2º Trabalho Laboratorial

Relatório



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Beatriz de Henriques Martins – up201502858

Nádia de Sousa Varela Carvalho – up201208223

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

22 de dezembro de 2017

Índice

[1 Sumário 3](#_Toc501443960)

[2 Introdução 3](#_Toc501443961)

[3 Aplicação Download 3](#_Toc501443962)

[3.1 Arquitetura 3](#_Toc501443963)

[3.2 Resultados 3](#_Toc501443964)

[4 Configuração de uma rede 3](#_Toc501443965)

[4.1 Configurar um IP de Rede 3](#_Toc501443966)

[4.2 Implementar duas LAN´s virtuais no switch 3](#_Toc501443967)

[4.3 Configurar um router em Linux 3](#_Toc501443968)

[4.4 Configurar um router comercial e implementar NAT 4](#_Toc501443969)

[4.5 DNS 4](#_Toc501443970)

[4.6 Ligações TCP 4](#_Toc501443971)

[5 Conclusão 4](#_Toc501443972)

[Anexo – Código fonte 5](#_Toc501443973)

[clientTCP.c 5](#_Toc501443974)

[parser.h 6](#_Toc501443975)

[parser.c 7](#_Toc501443976)

[tcp.h 9](#_Toc501443977)

[tcp.c 10](#_Toc501443978)

# Sumário

Este relatório tem o objetivo de cimentar os conhecimentos adquiridos e de apoio ao projeto desenvolvido, configuração de uma rede, utilizando os comandos de configuração do *router* e *switch,* e desenvolvimento de uma aplicação capaz de fazer o download de um ficheiro através de um cliente FTP.

O projeto foi concluído com sucesso. Todos os objetivos propostos foram cumpridos.

# Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Redes de Computadores do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto. Os objetivos principais do projeto eram o desenvolvimento de uma aplicação, cliente FTP, capaz de fazer o download de um ficheiro utilizando o *File Transfer Protocol* e a configuração de uma rede.

Este relatório está divido em 2 grandes parte. A primeira é a implementação de um cliente FTP, a segunda a configuração e analise de uma rede. Inicialmente, será explicada a implementação e estrutura da aplicação FTP, e de seguida iremos abordar as experiências realizadas nas aulas teórico-práticas e uma breve explicação do objetivo de cada uma delas.

# Aplicação Download

## Arquitetura

A aplicação foi desenvolvida utilizando uma estratégia de divisão em duas partes, onde na primeira é realizado o processamento da *string* de input do utilizador e na segunda é efetuada a ligação com o servidor, e as instruções guardadas no passo anterior: login e download do ficheiro indicado.

Para realizar o processamento da *string* de input do utilizador, utilizamos uma estrutura à qual denominamos *urlInfo* que armazena o username e password do utilizador, o informação e url do host, o path para o mesmo e o nome do ficheiro para fazer download.

Código da estrutura

Após o processamento e validação da *string* de input do utilizador, a informação é guardada na estrutura *urlInfo*, acontecendo tudo isto dentro da função parseURL. Esta função, quando processa o nome do utilizador, chama a função *initUser* ao receber um utilizador especifico ou a função *initDefaultUser*, que inicia o utilizador com nome anonymous e password mail@domain (definidos inicialmente como constantes).

Definição da função parseURL

Definição da função *initUser*

Definição da função *initDefaultUser*

Após a estrutura estar instanciada e ter sido feito o processamento da *string* de input do utilizador, é chamada a função *createConnection* que realiza a ligação do cliente FTP ao servidor através de um socket. Nós mantivemos a estrutura do código que nos tinha sido facultado no ficheiro *clientTCP.c* no moodle. Nesta altura a ligação criada é uma ligação de controlo.

Definição da função *createConnection*

Em seguida é feito o login do utilizador no servidor, utilizando as credenciais de utilizador facultadas e guardadas na estrutura *urlInfo* servindo-se da função de *login*, que por sua vez usa as funções de escrita e leitura do socket, *readSocket* e *writeSocket,* definidas no ficheiro tcp.h.

Definição da função *login*

Definição da função *readSocket*

Definição da função *writeSocket*

Depois, é colocado o servidor em modo passivo, permitindo ao cliente iniciar ambas as conexões ao servidor, o que resolve o problema de a firewall do computador poder filtrar os dados recebidos pelo cliente. O servidor responde, enviando uma string com o endereço ip e a porta, por onde irá ocorrer o download. Em seguida volta a ser feita uma chamada à função *createConnection* descrita acima, desta vez para criar uma ligação de *data.* Com esta ligação pedimos ao servidor o ficheiro para fazer o download, recorrendo à função *sendAndRetrieve.* Esta envia o comando ‘RETR’ seguido do *path* para o ficheiro e nome do mesmo que tinham sido guardados na estrutura *urlInfo*. Caso a resposta do servidor verifique a existência do ficheiro, é feita uma chamada à função *downloadFile* que faz a leitura do ficheiro pedido e o guarda. Caso contrário, o programa termina com uma mensagem de erro na pesquisa do ficheiro. Após terminar o download, os sockets usados são fechados e é libertada a memória alocada, o acontece durante a chamada à função *closeConnection*.

