

Redes de Computadores

Relatório 2º Trabalho Laboratorial

*Mestrado Integrado em Engenharia Informática e Computação*

3MIEIC01 | Grupo 7

Francisco Gonçalves | up201704790

José Pedro Baptista | up201705255

José Martins | up201605497

# Sumário

O 2º trabalho laboratorial da unidade curricular Redes de Computadores baseia-se no desenvolvimento de uma aplicação a funcionar em modo cliente, e a configuração e teste de redes Ethernet e IP.

# Introdução

Este trabalho teve dois grandes objetivos: o desenvolvimento de uma aplicação de download e a configuração de uma rede.

Neste relatório serão apresentadas todas as escolhas tomadas durante o desenvolvimento destes projetos, sendo este estruturado da seguinte forma:

* **Aplicação de download:** Arquitetura e resultados;
* **Análise e configuração de rede:** Análise das experiências realizadas nas aulas laboratoriais. Cada experiência terá as “perguntas” apresentadas e serão abordados todos os tópicos de forma indireta.
* **Conclusão**: Síntese da informação apresentada nas seções anteriores; reflexão sobre os objectivos de aprendizagem alcançados.

Aplicação de download

# Arquitetura

A aplicação desenvolvida tem como objetivo fazer download de um ficheiro especificado num caminho que engloba o protocolo FTP. Para tal, após compilar, o programa deve ser chamado com 2 argumentos: o nome do executável e um link FTP. O link FTP pode seguir dois formatos:

ftp://<host>/<url\_path>

ftp://[<user>:<password>@]<host>/<url\_path>

Ambas as opções obrigam a que o endereço comece com “ftp://” e que sejam especificados o *host* (domínio) e um caminho para o ficheiro. A segunda opção apresenta um campo opcional entre parêntesis retos onde se especifica o utilizador e password.

O servidor apenas permite o username ‘anonymous’, de maneira que qualquer utilizador que seja introduzido é depois transformado na string ‘anonymous’. Quanto à password, caso não seja especificada inicialmente, ao correr o programa o utilizador encontrará um campo onde pode introduzir uma password qualquer e será sempre aceite. Em caso de má utilização do executável o programa pedirá para tentar novamente especificando os detalhes acima mostrados na consola.

Após ser feita a leitura e sincronização de dados do *link* (parse) o programa tratará de obter o endereço IP do servidor e de criar uma *socket*, conectando-a ao servidor. Este código foi inicialmente fornecido pelos docentes.

Quando esta conexão é bem sucedida o programa passa a funcionar bastante à base de duas funções:

int receive\_msg(ftp\_t\* ftp, char\* msg)

int send\_msg(ftp\_t\* ftp, const char\* msg)

A primeira função responsabiliza-se por receber mensagens do servidor usando ***fdopen*** para abrir o ***fd*** da *socket* do servidor um ciclo while que recebe mensagens usando *fgets* enquanto o **quarto caractere** de cada linha lida é diferente do caractere espaço (ascii 32). A segunda função encarrega-se de enviar uma mensagem para o servidor usando a função *write*,escrevendo para o ***fd*** da *socket* do servidor a mensagem especificada em msg.

Deste modo é possível enviar e receber respostas para as seguintes mensagens:

* Enviar USER e PASS para fazer login
* Enviar CWD para trocar para o diretório do ficheiro a transferir
* Enviar PASV para entrar no modo passivo
* Enviar RETR para obter o obter o ficheiro pretendido (e no fim QUIT)

Após obter endereço IP do *host* e efetuar a conexão ao servidor (***socket fd*** configurado), a função main verifica a password (aceitando qualquer uma caso tenha sido especificada na chamada do executável; caso contrário pede ao utilizador uma password qualquer) e faz login no servidor enviando **USER** <username>, seguido de **PASS** <password>, recebendo as respetivas respostas.

Após efetuado o login, o programa envia **CWD** <filepath> que já está inicializado na variável url (função **split\_path** usada dentro de **url\_parser** - ver anexo 1) e recebe uma resposta de aceitação, trocando então para a diretoria onde está o ficheiro a ser transferido.

Logo a seguir o programa pede ao servidor para entrar em modo passivo enviando **PASV**. Juntamente com este envio é feito o cálculo do *port* final calculado com *Port1\*256 + Port2* (a informação sobre os ports vêm do servidor usando sscanf). Após este cálculo é aberta uma nova socket e conectada ao **data** **fd** da estrutura **ftp**.

De seguida é feito o pedido **RETR** <filename> ao servidor para obter o ficheiro pretendido. Após receber a mensagem de aceitação do pedido RETR, o programa trata de criar e escrever o ficheiro para a pasta atual com a função:

int write\_file(ftp\_t\* ftp, const char\* filename);

Quando o ficheiro acaba de ser “transferido” (término da escrita) o programa desconecta-se do servidor, recebendo uma mensagem de sucesso (226) e envia QUIT.

**typedef** struct URL {

char ip[URL\_STRLEN];

char host[URL\_STRLEN];

char username[URL\_STRLEN];

char password[URL\_STRLEN];

char filepath[URL\_STRLEN];

char filename[URL\_STRLEN];

int port;

} url\_t;

