste caso, do *router*. O NAT também ainda tem funções adicionais de segurança.

## **Experiência 5 – DNS**

1. **Como configurar o serviço DNS num *host*?**

Para configurar o DNS, em primeiro lugar tem de se editar o ficheiro */etc/resolv.conf***.** Este ficheiro tem de ser alterado de acordo com a seguinte informação: **search** **netlab.fe.up.pt** (como nome do servidor DNS) e **nameserver 172.16.1.1** (como endereço de IP).

1. **Que pacotes são trocados pelo DNS e que informações são transportadas?**

Em termos de pacotes, o host envia um pacote para o servidor. Este pacote, que tem associado o *hostname*, vai esperar no servidor por receber o seu IP, sendo que o *receiver*, nesta sequência, envia outro pacote com o IP do *host.*

## **Experiência 6 – Ligação TCP**

1. **Quantas conexões TCP foram abertas pela aplicação FTP?**

São abertas 2 conexões TCP pela aplicação FTP, uma de controlo e outra de dados.

1. **Em que conexão é transportado o controlo de informação FTP?**

A informação de controlo FTP é transportada pela conexão de controlo.

1. **Quais são as fases da conexão TCP?**

Existem 3 fases: o estabelecer da conexão, a transferência de dados e o fim da conexão.

1. **Como funciona o mecanismo ARC TCP? Quais os campos TCP relevantes? Que informação relevante pode ser observada nos logs?**

O ARC TCP assenta num mecanismo de janela deslizante com controlo de erros na transmissão de dados. Para isso o emissor coloca diversas informações em diferentes campos das mensagens. **Acknowledgment Numbers**, que indicam que a trama foi recebida corretamente, **window size**, que especifica o numero de pacotes que o emissor pode enviar sem receber confirmação e **sequence number,** que informa o numero do pacote a ser enviado.

**5. Como é que o mecanismo de controlo de congestão TCP**

**funciona? Como é que o fluxo de dados da conexão evoluiu ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?**

O mecanismo de controlo de congestão mantém na rede um numero de pacotes estimado para essa rede. Não são enviados mais pacotes do que o mínimo da janela definida pelo recetor e pela janela de congestão.

**6. De que forma é afetada a conexão de dados TCP pelo aparecimento de uma segunda conexão TCP? Como?**

Com o aparecimento de uma segunda conexão TCP, uma transferência de dados pré-existente sofre uma queda na taxa de transmissão, uma vez que a taxa de transferência passa a ser distribuída de igual forma para cada ligação.

# **Conclusões**

O segundo trabalho laboratorial da unidade curricular Redes de Computadores teve como objetivos a configuração de uma rede e a implementação de um cliente de *download*.

Neste trabalho foram consolidados e integrados novos conceitos relacionados com funcionalidades dos equipamentos de redes e o seu funcionamento e configuração.

O trabalho foi concluído com sucesso, uma vez que levou à aprendizagem dos conceitos propostos, sendo que por diversos constrangimentos, não tenha sido possível desenvolver o trabalho em laboratório de forma tão exaustiva como pretendido.