Definição da função *passiveMode*

Definição da função *sendAndRetrieve*

Definição da função *downloadFile*

Definição da função *closeConnection*

## Resultados

A aplicação foi testada em diversos ficheiros, com o servidor ftp da Universidade da FEUP e outros desconhecidos, em modo anónimo e de utilizador conhecido e foi sempre capaz de fazer download do ficheiro, quer ficheiros de texto de 78 bytes até a um ficheiro MP4 de . Em caso de erro, a aplicação termina sempre apresentando o motivo do erro.

# Configuração de uma rede

## Configurar um IP de Rede

A primeira experiência tem como objetivo configurar dois computadores na mesma rede, permitindo o estabelecimento de uma comunicação entre os mesmos. Para tal, configuramos o tux61 e o tux64 (trabalhamos sempre na bancada 6 da sala I321), usando o comando *ifconfig*, para que estes assumissem, respetivamente, os IPs 172.16.60.1 e 172.16.60.254 nas portas eth0. Depois de configuradas as rotas e para verificar a conectividade entre os dois tuxes, foi chamado o comando *ping,* cujo resultado foi analisado pelo programa Wireshark. Este comando gera um pacote ICMP que utiliza para transferir mensagens de controlo entre endereços IP. Analisando o log gerado, podemos verificar o envio de uma mensagem broadcast a todos computadores pelo pacote ARP, protocolo que atribui a um IP o respetivo endereço de hardware, MAC, para cada placa de rede, com o objetivo de saber qual o mesmo, para que os pacotes ICMP possam receber resposta apenas deste último. Para verificar se estam ativos a ligação e o sistema, são enviados pacotes LOOP de 10 em 10 segundos pela interface *loopback.*

## Implementar duas LANs virtuais no switch

A segunda experiência implica a criação de duas LANs virtuais no switch, denomindas vlan60 e vlan61. À primeira pertencem os computadores tux61 e tux64 e à segunda, para já, o tux62.

## Configurar um router em Linux

## Configurar um router comercial e implementar NAT

## DNS

## Ligações TCP

# Conclusão

Após a conclusão do segundo projeto, foi possível interiorizar os a maior parte dos conceitos necessários para a implementar do que era pedido no guião e perceber ainda melhor algo que é usado no nosso dia-a-dia.

A implementação do cliente FTP foi concluída com sucesso, uma vez que é possível fazer o download correto de diferentes tipos e tamanhos de um ficheiro. Com a implementação deste novo protocolo, o grupo consguiu fazer um paralelo com o protocolo implementado no primeiro projeto. Assim, percebemos melhor as diferenças entre um protocolo aplicado e usado na integra e um protocolo que quase entrou em desuso (porta de série).

A configuração de uma rede também foi concluída com sucesso.

Podemos concluir que o projeto foi concluído com sucesso, visto que todos os objetivos do segundo guião foram implementados com sucesso.

# Anexo – Código fonte da Aplicação de Download

# clientTCP.c

#include <arpa/inet.h>

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include "parser.h"

#include "tcp.h"

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

if (argc != 2) {

fprintf(stderr, "Usage: %s <address>\n", argv[0]);

exit(1);

}

urlInfo info;

if (parseURL(argv[1], &info) != 0) {

fprintf(stderr, "Invalid URL\n");

exit(1);

}

int controlSocketFD;

if ((controlSocketFD = createConnection(

inet\_ntoa(\*((struct in\_addr \*)info.infoHost->h\_addr)),

CLIENT\_CONNECTION\_PORT)) == 0) {

fprintf(stderr, "Error opening control connection\n");

exit(1);

}

login(controlSocketFD, &info);

char data\_address[MAX\_STRING\_SIZE];

int port;

passiveMode(controlSocketFD, data\_address, &port);

int dataSocketFD;

if ((dataSocketFD = createConnection(data\_address, port)) == 0) {

fprintf(stderr, "Error opening data connection\n");

exit(1);

}

sendAndRetrieve(controlSocketFD, &info);

downloadFile(dataSocketFD, &info);

closeConnection(controlSocketFD, dataSocketFD);

return 0;