**typedef** struct FTP {

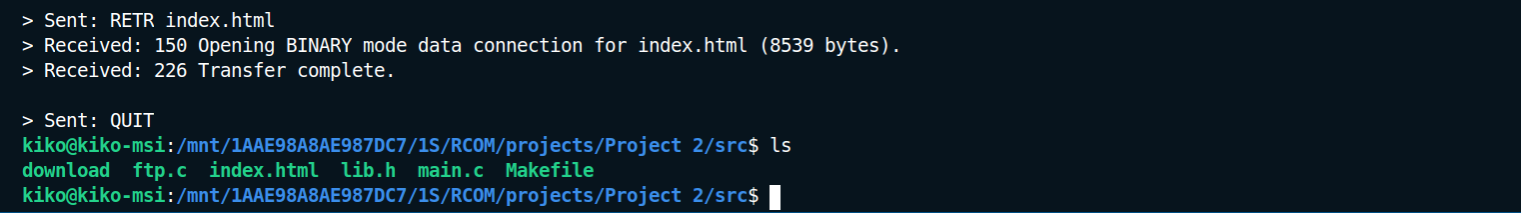
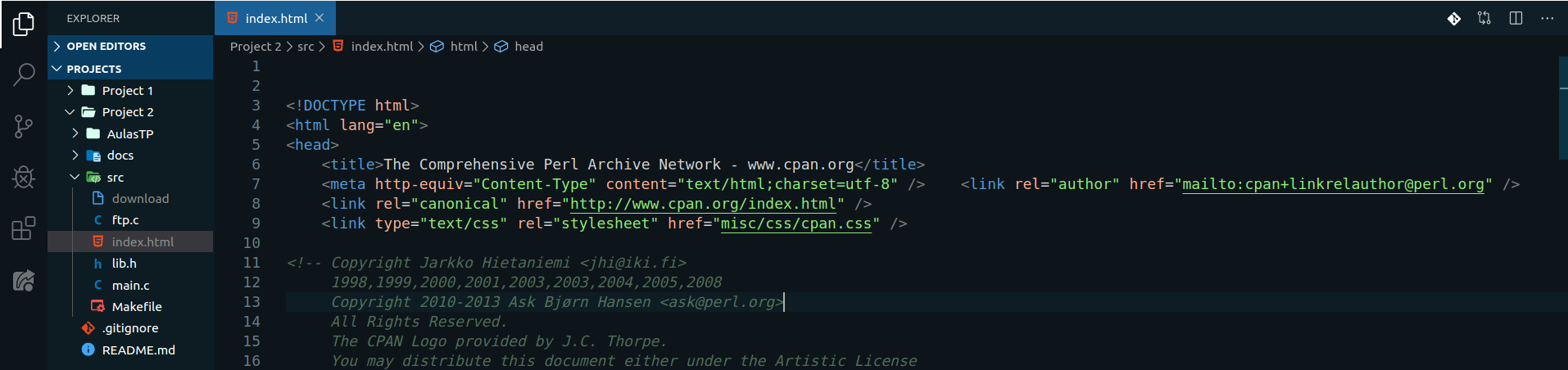
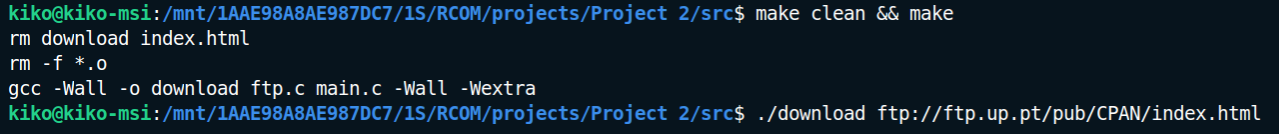
int server\_socket\_fd;

int data\_socket\_fd;

} ftp\_t;

As estruturas acima definidas foram usadas para facilitar o uso do programa. O tipo **url** serve para realizar o parse da informação fornecida no link (segundo argumento da chamada do executável). O tipo **ftp** é utilizado para armazenar o inteiro correspondente aos **fd**s utilizados nas duas sockets abertas durante o programa. Em caso de qualquer erro as funções retornam 1, e em caso de sucesso 0.

Exemplo de download bem sucedido

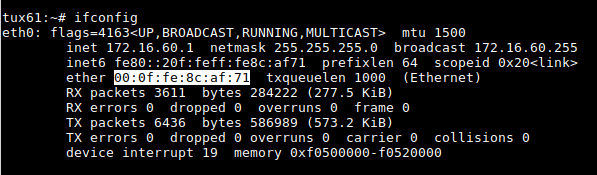
Fazendo uso do ficheiro Makefile que criámos e correndo o programa com o nome do executável e o seguinte link é esperado que seja obtido o ficheiro index.html na pasta src. Em baixo podem observar-se as imagens que ilustram o download bem sucedido. O ficheiro index.html é removido e escrito novamente em /src.

Análise e configuração de rede

# Experiência 1

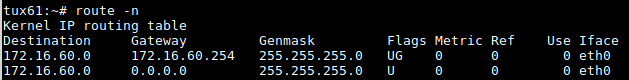
Nesta experiência, foi-nos pedido para configurar as portas eth0 do tux1 e tux4 através do comando **ifconfig,** com os endereços indicados (no caso, 172.16.60.1 e 172.16.60.254, respetivamente). Com a execução (no tux1) do comando **ifconfig eth0 172.16.60.1/24**, foi possivel fazer isso mesmo, como comprova a imagem seguinte:

O mesmo foi feito no tux4 (com o endereço ip correspondente).

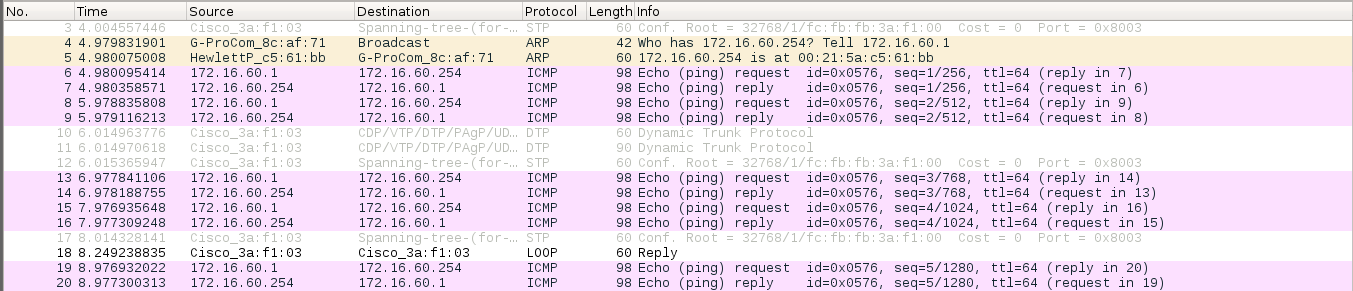
Foi também preciso adicionar rotas no tux1 e no tux4, para permitir a comunicação entre eles através da rede.

**Tux4**: route add -net 172.16.60.0/24 gw 172.16.60.1

**Tux1**: route add -net 172.16.60.0/24 gw 172.16.60.254

Este comando permite adicionar uma rota estática à tabela de rotas. Após a execução dos mesmos, podemos verificar o resultado, executando **route -n**:

Por fim, foi preciso executar o comando **ping** no tux1 para o tux4, com objetivo de verificar que já existia conexão entre os dois computadores. O resultado foi o seguinte:



(dados retirados do programa recomendado - **wireshark**)

Como podemos ver, após a execução do **protocolo de** **ligação ARP**, o tux1 envia dados segundo o protocolo **ICMP** (pedido) e o tux4 recebe e responde para o tux1 segundo o mesmo protocolo.

A experiência foi bem sucedida. Resta agora fazer referência à parte inicial do log: o **protocolo de ligação ARP**.



O protocolo de ligação ARP é usado para obtenção do mac address da máquina de destino através do endereço IPv4 da mesma na rede. Isto é necessário para possibilitar o envio de informação entre máquinas dentro da mesma rede. Consiste em enviar um request do mac address como broadcast e esperar resposta, que será guardada numa ARP Table (localizada na máquina que enviou o request), fazendo com que não seja necessário repetir este processo até que a ARP table (ou entrada em questão) seja apagada. Para verificar o estado da tabela ARP, basta executar o comando **arp -a.**



Neste caso, só tem uma entrada, que corresponde ao tux4:

172.16.60.254 -> Endereço ip da interface eth0 do tux4

00:21:5a:c5:61:bb -> Mac address do tux4

Para apagar as entradas da tabela ARP, deve usar-se o comando **arp –d ipaddress.**

Uma interface loopback é uma interface virtual, contrariamente a uma interface do tipo Fast Ethernet ou Gigabit Ethernet, que são físicas. Este tipo de interfaces pode ser usado para imitar interfaces interfaces físicas e estão sempre disponíveis.