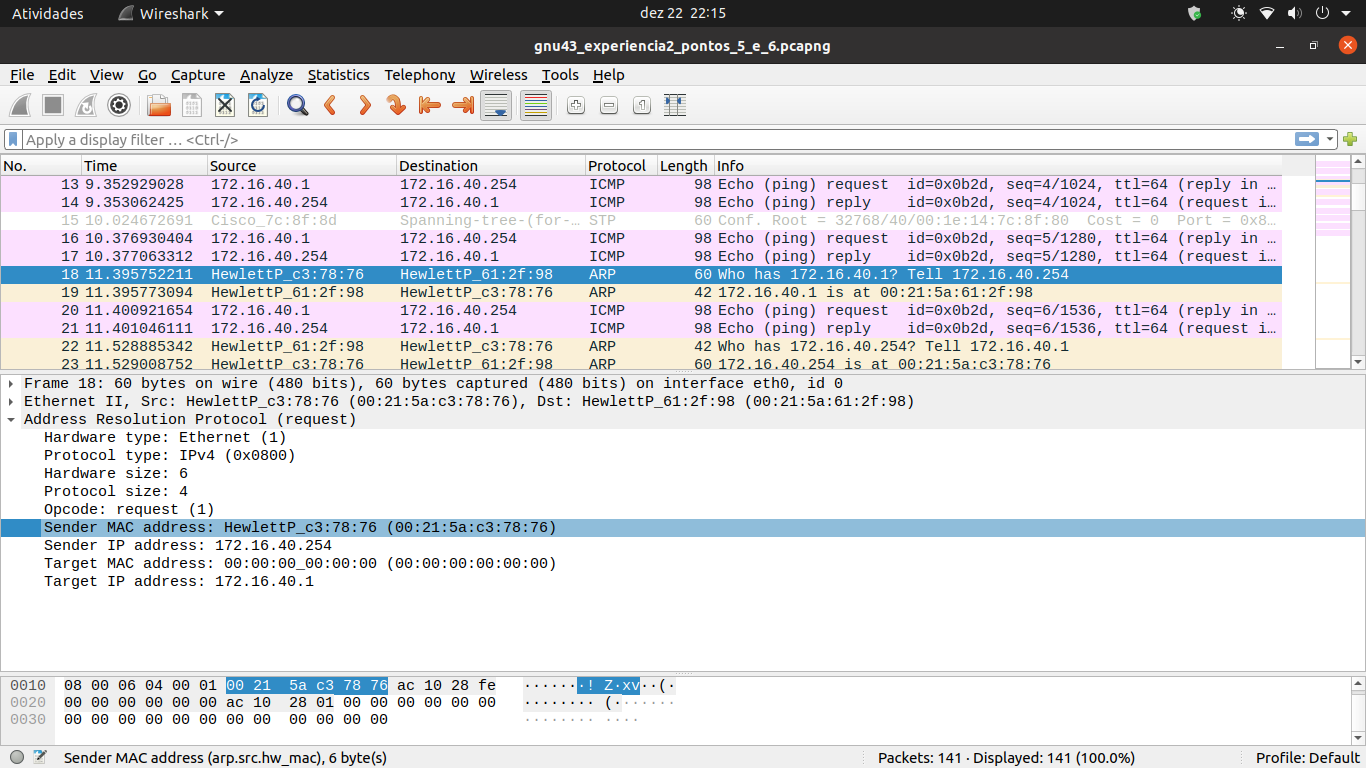
# Anexo 1

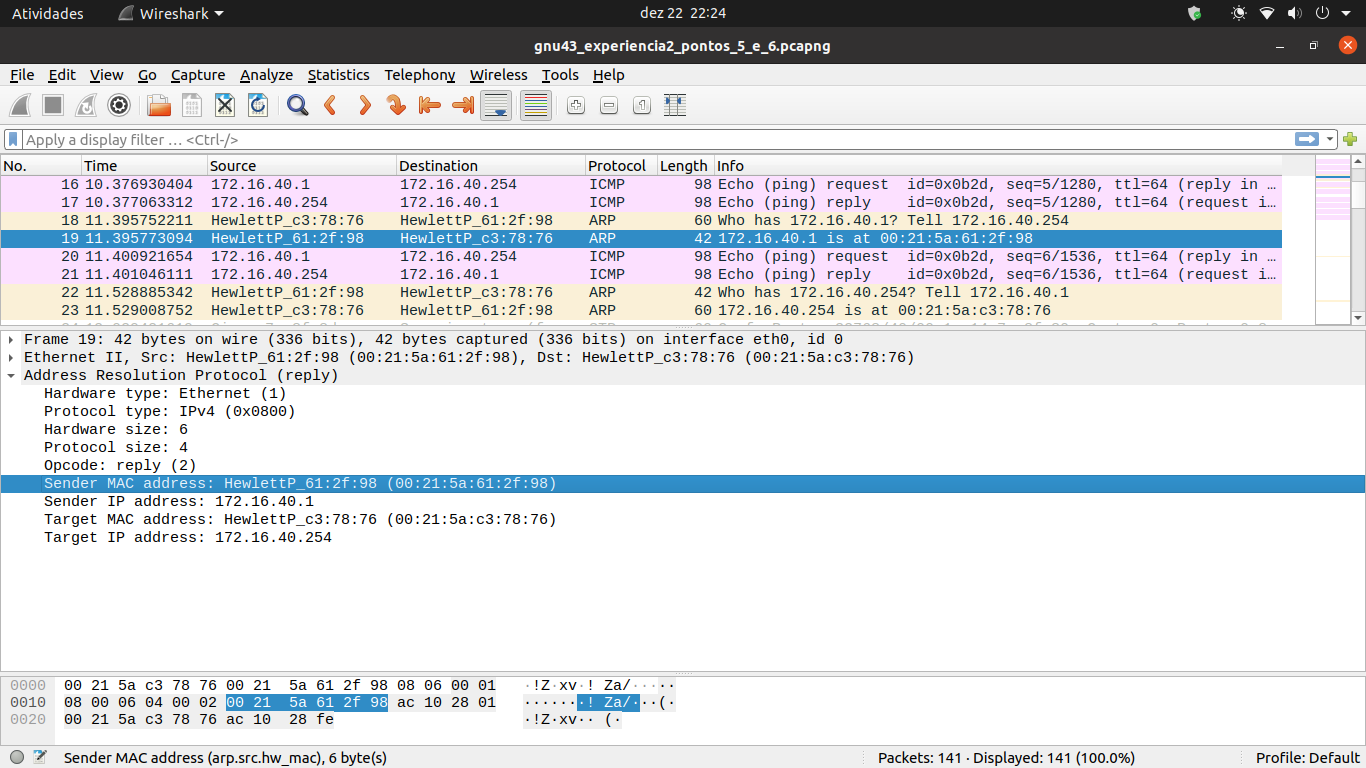
## Imagens

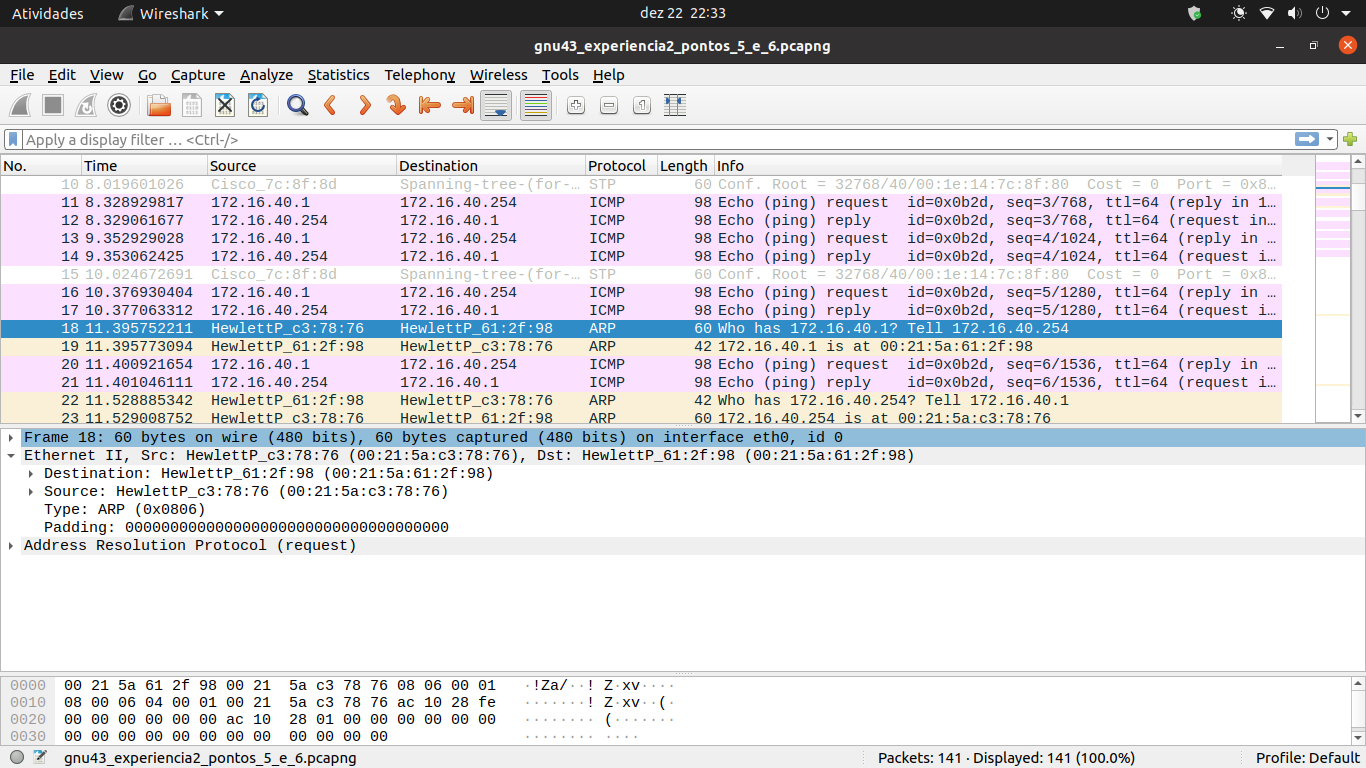
