Redes de Computadores

Configuration of a network and development of download application

Flávio Couto, João Gouveia and Pedro Afonso Castro

December 24, 2015

Abstract

This report aims to explain our approach regarding the second project proposed by our teachers in the subject "Redes de Computadores". This project was split into two parts. The development of a download application, using TCP sockets to connect to an FTP server to download the file requested by the user, and into the configuration and studying of a computer network, consisting in configuring an Internet Protocol (IP) network, implementing virtual LANs (VLANs) in a switch, configuring a router and a commercial router, implementing Network Address Translation (NAT) and configuring the Domain Name System (DNS), establishing a TCP connection (using the application developed in the first part of the project).

Our group managed to comprehend and reach all the purposed goals with success, as we will show in this report.

Contents

1 Introduction

The purpose of this project is twofold: develop an application capable of downloading files from an FTP server through TCP sockets and implement a computer network. To implement the computer network, several equipment was used, namely a switch, a commercial router and a computer with Linux. The appliction, just like the serial port protocol implemented on the previous project, works on any computer capable of interpreting POSIX calls.

In what concerns the report, its main objective is not only to describe the approach we used in this project, regarding both parts of the project, but also to show what we learnt and concluded from the development of this project. Tests are also provided in order to demonstrate the result of our work.

This report is composed of an introduction, where the aim of the project and of this report are explained, the FTP Client section, where the application is described, a Network configuration section, where the computer netword is described, a conclusion, where we explain what we learnt from this project, evaluate our performance and our overall opinion regarding the project, and the appendices section, where we place the code and some images that help with the comprehension of each part.

2 FTP Client

As previously stated, one of this project's parts was developing an FTP Client capable of connecting to an FTP server, logging in (either with the credentials provided by the user, or in anonymous mode), and downloading a file requested by the user. Said FTP Client was developed in the C programming language, and to develop it several documents regarding the File Transfer Protocol had to be studied, namely RFCs (Request for Comments) 959, describing the FTP Protocol specification, and 1738, regarding URL syntax. In order to communicate with the FTP server, we resort to socket programming, more specifically, to TCP sockets.

2.1 Architecture of the download application

The application is essentialy split into two main modules: the URL Parsing Module and the FTP Client module. The URL Parsing module validates the URL, determining if it is an authenticated or anonymous connection, and parses it onto several variables to be used by the FTP Client, which makes the connection, authentication, file downloading and disconnection with said data. In addition to that, both modules have some debug messages shown in the terminal if the symbolic constant DEBUG is defined. In order to make them appear, just define

said constant in the ftpClient.h file.

2.1.1 The URL Parsing Module

As previously mentioned, the URL Parsing module is divided into two parts: validation of the URL and subsequent parsing of said URL. The implementation of this module can be found in the parseUrl.h and parseUrl.c files. The URL validation is made using POSIX's implementation of regular expressions (regex for short). Two regular expressions are used, one for authenticated login and one for anonymous login:

```
char* regex_auth = "ftp://[A-Za-z0-9]+:[A-Za-z0-9]+@[A-Za-z0
    -9._~:?#!$&'()*+,;=-]+/[A-Za-z0-9._~:/?#@!$&'()*+,;=-]+"
    ;
char* regex_auth_anon = "ftp://[A-Za-z0-9._~:?#!$&'()
    *+,:;=-]+/[A-Za-z0-9._~:/?#@!$&'()*+,:;=-]+";
```

The URL validation is done in the following functions:

```
int validateURL(char* url, int size);
int validateURLAnon(char* url, int size);
```

The first function is public and the one called by main() to validate a URL. The second function is used only in the module, called by validateURL() in case the authenticated connection regex fails. Each one uses its respective regex. validateURL() returns 0 on authenticated connection, 1 on anonymous connection, 2 on invalid URL and -1 on an error situation.

The parsing is done with the following function:

```
void parseURL(char* url, int size, char* host, char* user,
    char* password, char* path, int anon);
```

The anon variable is a boolean variable which is equal to 1 if it's a anonymous url or 0 otherwise. The host, user, password and path are filled with their respective values, being the password asked to the user in case of an anonymous login.

2.1.2 The FTP Client Module

The FTP Client Module can be found in the ftpClient.c and ftpClient.h files. After parsing the URL, the information is sent to the FTP Client module, which then does the following steps:

1. Connects to the FTP server through a TCP socket created in the ftp_connect_socket() function using the ftp_connect() function.
2. Logins to the FTP server using the ftp_login_host() function.