Nesta situação, a interface loopback foi utilizada para permitir que o computador comunicasse com ele próprio.

# Experiência 2

Nesta experiência, foi-nos pedido para criar duas virtual LANs num switch:

* vlany0, possui tux1 e tux4 associados;
* vlany1, apenas possui o tux2 conectado.

Para, p.e. , configurar a vlany0 foi necessário percorrer os seguintes passos:

Em primeiro lugar, na consola do switch criar a vlan com os comandos:

**>> configure terminal**

**>> vlan y0**

**>> end**

De seguida, ainda no switch, associar o tux1 e tux4 à vlany0 da seguinte forma:

**>> configure terminal**

**>> interface fastethernet 0/[identificador da porta no switch]**

**>> switchport mode access**

**>> switchport access vlan y0**

**>> end**

Após configurar as vlans foi possível verificar através do comando **ping -b** e através dos logs obtidos que existem 2 broadcast domains. Isto verifica-se pois ao efetuar ping broadcast no tux1 apenas se obtém uma resposta por parte do tux4. Por outro lado, efetuando ping broadcast no tux2 não se obtém resposta de nenhum tux.

Estes resultados são os esperados visto os tux1 e tux4 se encontrarem numa sub-rede diferente da do tux2.

# 

Ping Broadcast no tux1 é respondido pelo tux4



Ping Broadcast no tux2 não obtém resposta

# Experiência 3

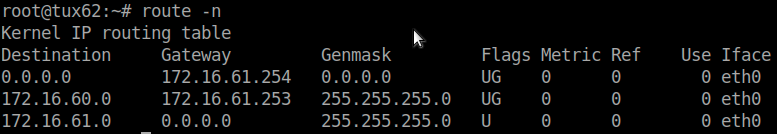
O objetivo desta experiência era configurar um router em Linux, mais precisamente transformar o tux4 num router que estabeleça a ligação entre as duas vlans criadas na experiência 2.

É necessário assim ativar a interface eth1 no tux4 da mesma forma que se ativou a eth0 na experiência 1 e associar o tux à vlan y1 tal como na experiência 2.

Após estes comandos, resta adicionar as rotas aos tux1 e tux2 necessárias para que estes possam comunicar entre si. Estas rotas são adicionadas com os comandos **:**

* **route add -net 172.16.y1.0/24 gw 172.16.y0.254,** no tux1, permite que este acceda a rede 172.16.y1.0 pela gateway 172.16.y1.254 no tux4;
* **route add -net 172.16.y0.0/24 gw 172.16.y1.253,** no tux2, permite que este acceda a rede 172.16.y0.0 pela gateway 172.16.y1.253 no tux4.

Através do comando **route -n** é possível verificar a informação da *forwarding table* de cada um dos tuxs:



Forwarding table do tux2.

Verifica-se assim na figura X que o tux2 possui agora uma rota para a rede 172.16.y0.0 usando como gateway o IP 172.16.y1.253 do tux4.

Finalmente, após estas configurações é assim possível enviar um *ping* entre o tux1 e o tux2.



Ping efetuado pelo tux1 é respondido pelo tux2.

Experiência 4

Nesta experiência, fomos encarregues de configurar um router comercial, implementar um nat no mesmo e depois disso fazer alguns testes.

Para fazer a configuração do router, foi necessário configurar as interfaces gigabitethernet 0/0 e gigabitethernet 0/1 do seguinte modo:

(no terminal de configuração do router)

**interface gigabitethernet 0/1**

**ip address 172.16.1.69 255.255.255.0**

**no shutdown**

**interface gigabitethernet 0/0**

**ip address 172.16.61.254 255.255.255.0**

**no shutdown**

Após inserir estes comandos foi necessário fazer o seguinte:

(no terminal de configuração do switch)

**configure terminal**

**interface fastethernet 0/12**

**switchport mode access**

**switchport access vlan 61**

Os comandos acima servem para adicionar a porta 12 do switch à vlan 61 (onde também se encontra a interface eth0 do tux2 e a interface eth1 do tux4). Esta porta é onde se encontra o cabo que está ligado ao router comercial (interface gigabitethernet 0/0).

Uma vez adicionada a interface 0/1 do router comercial à vlan 61, foi preciso adicionar default routes em todos os tux:

**Tux 61:**

**route add default gw 172.16.60.254**

172.16.60.254 -> IP da interface eth0 do tux4

**Tux 62:**

**route add default gw 172.16.61.254**

172.16.61.254 -> IP da interface do router comercial 0/0

**Tux 64:**

**route add default gw 172.16.61.254**

172.16.61.254 -> IP da interface do router comercial 0/0

Por fim, foi também necessário adicionar um route no router comercial, para a rede da vlan 60, onde se encontra o tux1, para que o router pudesse dar resposta a qualquer pedido que o tux1 lhe fizesse. Procedemos da seguinte forma:

(no terminal de configuração do router)

**ip route 172.16.60.0 255.255.255.0 172.16.61.253**

Nesta fase, o tux1 já era capaz de executar com sucesso o comando ping em todas as interfaces do tux4, tux2 e Rc (como dizia o enunciado).

Agora foi-nos pedido que desligássemos os redirecionamentos no tux2 e apagássemos o caminho que o tux2 tinha até à rede onde se encontrava o tux1 (172.16.60.0):

**echo 0 > /proc/sys/net/ipv4/conf/eth0/accept\_redirects**

**echo 0 > /proc/sys/net/ipv4/conf/all/accept\_redirects**

**route del -net 172.16.60.0/24**

Após isto, executamos o comando **traceroute 172.16.60.1** e verificamos a seguinte ordem de acontecimentos:

Como o tux2 não tem caminho até ao tux1, transmite os packets para a default route (Router Comercial - 172.16.61.254).

Como o router comercial já tem caminho para tux1, envia os packets para tux1.

Por fim, o tux1 recebe os packets e responde ao tux2 (diretamente).

Resta apenas fazer referência à implementação do nat.