Uma imagem com texto

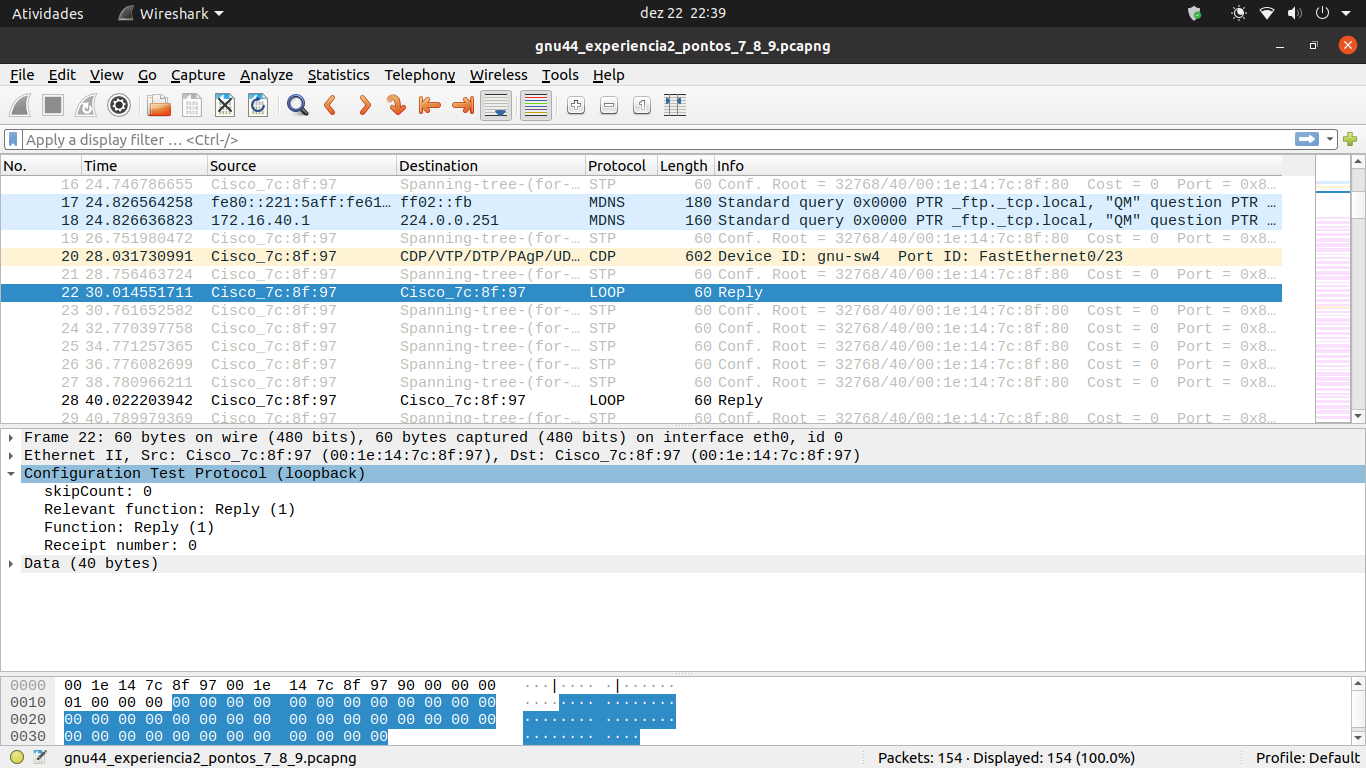
Descrição gerada automaticamente

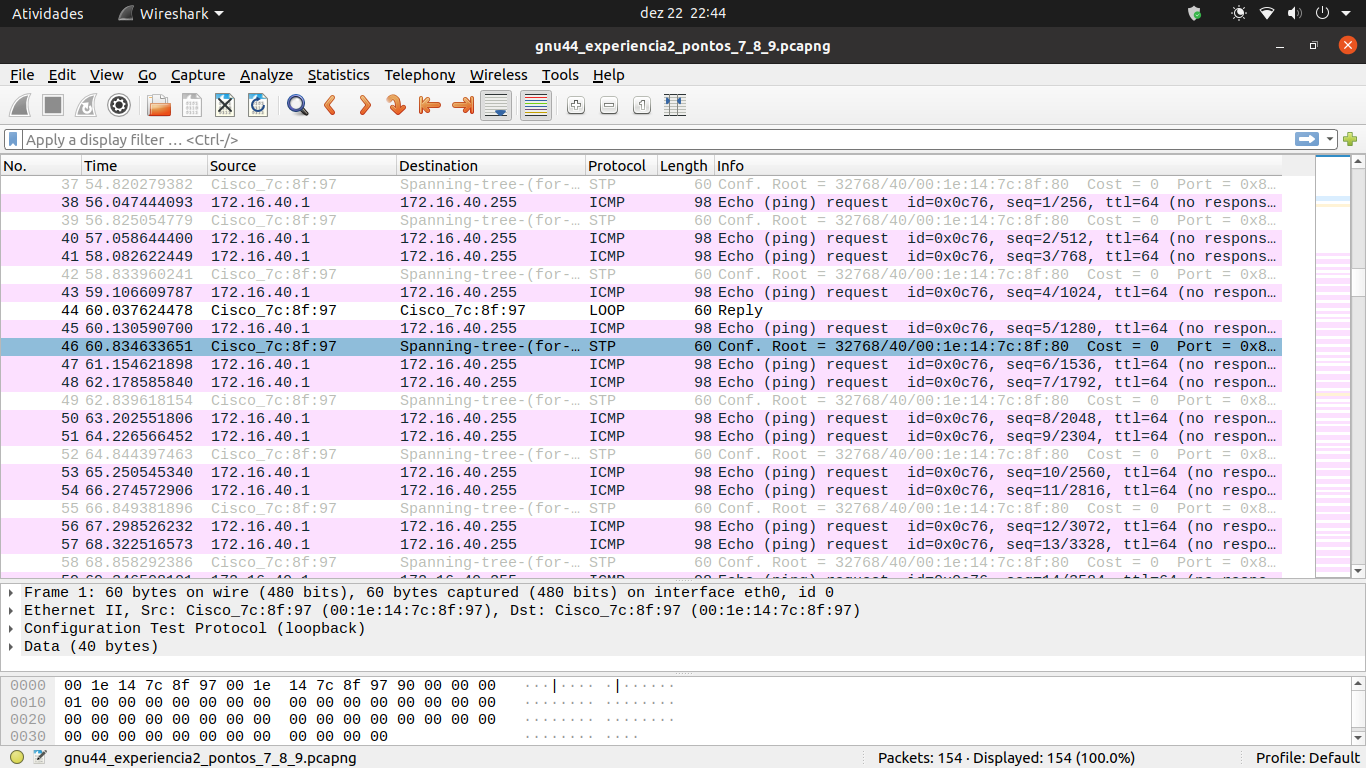
Figura 1: Configuração do NAT