3. Enters Passive mode and opens another TCP socket (the data socket) in the ftp_set_passive_mode() function.
4. Requests the file in the ftp_retr_file() function, downloading it in the ftp_download_file() function.
5. Disconnects from the server and closes the TCP sockets with the ftp_disconnect() function.

Besides the stated functions, ftp_send_command() and ftp_read_answer() are used to send a command to the ftp server and read its reply, respectively. In order to connect to the server through the TCP socket, an IP address is required. So, the getIpAddress() function is used for said purpose.

The FTP Client module contains a structure with the information necessary for every step described previously:

```
typedef struct {
        char server_address[IP_MAX_SIZE]; //NNN.NNN.NNN.NNN
        int socketfd;
        int datafd;
        char username[MAX_STRING_SIZE];
        char password[MAX_STRING_SIZE];
        char path_to_file[MAX_STRING_SIZE];
        int file_size;
} ftp_t;
```

The server_address field contains the IP address of the FTP server provided by the user, translated by the function getIpAddress().

The socketfd and datafd contain the file descriptors of the TCP sockets to communicate with the server. The username, password and path fields are the same from the URL Parsing module. The file_size stores the file size given by the FTP server, used for calculating the current progress and to check the file's integrity.

2.2 Report of a successful download

We decided to perform two tests: an authenticated connection to the FTP server tom.fe.up.pt, requesting the file public_html/reader/index.html and an anonymous connection to the FTP server speedtest.tele2.net requesting the file 1KB.zip.

The first test results can be seen in Fig. 1 and the second test results can be seen in Fig. 2.