}

parser.h

#ifndef \_\_PARSE\_URL\_H\_\_

#define \_\_PARSE\_URL\_H\_\_

#define LINK\_HEADER "ftp://"

#define DEFAULT\_USER "anonymous"

#define DEFAULT\_PASSWORD "mail@domain"

typedef struct {

char user[256];

char password[256];

char urlHost[256];

struct hostent\* infoHost;

char path[256];

char filename[256];

} urlInfo ;

/\*\*

\* @brief Retrieves the username, password and download file name from a ftp URL

\* @param url URL with the information to parseURL

\* @param info Struct to complete with the information read from the URL

\* @returns 0 if URL is correct

\*/

int parseURL(char url[], urlInfo\* info);

#endif

parser.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include "parser.h"

/\*\*

\* Init Default User (Anonymous:mail@domain)

\* @method initDefaultUser

\* @param info link info

\*/

void initDefaultUser(urlInfo \*info) {

memcpy(info->user, DEFAULT\_USER, strlen(DEFAULT\_USER) + 1);

memcpy(info->password, DEFAULT\_PASSWORD, strlen(DEFAULT\_PASSWORD) + 1);

};

/\*\*

\* Init User

\* @method initUser

\* @param info link info

\* @param url link

\* @param atPos slash pos

\* @return

\*/

int initUser(urlInfo \*info, char \*url, char \*atPos) {

char \*slashChar = strchr(url, '/'); // slashChar can't be null

slashChar += 2;

char \*password = strchr(slashChar, ':');

if (password == NULL) {

fprintf(stderr, "There must be a ':' separating the username and "

"password!' on the link\n");

return 1;

}

memcpy(info->user, slashChar, password - slashChar);

info->user[password - slashChar] = 0;

password++;

memcpy(info->password, password, atPos - password);

info->password[atPos - password] = 0;

return 0;

}

/\*\*

\* parse link to urlInfo struct

\* @method parseURL

\* @param url string of link

\* @param info urlInfo

\* @return

\*/

int parseURL(char url[], urlInfo \*info) {

if (strncmp(url, LINK\_HEADER, strlen(LINK\_HEADER)) != 0) {

fprintf(stderr, "Your link must begin with 'ftp://'\n");

return 1;

}

char \*atPos = strrchr(url, '@');

if (atPos == NULL) {

initDefaultUser(info);

atPos = url + strlen("ftp://");

} else {

if (initUser(info, url, atPos) != 0)

return 1;

atPos++;

}

// find slashes for parsing of information

char \*firstSlashChar = strchr(atPos, '/');

memcpy(info->urlHost, atPos, firstSlashChar - atPos);

info->urlHost[firstSlashChar - atPos] = 0;

char \*lastSlashChar = strrchr(url, '/');

lastSlashChar++;

memcpy(info->path, firstSlashChar, lastSlashChar - firstSlashChar);

info->path[lastSlashChar - firstSlashChar] = 0;

memcpy(info->filename, lastSlashChar, strlen(lastSlashChar) + 1);

if ((info->infoHost = gethostbyname(info->urlHost)) == NULL) {

herror(info->urlHost);

exit(1);

}

return 0;

}

tcp.h

#include "parser.h"

#define CLIENT\_CONNECTION\_PORT 21

#define MAX\_STRING\_SIZE 512

/\*\*

\* creat connection to address

\* original code from the clientTCP.c file given on moodle

\* @method createConnection

\* @param address

\* @param port

\* @return

\*/

int createConnection(char\* address, int port);

/\*\*

\* Login to user

\* @method login

\* @param controlSocketFD socket's file descriptor

\* @param info info of user to login

\*/

void login(int controlSocketFD, urlInfo\* info);

/\*\*

\* Enter Passive Mode

\* @method passiveMode

\* @param sockfd socket's file descriptor

\* @param ip

\* @param port

\*/

void passiveMode(int sockfd, char\* ip, int\* port);

/\*\*

\* Send command to retrieve Data from file to download

\* @method sendAndRetrieve

\* @param controlSocketFD control socket's file descriptor

\* @param info link's info

\*/

void sendAndRetrieve(int controlSocketFD, urlInfo\* info);

/\*\*

\* Download File

\* @method downloadFile

\* @param dataSocketFD data socket's file descriptor

\* @param info link's info

\* @return

\*/

int downloadFile(int dataSocketFD, urlInfo\* info);

/\*\*

\* Close Connection

\* @method closeConnection

\* @param controlSocketFD control socket's file descriptor

\* @param dataSocketFD data socket's file descriptor

\* @return

\*/

int closeConnection(int controlSocketFD, int dataSocketFD);

tcp.c

#include <arpa/inet.h>

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include "tcp.h"