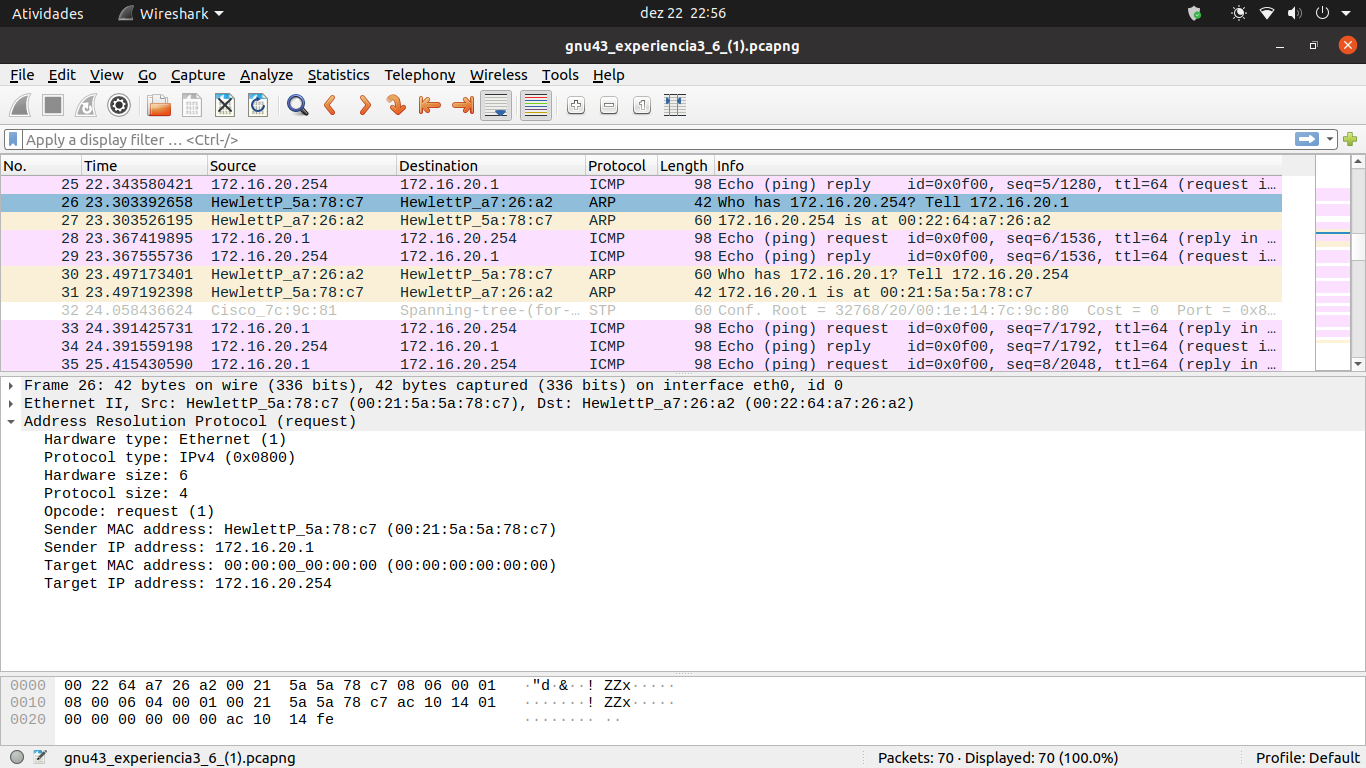
Figura 2

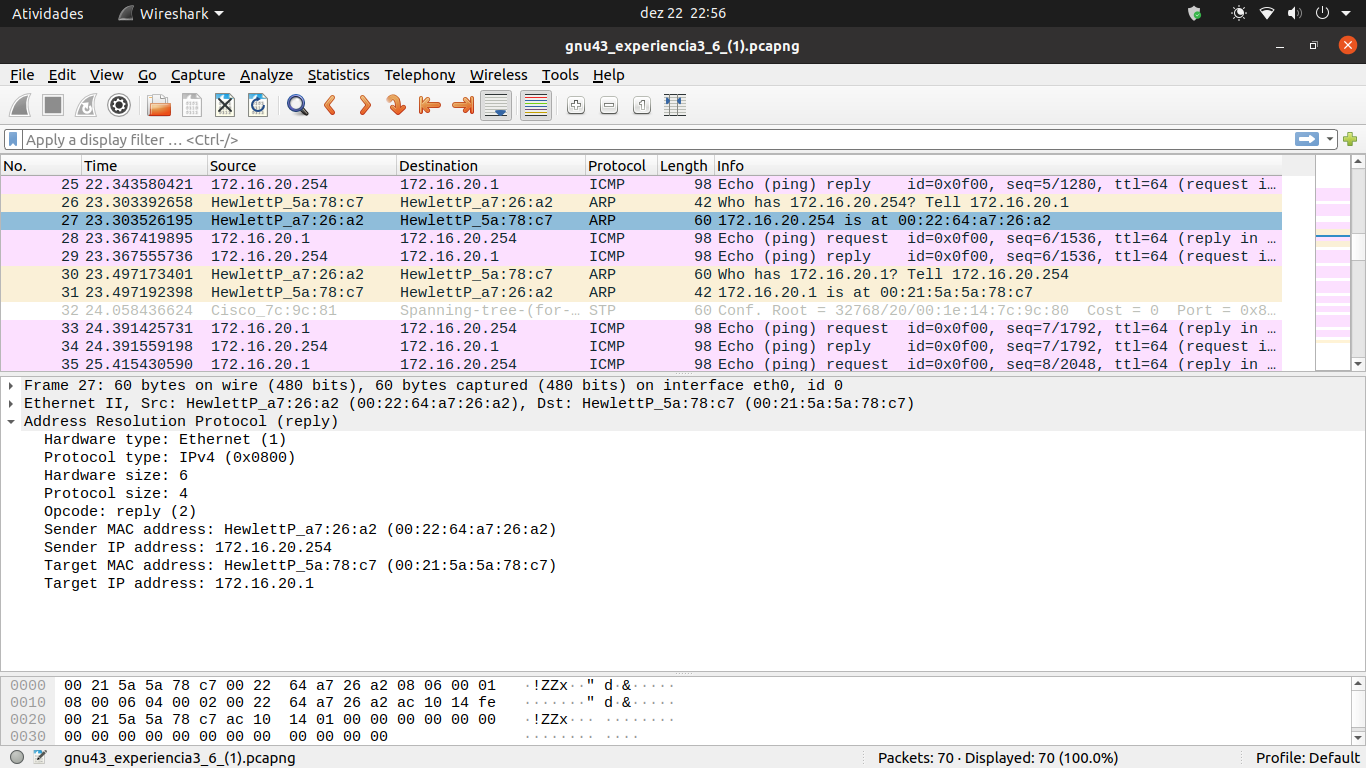
Figura 3

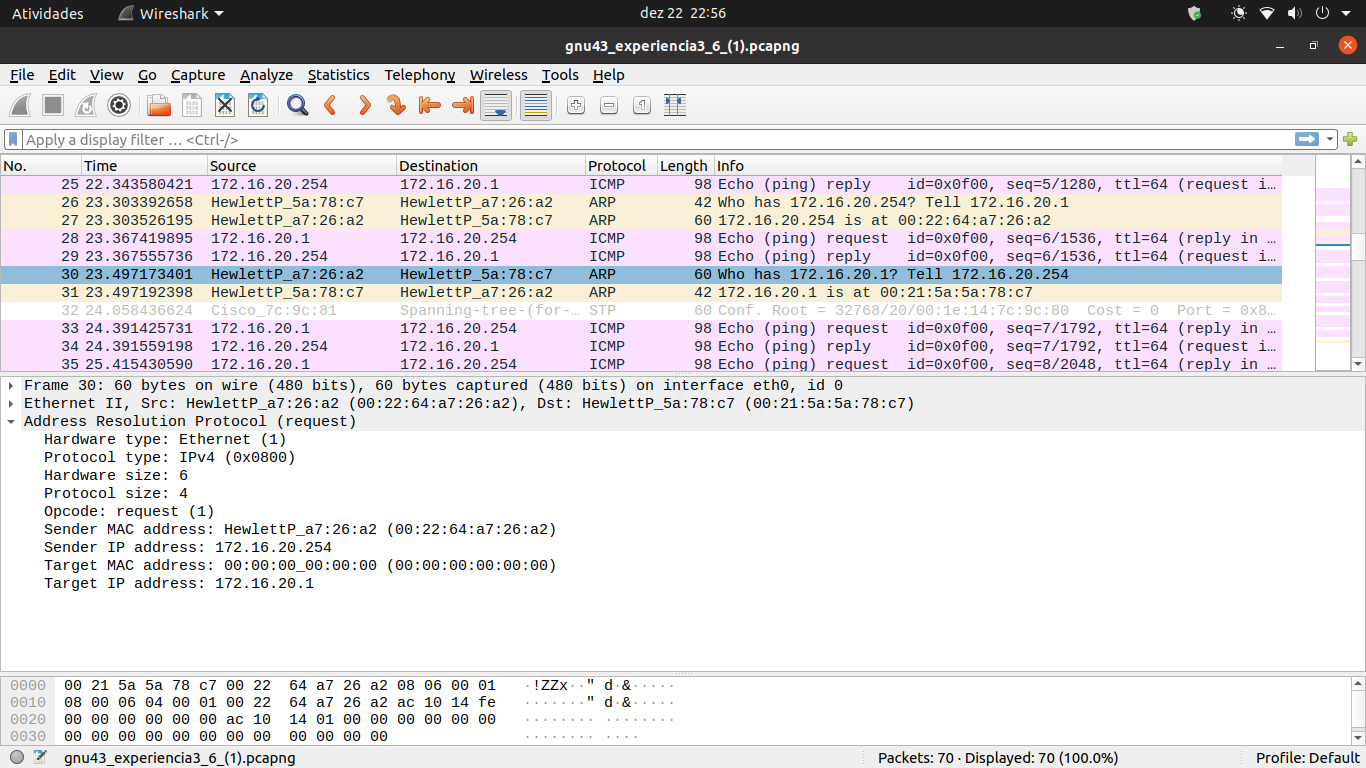
Figura 4