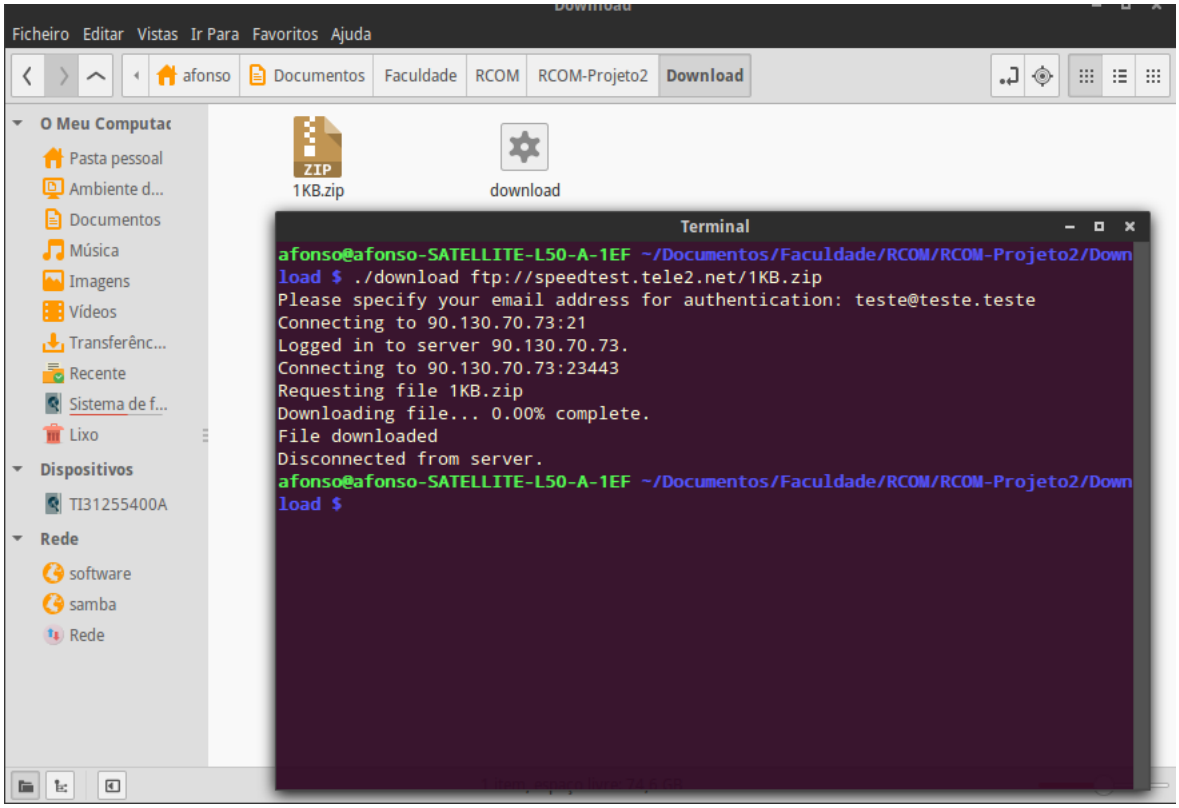

Figure 1: Result of the authenticated connection test

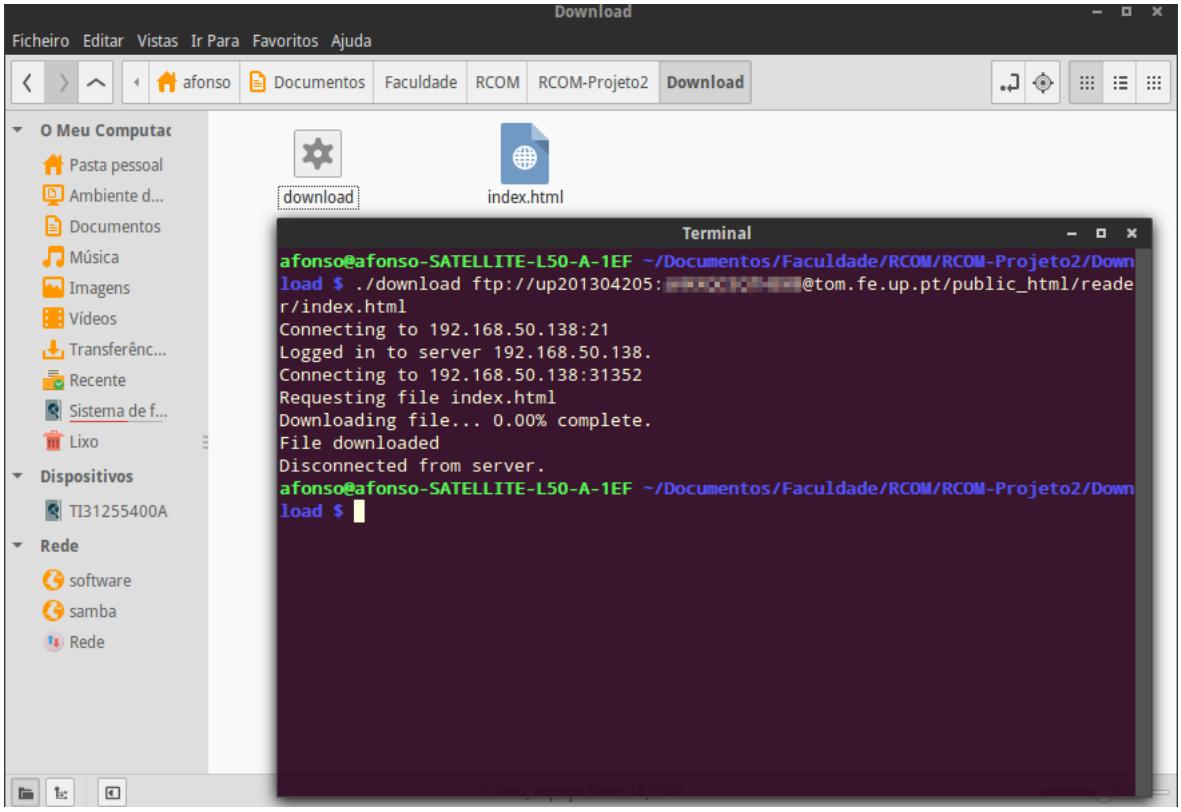

Figure 2: Result of the anonymous connection test

3 Network Configuration and Analysis

The second half of this project consisted in configuring and analysing a network and the trafic on it. For that we did some experiences in which we used the configuration commands present in section B of the Appendices for each experience and obtained logs with the Wireshark software that are annexed to this document in the locations following the format tuxT/expE-stepSTuxX[OtherInfo].

3.1 Experience 1 - Configure an IP network

This experience's objective was to configure a machine network interface IP address and study ARP and ICMP packets. In order to do that Tux1 eth0 interface was assigned the IP adress 172.16.50.1/24 and Tux4 eth0 interface was assigned the IP address 172.16.50.254/24. Both interfaces were connected to the default vlan on the switch (Tux1 eth0 in port Fa 0/1, Tux4 eth0 in port Fa 0/4).