#define READ 1

#define NO\_READ 0

/\*\*

\* Read From Socket

\* @method readSocket

\* @param sockfd socket's file descriptor

\* @param str to read

\* @return

\*/

int readSocket(int sockfd, char \*str) {

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

int allocated = 0;

if (str == NULL) {

str = (char \*)malloc(sizeof(char) \* MAX\_STRING\_SIZE);

allocated = 1;

}

do {

memset(str, 0, MAX\_STRING\_SIZE);

str = fgets(str, MAX\_STRING\_SIZE, fp);

printf("%s", str);

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

char reply\_series = str[0];

if (allocated)

free(str);

return (reply\_series > '4');

}

/\*\*

\* Write to Socket

\* @method writeSocket

\* @param sockfd socket's file descriptor

\* @param cmd command to write

\* @param response response from read

\* @param read

\* @return

\*/

int writeSocket(int sockfd, char \*cmd, char \*response, int read) {

int return\_val = write(sockfd, cmd, strlen(cmd));

if (read)

return readSocket(sockfd, response);

else

return (return\_val == 0);

}

int createConnection(char \*address, int port) {

int sockfd;

struct sockaddr\_in server\_addr;

/\*handling server address (part of clientTCP original code)\*/

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

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

server\_addr.sin\_addr.s\_addr =

inet\_addr(address); /\*32 bit Internet address network byte ordered\*/

server\_addr.sin\_port =

htons(port); /\*server TCP port must be network byte ordered \*/

/\*open TCP socket\*/

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

perror("socket()");

return 0;

}

/\*connect to server\*/

if (connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) <

0) {

perror("connect()");

return 0;

}

return sockfd;

}

void login(int controlSocketFD, urlInfo \*info) {

char username\_cmd[MAX\_STRING\_SIZE], password\_cmd[MAX\_STRING\_SIZE];

readSocket(controlSocketFD, NULL);

sprintf(username\_cmd, "USER %s\r\n", info->user);

writeSocket(controlSocketFD, username\_cmd, NULL, READ);

sprintf(password\_cmd, "PASS %s\r\n", info->password);

if (writeSocket(controlSocketFD, password\_cmd, NULL, READ) != 0) {

fprintf(stderr, "Bad login. Exiting...\n"); // TODO: Ask for valid login

exit(1);

}

}

void passiveMode(int sockfd, char \*ip, int \*port) {

char response[MAX\_STRING\_SIZE];

if (writeSocket(sockfd, "PASV\r\n", response, READ) != 0) {

fprintf(stderr, "Error entering passive mode. Exiting...\n");

exit(1);

}

int values[6];

char \*data = strchr(response, '(');

sscanf(data, "(%d, %d, %d, %d, %d, %d)", &values[0], &values[1], &values[2],

&values[3], &values[4], &values[5]);

sprintf(ip, "%d.%d.%d.%d", values[0], values[1], values[2], values[3]);

\*port = values[4] \* 256 + values[5];

}

void sendAndRetrieve(int controlSocketFD, urlInfo \*info) {

char cmd[MAX\_STRING\_SIZE];

writeSocket(controlSocketFD, "TYPE L 8\r\n", NULL, READ);

sprintf(cmd, "RETR %s%s\r\n", info->path, info->filename);

if (writeSocket(controlSocketFD, cmd, NULL, READ) != 0) {

fprintf(stderr, "Error retrieving file. Exiting...\n");

exit(1);

}

}

int downloadFile(int dataSocketFD, urlInfo \*info) {

FILE \*fileToDownload;

if (!(fileToDownload = fopen(info->filename, "w"))) {

printf("ERROR: Cannot open file.\n");

return 1;

}

char buffer[1024];

int sizeInBytes;

while ((sizeInBytes = read(dataSocketFD, buffer, sizeof(buffer)))) {

if (sizeInBytes < 0) {

fprintf(stderr, "ERROR: Nothing was received from data socket fd.\n");

return 1;

}

if ((sizeInBytes = fwrite(buffer, sizeInBytes, 1, fileToDownload)) < 0) {

fprintf(stderr, "ERROR: Cannot write data in file.\n");

return 1;

}

}

fclose(fileToDownload);

printf("Finished file's download!\n");

return 0;

}

int closeConnection(int controlSocketFD, int dataSocketFD) {

printf("Closing connection\n");

if (writeSocket(controlSocketFD, "QUIT\r\n", NULL, NO\_READ) != 0) {

fprintf(stderr, "Error closing connection. Exiting anyway...\n");

close(dataSocketFD);

close(controlSocketFD);

exit(1);

}

close(dataSocketFD);

close(controlSocketFD);

printf("End of Program!\n");

return 0;

}