Um NAT (Network Address Translation) é um processo onde uma máquina da rede associa um endereço público a outra (ou outras) que se encontra(m) dentro de uma rede privada. O NAT é usado principalmente para limitar o número de endereços IP públicos, por questões económicas e de segurança. Implementamos esta funcionalidade conforme sugere o guia:

(no terminal de configuração do router)

**interface gigabitethernet 0/0**

**ip address 172.16.61.254 255.255.255.0**

**no shutdown**

**ip nat inside**

(interface interna do NAT)

**interface gigabitethernet 0/1**

**ip address 172.16.1.69 255.255.255.0**

**no shutdown**

**ip nat outside**

(interface externa do NAT)

**ip nat pool ovrld 172.16.1.69 172.16.1.69 prefix 24**

**ip nat inside source list 1 pool ovrld overload**

**access-list 1 permit 172.16.60.0 0.0.0.7**

**access-list 1 permit 172.16.61.0 0.0.0.7**

Por fim, foi necessário adicionar um route default para no router comercial para o router da sala do seguinte modo:

(no terminal de configuração do router)

**ip route 0.0.0.0 0.0.0.0 172.16.1.254**

Mesmo tendo adicionado um caminho para o router do lab, ainda não é possível, nesta fase, o acesso a redes externas e, por isso, nem à internet. Para isso é foi necessária a experiência 5.

Experiência **5**

Nesta experiência, apenas tínhamos de configurar o serviço DNS em todos os tux. A principal função deste serviço é a tradução de nomes de domínios para os seus endereços IP respetivos. Para isso executamos os seguintes comandos na consola de cada um:

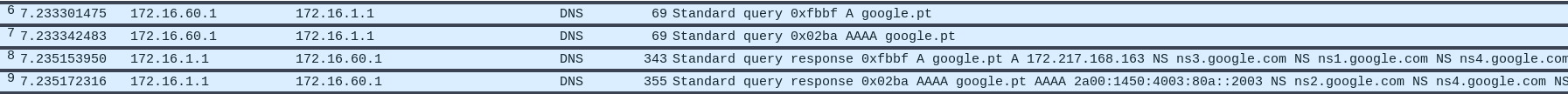
# **echo -e "search netlab.fe.up.pt\nnameserver 172.16.1.1" > /etc/resolv.conf**

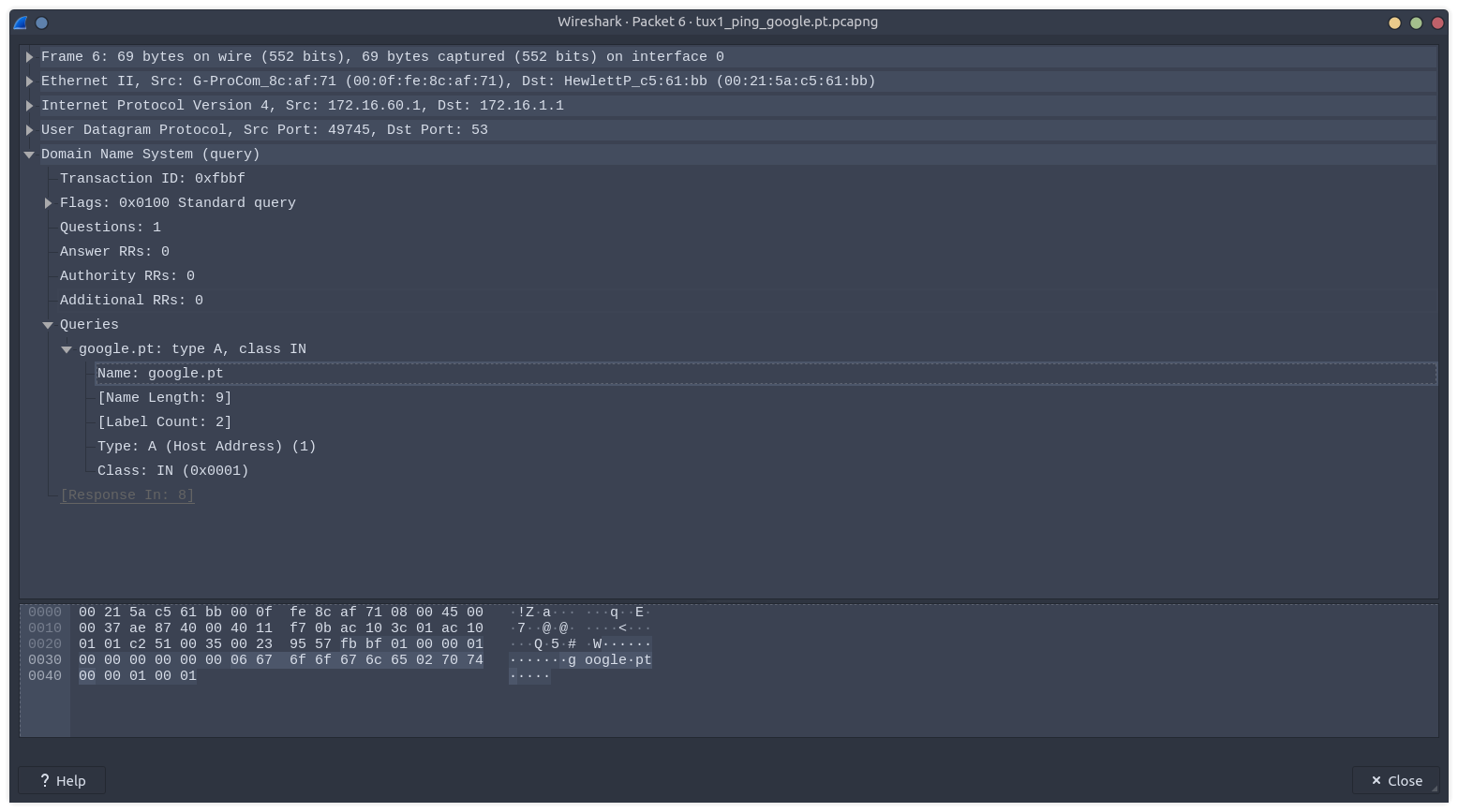
Que é equivalente a escrever:

# **search netlab.fe.up.pt**

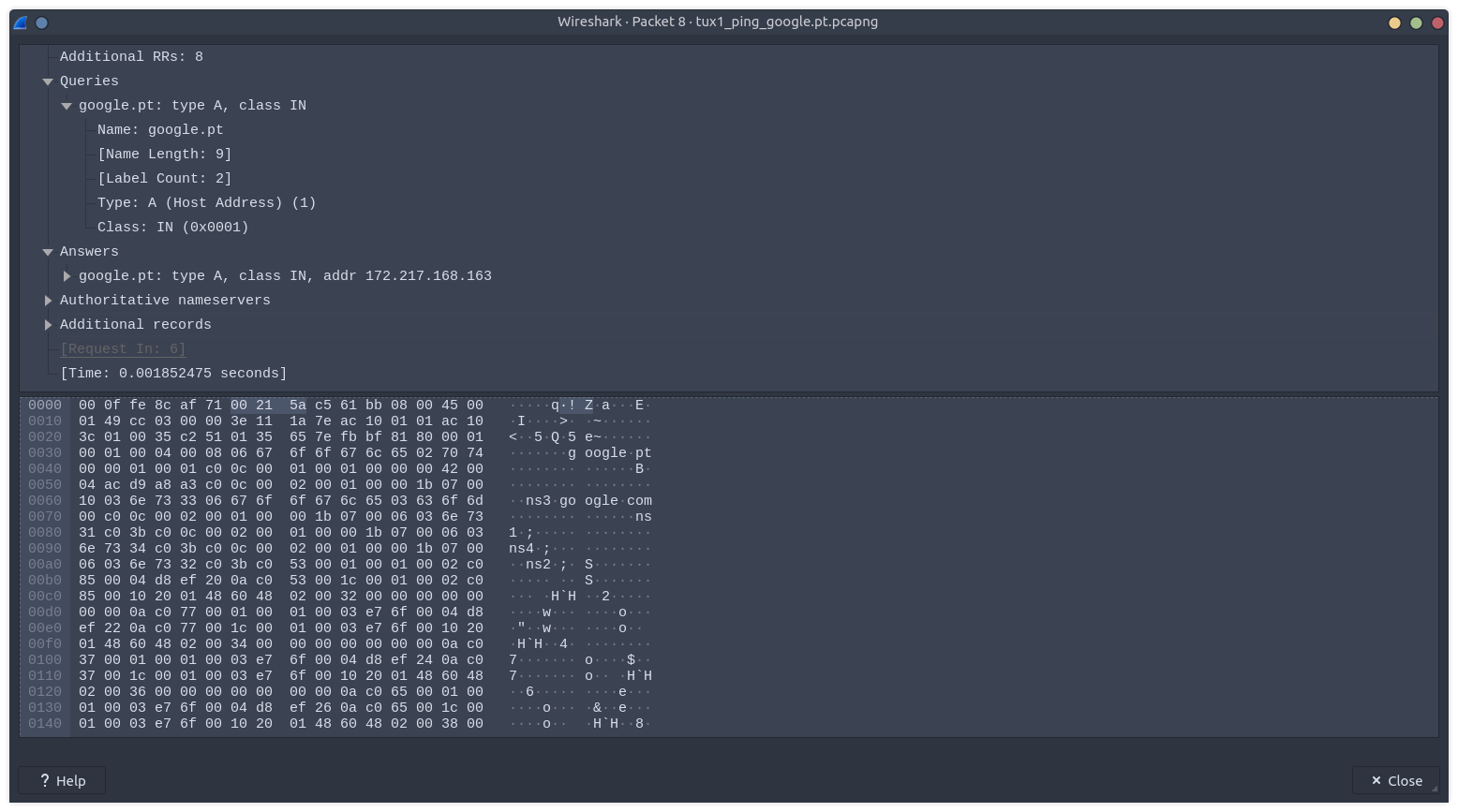
# **nameserver 172.16.1.1**

No ficheiro resolv.conf (como descrevia o guião). Depois de terminada esta configuração já foi possível o acesso à internet, quer através do browser, quer através da execução do comando ping:





Pedido de tradução do nome do domínio inserido para o seu endereço IP



Resposta com o respetivo endereço IP

# 

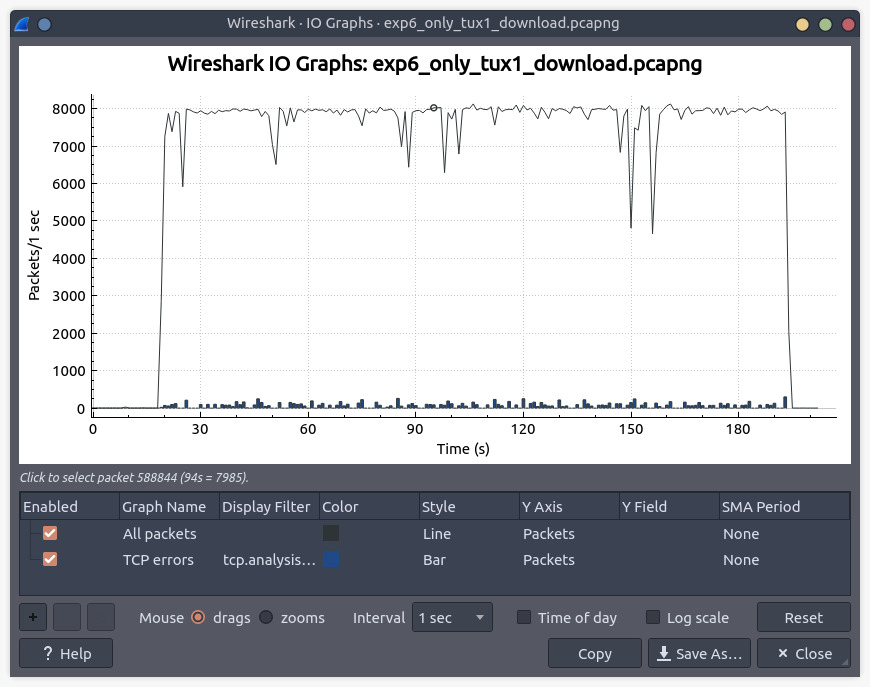
# 

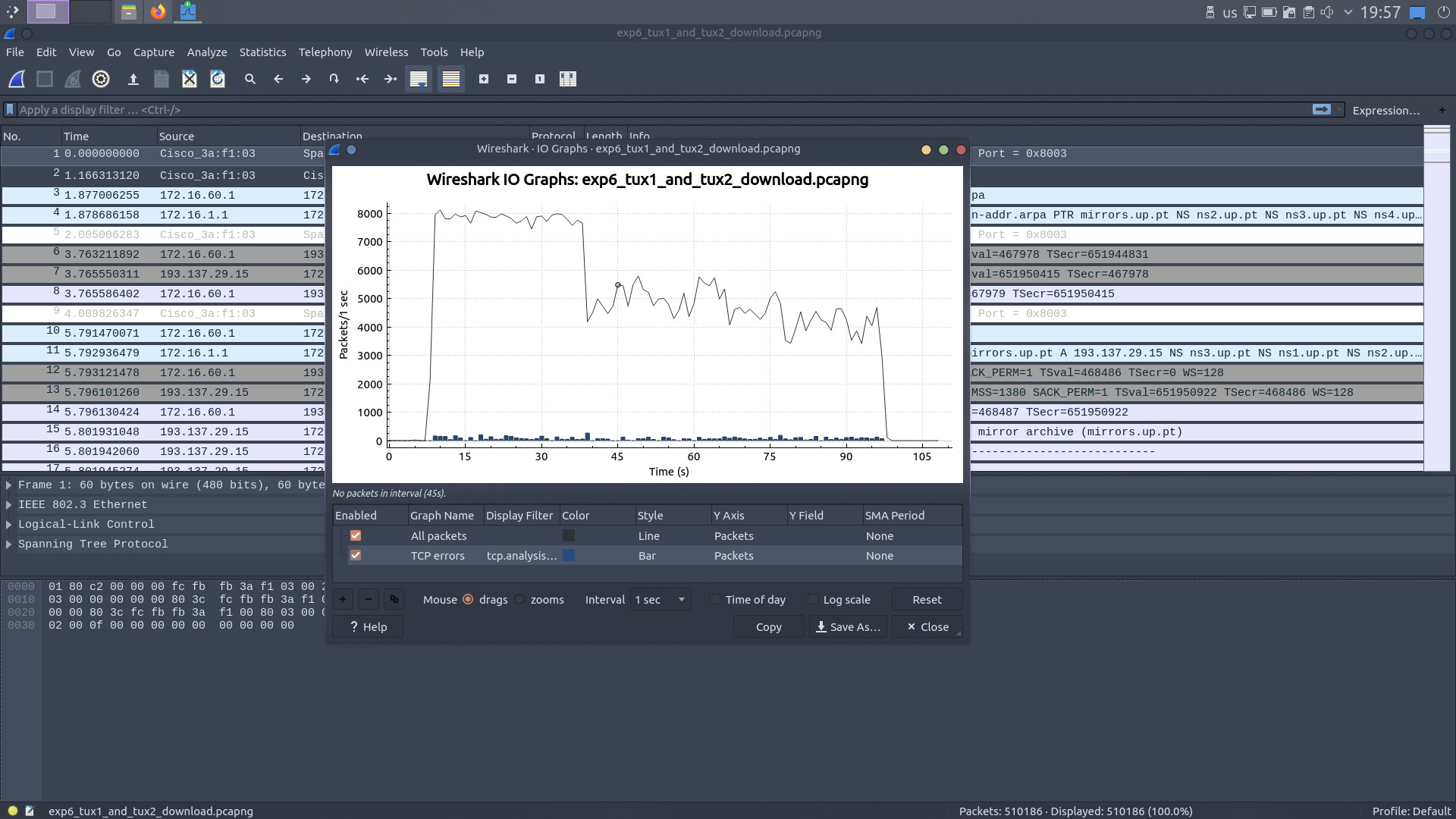
# Experiência 6

A aplicação ftp que desenvolvemos abre duas conexões TCP: uma para enviar comandos do protocolo FTP para o servidor (recebendo a(s) respetiva(s) resposta(s)) e outra para a receção de dados do servidor (e respetiva resposta do cliente). A primeira conexão TCP mencionada é, então, responsável pelo transporte de informação de controlo FTP (troca de comandos).

A aplicação que desenvolvemos ilustra bem as fases de uma conexão TCP. A primeira corresponde ao estabelecimento da ligação. Na segunda fase dá-se a troca de dados e, na terceira e última fase, é feita a desconexão.

O *Transmission Control Protocol* (**TCP**) faz uso do *Automatic Repeat Request* (**ARQ**) através do método da janela deslizante, que consiste no controlo de erros nas transmissões de dados. Esse controlo de erros é feito através do uso de *acknowledgement numbers*, que são enviados pelo recetor, indicando se a trama de dados foi recebida com sucesso. Para além disso o controlo de erros faz uso ainda da *window size*, que indica a gama de pacotes que o emissor vai enviar e do *sequence number*, que representa o índice do pacote a ser enviado.

Para realizar o controlo da congestão, o **TCP** usa uma janela de congestão, que consiste numa estimativa do nº de bytes que a rede consegue enviar, não enviando mais do que o nº mínimo de octetos da janela definida pelo recetor e da janela de congestão.

Olhando para o gráfico da esquerda, que representa o download para o tux1 apenas, podemos observar que o bitrate (packets/s) é bastante consistente, já que depois de atingir um valor máximo, não volta a diminuir muito, exceto em casos esporádicos (150s, por exemplo). No início do segundo download verificamos um bitrate consistente semelhante ao do download do tux1. No entanto, por volta dos 40s há queda que faz o bitrate não voltar ao nível inicial durante o resto do download. Isto deve-se à inicialização dum novo download a partir do tux2, o que bate certo com o mecanismo de controlo de congestão, já que há uma divisão do bitrate para cada ligação, a partir do momento em que foi inicializado o segundo download. Quando surge uma segunda conexão TCP, a transferência de dados pode ser afetada, passando por uma queda na taxa de transmissão, já que, se estiverem duas transferências ativas em simultâneo, esta taxa é distribuída equitativamente para as ligações existentes.

Conclusão

Tanto a aplicação de download como a configuração da rede foram desenvolvidas com sucesso, sendo que todos os testes efetuados pelo professor responsável durante a avaliação presencial evidenciaram os resultados pretendidos.

O grupo considera que todos os objetivos de aprendizagem foram alcançados e bem consolidados ficando assim com um bom conhecimento acerca de comunicações multi-acesso, encaminhamento, controlo de fluxo e controlo de congestionamento.

Referências

* https://whatismyipaddress.com/nat
* http://www.omnisecu.com/cisco-certified-network-associate-ccna/what-is-loopback-interface-in-a-router.php
* https://www.youtube.com/watch?v=Rck3BALhI5c
* https://www.techopedia.com/definition/28503/dns-server
* https://www.melvinswebstuff.com/blog/2009/04/24/calculate-ftp-data-port/

Anexos

Anexo I - Código da aplicação de download ftp

lib.h

**#pragma** once

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#include** <strings.h>

**#include** <stdbool.h>

**#include** <errno.h>

**#include** <netdb.h>

**#include** <unistd.h>

**#include** <arpa/inet.h>

**#include** <sys/types.h>

**#include** <sys/socket.h>

**#include** <netinet/in.h>

*// ------------- MACROS -------------*

**#define** MAX\_SIZE **1024**

**#define** URL\_STRLEN **256**

**#define** PASV **"PASV\n"**

**#define** QUIT **"QUIT\n"**

*// ------------- TYPES --------------*

**typedef** struct URL {

char ip[URL\_STRLEN];

char host[URL\_STRLEN];

char username[URL\_STRLEN];

char password[URL\_STRLEN];

char filepath[URL\_STRLEN];

char filename[URL\_STRLEN];

int port;

} url\_t;

*// ----------------------------------*

**typedef** struct FTP {

int server\_socket\_fd;

int data\_socket\_fd;

} ftp\_t;

*// ------- Function declarations -----*

*/\*\**

*\** @brief *tells the user how to run the executable*

*\** @paramn *number of arguments*

*\** @params *string provided in executable called*

void print\_usage(char\* s);

*/\*\**

*\** @brief *check executable call usage*

*\** @paramn *number of arguments*

*\** @paramarray *of strings in executable call*

*\** @return *0 upon success, 1 otherwise\*/*

int check\_usage(int n, char\*\* s);