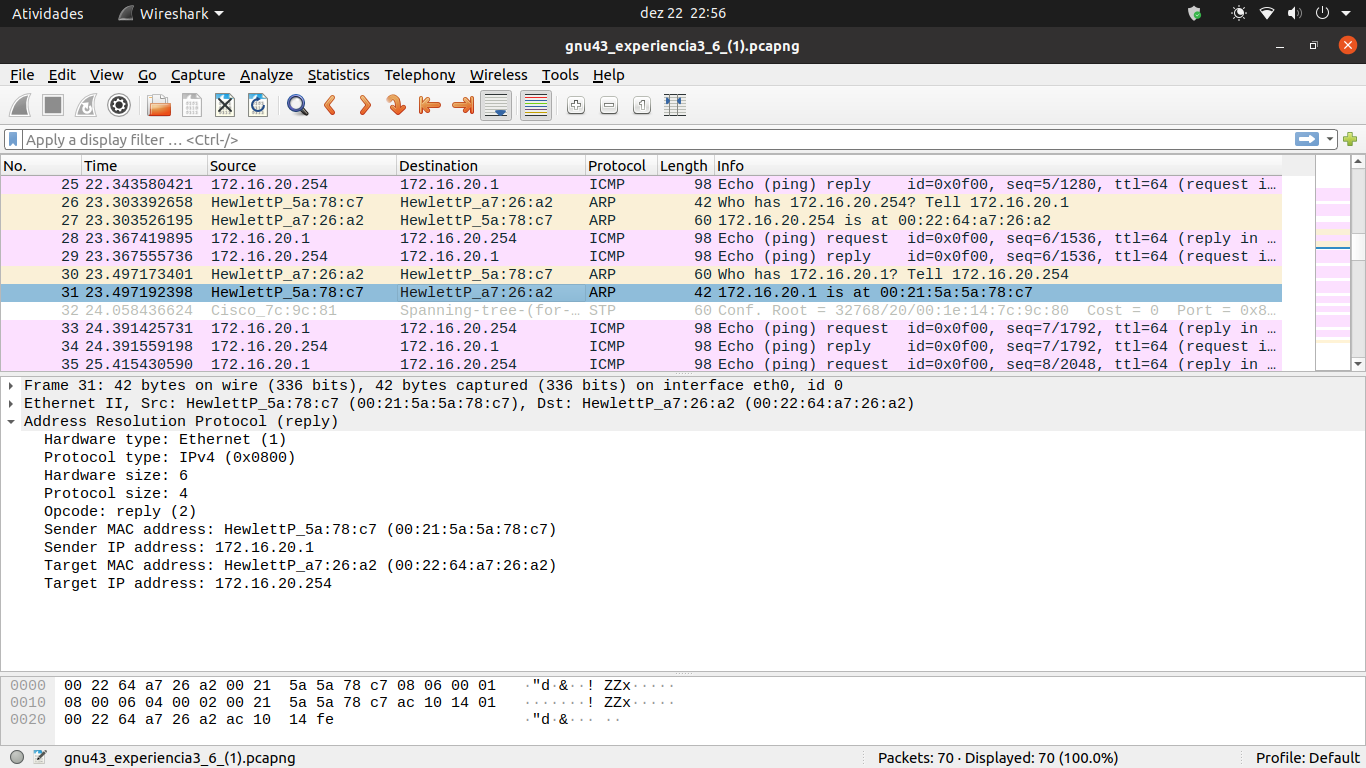
Figura 5

Figura 6

Figura 7

Figura 8

Figura 9

Figura 10

# Anexo 2

## Código da aplicação de *download*

main.c

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <netdb.h>

#include <string.h>

#include <sys/stat.h>

#include <fcntl.h>

#include "handlers.h"