*/\*\**

*\** @brief *parse information given in executable call into url\_t var*

*\** @paramlink *string array in executable call*

*\** @return *0 upon success, 1 otherwise\*/*

int url\_parser(url\_t\* url, char\* link);

*/\*\**

*\** @brief *splits a whole path into path to file and filename*

*\** @parampath *initially path (ends up being just path to file)*

*\** @paramfile *initially path (ends up being just the filename)*

*\** @return *0 upon success\*/*

int split\_path(char\* path, char\* file);

*/\*\**

*\** @brief *check valid password (any is valid)*

*\** @parampass *password (value depends on executable call)*

*\** @return *final password (char\* element)\*/*

char\* check\_password(char\* pass);

*/\*\**

*\** @brief *send ftp message to server*

*\** @paramftp *ftp struct element containing the File Descriptors*

*\** @parammsg *message received*

*\** @return *0 upon success, 1 otherwise\*/*

int send\_msg(ftp\_t\* ftp, const char\* msg);

*/\*\**

*\** @brief *receive ftp message from server*

*\** @paramftp *ftp struct element containing the File Descriptors*

*\** @parammsg *message sent*

*\** @return *0 upon success\*/*

int receive\_msg(ftp\_t\* ftp, char\* msg);

*/\*\**

*\** @brief *connect socket to server*

*\** @paramport *server address port*

*\** @paramip *IP address of the server*

*\** @return *integer fd (file descriptor)\*/*

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

*/\*\**

*\** @brief *connects to server (220)*

*\** @paramftp *ftp struct element containing the File Descriptors*

*\** @return *0 upon success\*/*

int server\_connect(int port, const char\* ip, ftp\_t\* ftp);

*/\*\**

*\** @brief *disconnects from server - asks for final message (226) and sends QUIT*

*\** @paramftp *ftp struct containing the File Descriptors*

*\** @return *0 upon success\*/*

int disconnect(ftp\_t\* ftp);

*/\*\**

*\** @brief *login using USER and PASS requests with the information previously parsed*

*\** @paramftp *ftp struct element containing the File Descriptors*

*\** @paramuser *login username string*

*\** @parampassword *login password*

*\** @return *0 upon success\*/*

int login(ftp\_t\* ftp, const char\* user, const char\* password);

*/\*\**

*\** @brief *sends CWD request-tells server the pathname prefix of the file to download*

*\** @paramftp *struct containing the File Descriptors*

*\** @parampath *path to the directory of the file*

*\** @return *0 upon success\*/*

int ftp\_cwd(ftp\_t\* ftp, const char\* path) ;

*/\*\**

*\** @brief *sends PASV request and asks server to accept data connection*

*\** @paramftp *struct containing the File Descriptors*

*\** @return *0 upon success*

*\*/*

int ftp\_pasv(ftp\_t\* ftp);

*/\*\**

*\** @brief *sends RETR request and asks server for the contents of the file specified*

*\** @paramftp *struct element containing the File Descriptors*

*\** @paramfilename *name of the file to be transferred*

*\** @return *0 upon success\*/*

int ftp\_retr(ftp\_t\* ftp, const char\* filename);

*/\*\**

*\** @brief *writes the intended file specified in filename*

*\** @paramftp *ftp struct containing the File Descriptors*

*\** @paramfilename *name of file to be downloaded*

*\** @return *0 upon success\*/*

int write\_file(ftp\_t\* ftp, const char\* filename);

**ftp.c**

**#include "lib.h"**

int check\_usage(int argc, char\*\* argv) {

if(argc != 2) {

printf("\nWrong executable call (number of arguments must be 2)\n");

print\_usage(argv[0]); return 1;

}

else if(argc == 2 && (strncmp("ftp://", argv[1], 6) != 0)) {

print\_usage(argv[0]); return 1;

}

return 0;

}

*//---------------------------------------------------------------*

void print\_usage(char\* s) {

printf("\nThe section in [ ] is optional\n");

printf("ANONYMOUS \t%s ftp://<host>/<url\_path>\n", s);

printf("AUTHENTICATION\t%s ftp://[<user>:<password>@]<host>/<url\_path>\n\n\n",s);

}

*//---------------------------------------------------------------*

int url\_parser(url\_t\* url, char\* link) {

url->port = 21;

if (sscanf(link, "ftp://%[^:]%\*[:]%[^@]%\*[@]%[^/]%\*[/]%s",

url->username, url->password, url->host, url->filepath) == 4) {

split\_path(url->filepath, url->filename);

printf("\n========== Parsed url and authentication ==========\n\n");

printf("User: %s\n", url->username);

printf("Pass: %s\n", url->password);

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

printf("Path: %s\n", url->filepath);

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

memset(url->username, 0, sizeof(char) \* URL\_STRLEN);

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

}

else if (sscanf(link, "ftp://%[^/]%\*[/]%s", url->host, url->filepath) == 2) {

split\_path(url->filepath, url->filename);

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

strcpy(url->password, ""); *//will be altered later*

printf("\n=== Parsed url with no authentication (anonymous) ===\n\n");

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

printf("Path: %s\n", url->filepath);

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

}

else {

printf("Error while trying to parse url\n");

print\_usage(link);

return 1;

}

return 0;

}

*// -----------------------------------------------------------------*

int split\_path(char\* path, char\* file) {

strcpy(file, path);

size\_t len = strlen(path);

int pos = len-1;

const char\* aux = path;

char final[len];

for(size\_t i=0; i<len; i++) {

if(path[pos] == '/') break;

pos--;

}

len = len-pos; *//length of filename*

pos++; *//now pos is first char of filename*

for(size\_t i=0; i<len; i++) file[i] = file[pos+i];

strncpy(final, aux, pos-1); final[pos-1] = 0;

strcpy(path, final);

return 0;

}

*// -----------------------------------------------------------------*

char\* check\_password(char\* pass) {

char\* pw;

if(strlen(pass)) pw = pass; *//if it was specified in exe call no changes*

else {

char aux[20];

printf("Using server in ANONYMOUS mode\nType in any password: ");

while (strlen(fgets(aux, 20, stdin)) < 1)

printf("\nInvalid input. Try again: ");

pw = (char \*)malloc(strlen(aux)); strcpy(pw, aux);

}

return pw; *//final password*

}

*// -----------------------------------------------------------------*

int send\_msg(ftp\_t\* ftp, const char\* msg) {

if ((write(ftp->server\_socket\_fd, msg, strlen(msg))) <= 0) {

printf("Couldn\'t write (send\_msg)\n");

return 1;

}

printf("\n> Sent: %s", msg);

return 0;

}

*// -----------------------------------------------------------------*

int receive\_msg(ftp\_t\* ftp, char\* msg) {

FILE\* f = fdopen(ftp->server\_socket\_fd, "r");

memset(msg, 0, MAX\_SIZE);

while(msg[3] != ' ') {

memset(msg, 0, MAX\_SIZE);

msg = fgets(msg, MAX\_SIZE, f);

printf("> Received: %s", msg);

}

return 0;

}

*// -----------------------------------------------------------------*

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

int fd;

struct sockaddr\_in server\_addr;

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

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

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

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

*// open an TCP socket*

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

perror("socket()");

return 1;

}

*// connect to the server*

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

perror("connect()");

return 1;

}

return fd;

}

*// -----------------------------------------------------------------*

int server\_connect(int port, const char\* ip, ftp\_t\* ftp) {

char read\_buf[MAX\_SIZE];

if ((ftp->server\_socket\_fd = connect\_socket(port, ip)) < 0) {

printf("Couldn\'t connect socket\n"); return 1;

}

if (receive\_msg(ftp, read\_buf)) {

printf("Couldn\'t read info\n"); return 1;

}

return 0;

}

*// -----------------------------------------------------------------*

int login(ftp\_t\* ftp, const char\* user, const char\* password) {

char login[MAX\_SIZE];

sprintf(login, "USER %s\n", user); *// username into login string*

if(send\_msg(ftp, login)) {

printf("Failed to send message with username (USER)\n"); return 1;

}

if(receive\_msg(ftp, login)) {

printf("Failed to receive reply to username sent\n"); return 1;

}

memset(login, 0, sizeof(login)); *// cleaning login string*

sprintf(login, "PASS %s\n", password); *// password into login string*

if(send\_msg(ftp, login)) {

printf("Failed to send message with password (PASS)\n"); return 1;

}

if(receive\_msg(ftp, login)) {

printf("Failed to receive reply to password sent\n"); return 1;

}

return 0;

}

*// -----------------------------------------------------------------*

int ftp\_cwd(ftp\_t\* ftp, const char\* path) {

char cwd[MAX\_SIZE];

sprintf(cwd, "CWD %s\n", path);

if (send\_msg(ftp, cwd)) {

printf("Failed to send change directory message (CWD)\n");

return 1;

}

if (receive\_msg(ftp, cwd)) {

printf("Couldn\'t receive acceptance directory messages (CWD)\n");

return 1;

}

return 0;

}

*// -----------------------------------------------------------------*

int ftp\_pasv(ftp\_t\* ftp) {

char pasv[MAX\_SIZE] = PASV;

if (send\_msg(ftp, pasv)) {

printf("Failed to send passive mode msg\n");

return 1;

}

if (receive\_msg(ftp, pasv)) {

printf("Couldn\'t enter passive mode\n");

return 1;

}

int IP[4], P1, P2; *//IP array, and 2 auxiliary ports*

*//P1,P2 is the port the server is telling the client to use during the data transfer*

if ((sscanf(pasv, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",

&IP[0], &IP[1], &IP[2], &IP[3], &P1, &P2)) < 0) {

printf("Couldn\'t parse information to calculate port correctly\n");

return 1;

}

memset(pasv, 0, sizeof(pasv)); *//cleaning pasv string ---> becomes IP string*

if ((sprintf(pasv, "%d.%d.%d.%d", IP[0], IP[1], IP[2], IP[3])) < 0) {

printf("Couldn\'t generate IP address\n");

return 1;

}

int port = (P1 \* 256) + P2; *// calculating final ftp data port*

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

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

*//if fd has invalid value abort*

if ((ftp->data\_socket\_fd = connect\_socket(port, pasv)) < 0) return 1;

return 0;

}

*// -----------------------------------------------------------------*

int ftp\_retr(ftp\_t\* ftp, const char\* filename) {

char retr[MAX\_SIZE];

sprintf(retr, "RETR %s\n", filename);

if(send\_msg(ftp, retr)){ printf("Couldn\'t send RETR message\n"); return 1; }

if(receive\_msg(ftp, retr)) { printf("No information received\n"); return 1; }

return 0;

}

*// -----------------------------------------------------------------*

int write\_file(ftp\_t\* ftp, const char\* filename)

{

FILE\* F;

int bytes;

char buf[MAX\_SIZE];

if (!(F = fopen(filename, "w"))) {

printf("Couldn\'t open file with filename: %s\n", filename);

return 1;

}

while( (bytes = read(ftp->data\_socket\_fd, buf, sizeof(buf))) ) {

if(bytes<0){ printf("Nothing read from data socket fd\n");return 1; }

if((bytes = fwrite(buf, bytes, 1, F)) < 0) {

printf("Failed to write data to file\n"); return 1; }

}

fclose(F);

close(ftp->data\_socket\_fd);

return 0;

}

*// -----------------------------------------------------------------*

int disconnect(ftp\_t\* ftp)

{

char disconnect[MAX\_SIZE];

if (receive\_msg(ftp, disconnect)) {

printf("Couldn\'t receive disconnect message\n");

return 1;

}

sprintf(disconnect, QUIT);

if (send\_msg(ftp, disconnect)) {

printf("Couldn\'t send QUIT message\n");

return 1;

}

if(ftp->server\_socket\_fd) close(ftp->server\_socket\_fd);

return 0;

}

**main.c**

**#include "lib.h"**

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

printf("\n");

printf("-------------------------------------------------------\n");

printf("------------ This server is ANONYMOUS only ------------\n");

printf("--- Any USER introduced will be read as 'anonymous' ---\n");

printf("-------------------------------------------------------\n");

if(check\_usage(argc, argv)) return 1; *//check executable call*

url\_t url; ftp\_t ftp; struct hostent\* h; *//variables*

if(url\_parser(&url, argv[1])) return 1; *//parse info given in argv[1]*

*//------------------- get IP with host --------------------*

printf("\n=== Getting IP with host ===\n");

if ((h = gethostbyname(url.host)) == **NULL**) {

herror("gethostbyname");

return 1;

}

char\* ip = inet\_ntoa(\*((struct in\_addr \*) h->h\_addr\_list[0]));

strcpy(url.ip, ip);

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

*//---------------------------------------------------------*

if(server\_connect(url.port, url.ip, &ftp)) return 1; *//connect to the server*

strcpy(url.password, check\_password(url.password));

if(login(&ftp, url.username, url.password)) {

printf("Couldn\'t login as user %s\n", url.username);

return 1;

}

*//---------------------------------------------------------*

if (ftp\_cwd(&ftp, url.filepath)) {

printf("Couldn\'t change to the directory of the file\n");

return 1;

}

*//---------------------------------------------------------*

if (ftp\_pasv(&ftp)) {

printf("Couldn\'t enter passive mode\n");

return 1;

}

*//---------------------------------------------------------*

ftp\_retr(&ftp, url.filename); *//get file*

write\_file(&ftp, url.filename); *//download file*

disconnect(&ftp); *//exit*

return 0;

}