/\* gets ip address according to the host's name \*/

int getip(char host[], url\* url)

{

struct hostent \*h;

if ((h = gethostbyname(host)) == NULL)

{

perror("gethostbyname");

exit(1);

}

strcpy(url->ip\_address, inet\_ntoa(\*((struct in\_addr \*)h->h\_addr\_list[0])));

return 0;

}

int new\_socket\_id;

int main(int argc, char\*\* argv){

if(argc < 2){

printf("Error: You must input an url.\n Usage: ./download ftp://ftp.up.pt/pub/..\n");

exit(2);

}

int sockfd;

struct sockaddr\_in server\_addr;

char buf[] = "Mensagem de teste na travessia da pilha TCP/IP\n";

int bytes;

char command[MAX\_COMMAND\_LENGTH];

char response[MAX\_STRING\_LENGTH];

url url;

create\_url\_struct(&url);

parseURL(argv[1], &url);

char \*path = url.url\_path;

parseFilename(path, &url);

printf(" - Username: %s\n", url.user);

printf(" - Password: %s\n", url.password);

printf(" - Host: %s\n", url.host);

printf(" - Path: %s\n", url.url\_path);

printf(" - Filename: %s\n", url.filename);

getip(url.host, &url);

printf(" - IP: %s\n", url.ip\_address);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* start connection \*/

int socket\_id;

char buffer[MAX\_STRING\_LENGTH];

if((socket\_id = open\_socket(url.ip\_address, FTP\_PORT)) < 0){

perror("Error in open\_socket()\n");

exit(2);

}

printf("\n");

read\_socket(socket\_id, response);

printf("FTP connection established\n");

/\* Log In in ftp server \*/

//user

sprintf(command, "USER %s\r\n", url.user);

if(ftp\_send\_command(socket\_id, command, strlen(command), response)){

perror("Error sending command USER\n");

exit(3);

}

//pass

bzero(command, MAX\_COMMAND\_LENGTH);

bzero(response, MAX\_STRING\_LENGTH);

sprintf(command, "PASS %s\r\n", url.password);

if(ftp\_send\_command(socket\_id, command, strlen(command), response)){

perror("Error sending command PASSWORD\n");

exit(4);

}

//enter passive mode

bzero(command, MAX\_COMMAND\_LENGTH);

bzero(response, MAX\_STRING\_LENGTH);

sprintf(command, "PASV\r\n", url.password);

if (ftp\_passive\_mode(socket\_id, command, strlen(command), response)) {

perror("Error in ftp\_passive\_mode()\n");

exit(5);

}

printf("Entered passive mode successfully\n\n");

//reconstruct file path

char filepath[MAX\_STRING\_LENGTH];

bzero(command, MAX\_COMMAND\_LENGTH);

bzero(response, MAX\_STRING\_LENGTH);

sprintf(filepath, "%s", url.url\_path);

if (ftp\_retrieve\_file(socket\_id, filepath, command, response)) {

perror("ftp\_retr\_file()");

exit(6);

}

//download file

char filename[MAX\_STRING\_LENGTH];

strcpy(filename, url.filename);

if (ftp\_download\_file(new\_socket\_id, filename)) {

perror("ftp\_download\_file()\n");

exit(7);

}

printf("Downloaded file %s successfully\n", url.filename);

close(socket\_id);

}

handlers.h

#pragma once

#include <string.h>

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <signal.h>

#include <netdb.h>

#include <string.h>

#include <sys/stat.h>

#include <fcntl.h>

#define MAX\_STRING\_LENGTH 255

#define MAX\_COMMAND\_LENGTH 512

#define TRUE 1

#define FALSE 0

#define FTP\_PORT 21

typedef struct url{

char user[MAX\_STRING\_LENGTH];

char password[MAX\_STRING\_LENGTH];

char host[MAX\_STRING\_LENGTH];

char url\_path[MAX\_STRING\_LENGTH];

char filename[MAX\_STRING\_LENGTH];

char ip\_address[MAX\_STRING\_LENGTH];

} url;

extern int new\_socket\_id;

//url handling

void create\_url\_struct(url\* url);

void parseURL(char \*argument, url \*url);

void parseFilename(char \*path, url \*url);

//ftp handling

int ftp\_send\_command(int socket\_id, const char\* command, int command\_size, char\* response);

int ftp\_retr(int socket\_id, const char\* command, int command\_size);

int ftp\_passive\_mode(int socket\_id, char\* command, size\_t command\_size, char\* response);

int ftp\_retrieve\_file(int socket\_id, char\* filepath, char\* command, char\* response);

int ftp\_download\_file(int new\_socket\_id, char\* filename);

//socket handling

int open\_socket(const char\* ip\_address, const int port);

int read\_socket(int socket\_id, char\* response);

handlers.c

#include "handlers.h"

void create\_url\_struct(url\* url) {

memset(url->user, 0, MAX\_STRING\_LENGTH);

memset(url->password, 0, MAX\_STRING\_LENGTH);

memset(url->host, 0, MAX\_STRING\_LENGTH);

memset(url->url\_path, 0, MAX\_STRING\_LENGTH);

memset(url->filename, 0, MAX\_STRING\_LENGTH);

memset(url->ip\_address, 0, MAX\_STRING\_LENGTH);

}

void parseURL(char \*argument, url \*url){

// ftp://[<user>:<password>@]<host>/<url-path>

char \*user, \*password, \*host, \*url\_path;

int length = strlen(argument);

bool userAndPass = false;

if(argument[0] != 'f' || argument[1] != 't' || argument[2] != 'p' ||

argument[3] != ':' || argument[4] != '/' || argument[5] != '/'){

printf("Error parsing ftp://\n");

exit(1);

}

//Checks if the url has the user and password part

for(int i = 0; i < length; i++){

if(argument[i] == '@'){

userAndPass = true;

}

}

if(userAndPass){

//Get user

const char user\_deli[] = ":";

user = strtok(&argument[6], user\_deli);

strcpy(url->user, user);

//Get password

const char pass\_deli[] = "@";

int pass\_start\_index = strlen(user) + 7;

password = strtok(&argument[pass\_start\_index], pass\_deli);

strcpy(url->password, password);

//Get host

const char host\_deli[] = "/";

int host\_start\_index = pass\_start\_index + strlen(password) + 1;

host = strtok(&argument[host\_start\_index], host\_deli);

strcpy(url->host, host);

//Get url-path

const char url\_deli[] = "\n";

int url\_start\_index = host\_start\_index + strlen(host) + 1;

url\_path = strtok(&argument[url\_start\_index], url\_deli);

strcpy(url->url\_path, url\_path);

}

else{

memset(url->user, 0 ,sizeof(url->user));

strcpy(url->user, "anonymous");

memset(url->password, 0 ,sizeof(url->password));

strcpy(url->password, "");

//Get host

const char host\_deli[] = "/";

host = strtok(&argument[6], host\_deli);

strcpy(url->host, host);

//Get url-path

const char url\_deli[] = "\n";

int url\_start\_index = strlen(host) + 7;

url\_path = strtok(&argument[url\_start\_index], url\_deli);

strcpy(url->url\_path, url\_path);

}

}

void parseFilename(char \*path, url \*url){

char filename[MAX\_STRING\_LENGTH];

char path\_to\_remove[MAX\_STRING\_LENGTH];

strcpy(filename, path);

while (strchr(filename, '/')) {

const char path\_deli[] = "/";

strcpy(path\_to\_remove, strtok(&filename[0], path\_deli));

strcpy(filename, filename + strlen(path\_to\_remove) + 1);

}

strcpy(url->filename, filename);

}

int open\_socket(const char\* ip\_address, const int port){

int socket\_fd;

struct sockaddr\_in server\_addr;

// server address handling

bzero((char\*) &server\_addr, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr(ip\_address); /\*32 bit Internet address network byte ordered\*/

server\_addr.sin\_port = htons(port); /\*server TCP port must be network byte ordered \*/

// open a TCP socket

if ((socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket()\n");

return -1;

}

// connect to the server

if (connect(socket\_fd, (struct sockaddr \*) &server\_addr, sizeof(server\_addr)) < 0) {

perror("connect()\n");

return -1;

}

return socket\_fd;

}

int ftp\_send\_command(int socket\_id, const char\* command, int command\_size, char\* response){

int res = write(socket\_id, command, command\_size);

if( res <= 0){

perror("Error in ftp\_send\_command()\n");

return TRUE;

}

printf("Written %d bytes: %s\n", res, command);

printf("Server response: \n");

if(read\_socket(socket\_id, response)){

perror("Error reading from socket\n");

return TRUE;

};

printf("\n");

return FALSE;

}

int read\_socket(int socket\_id, char\* response){

FILE\* fp = fdopen(socket\_id, "r");

do{

memset(response, 0, MAX\_STRING\_LENGTH);

response = fgets(response, MAX\_STRING\_LENGTH, fp);

printf("%s", response);

} while (!('1' <= response[0] && response[0] <= '5') || response[3] != ' ');

// response[3] == ' ' means received a last status line

//response [0] <= 5 means received numerated status line

printf("\n\n");

return 0;

}

int ftp\_passive\_mode(int socket\_id, char\* command, size\_t command\_size, char\* response){

if(ftp\_send\_command(socket\_id, command, command\_size, response)){

perror("Error in ftp\_send\_command()");

exit(4);

}

char ip\_address[MAX\_STRING\_LENGTH];

int port\_num;

int ip1, ip2, ip3, ip4;

int port1, port2;

//copy values from response to corresponding variables

if (sscanf(response, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)", &ip1, &ip2, &ip3, &ip4, &port1, &port2) < 0) {

perror("sscanf()\n");

return 1;

}

// Creating server ip address

sprintf(ip\_address, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

// Calculating tcp port number

port\_num = port1 \* 256 + port2;

printf("IP: %s\n", ip\_address);

printf("PORT: %d\n", port\_num);

if ((new\_socket\_id = open\_socket(ip\_address, port\_num)) < 0) {

perror("open\_socket()\n");

return 1;

}

return 0;

}

int ftp\_retrieve\_file(int socket\_id, char\* filepath, char\* command, char\* response){

sprintf(command, "RETR %s\r\n", filepath);

if (ftp\_retr(socket\_id, command, strlen(command))){

perror("ftp\_command()\n");

return 1;

}

return 0;

}

int ftp\_retr(int socket\_id, const char\* command, int command\_size){

int res = write(socket\_id, command, command\_size);

if( res <= 0){

perror("Error in ftp\_retr()\n");

return TRUE;

}

printf("Written %d bytes: %s\n", res, command);

printf("\n");

return FALSE;

}

int ftp\_download\_file(int new\_socket\_id, char\* filename){

char buffer[MAX\_STRING\_LENGTH];

int file\_fd;

int res;

if((file\_fd = open(filename, O\_WRONLY | O\_CREAT, 0666)) < 0) {

perror("open()\n");

return 1;

}

while ((res = read(new\_socket\_id, buffer, sizeof(buffer)))) {

if (res < 0) {

perror("read()\n");

return 1;

}

if (write(file\_fd, buffer, res) < 0) {

perror("write()\n");

return 1;

}

}

close(file\_fd);

close(new\_socket\_id);

return 0;

}