After configuring the network we issued a ping to 172.16.50.254 from Tux1. It generate an ARP packet in order to discover the MAC address whose IP was the requested one. Tux4 replied to Tux1 with its MAC address. After that, an ICMP Echo packet was issued from Tux1 having its MAC and IP as source and having Tux4 MAC and IP as destination. Upon receiving this packet, Tux4 issued an ICMP Echo Reply as response with its MAC and IP as source and Tux1 MAC and IP as destination. By capturing the internet packets navigating through Tux1 eth0 with the Wireshark software we can analyse them in order to inspect their contents and obtain informations such as its ethernet type in the ethernet header, which allows us to distinguish between ARP (0x0806) and IP (0x0800) packects, its IP protocol in the IP header protocol field, which is 1 for ICMP packets, and the length of the receiving frame, which is also in a field of the IP datagram header.

Although it was already configured, the loopback interface is also one of the machine's network interfaces. It is a virtual network interface that the machine uses to communicate with itself. It is used mainly for diagnostics and troubleshooting, and to connect to servers running on the local machine.

3.2 Experience 2 - Implement two virtual LANs in a switch

In this experience with learnt how to setup virtual LANs in a switch and how they affected the communication between the network interfaces connected to the switch.

We began by adding Tux2 eth0 to our setup, assigning it the IP address 172.16.51.1/24 and connecting it to the switch in port Fa 0/2. Then we setup vlan50, added ports Fa 0/1 and Fa

0/4 to it, setup vlan51 and added port Fa 0/2 to it. After the setup, we used Wireshark and tryed to ping Tux2 and Tux4 from Tux1 but there was no response from Tux2, because Tux2 is not on the same vlan as Tux1 and Tux4.

Next we started Wireshark in all the machines and made a ping broadcast to IP 172.16.50.255 from Tux1. Tux4 replied, but not Tux2. Then we made a ping broadcast to IP 172.16.51.255 from Tux2. This time none of the other machines replied. With this we can conclude that there are 2 broadcast domains.

3.3 Experience 3 - Configure a router in Linux

This experience consisted in making one of the Tux machines into a router, connecting the two networks created previously.

For this, tux4 eth1 interface was assigned the IP address 172.16.51.253/24, connected to port Fa 0/5 on the switch and added to vlan51. Then a route was added on tux2 to reach the network 172.16.50.0/24 through gateway 172.16.51.253 and the ip address 172.16.50.254 was assigned as tux1 default gateway.

These configurations allowed communication between the 3 tux machines, which we tested with some ping commands that generated some more ARP, ICMP Echo and ICMP Echo Reply packets.

Each one of the tux machines as a forwarding table which contains information about a destination network, the gateway to reach it (which is the next hop on the connection), the interface associated with it and a metric, which evaluates the efficiency of a route. Tux1 has 2 routes on its table: the route to network 172.16.50.0/24, which it has direct access through eth0, and a route to all the other networks with 172.16.50.254 as the gateway, making it tux1 default gateway.

Tux2 has 2 routes on its table: the route to network 172.16.51.0/24, which it has direct access through eth0, and a route to 172.16.50.0/24 with 172.16.51.253 as the gateway.

Tux3 also has 2 routes on its table: the route to network 172.16.50.0/24, which it has direct access through eth0, and the route throught 172.16.51.0/24, which it has direct access through eth1.

When pinging tux2 from tux1, the packet traveling in 172.16.50.0/24 has source IP and MAC address corresponding to tux1 eth0 but, although its destination IP address is tux2 eth0's IP address, the MAC address is tux4 eth0's IP address. This packet is then received by tux4 which forwards it to tux2, but it changes the source MAC to tux4 eth1's MAC and the destination MAC to tux2 eth0's MAC. A similar operation happens with packets travelling to tux1 from

tux2, as expected.

3.4 Experience 4 - Configure a Commercial Router and Implement NAT

In order to do so, the router gigabitethernet 0/0 interface was assigned IP address 172.16.51.254/24 and was connected to the switch port Gi 0/1, which was added to vlan51. Then router gigabitethernet 0/1 interface was assigned IP address 172.16.1.59/24 and connected to the lab network. After that we made 172.16.51.254 the default gateway for tux4 and tux2 and added a route on the router for network 172.16.50.0/24 using 172.16.51.253 as the gateway. With these configurations we were able to ping all the network interfaces of tux4, tux2 and the router from tux1.

To analyse route redirections we disabled the acceptance of route redirections on tux2 and removed the route to the network 172.16.50.0/24. Then we used the traceroute utility to check the path taken to 172.16.50.1, which was: tux2, router, tux4, tux1. After restoring the route the path became: tux2, tux4, tux1. By removing again the route but accepting redirects on tux2, the path followed by the first communication was: tux2, router, tux4, tux1 but the following ones were: tux2, tux4, tux1 because the ICMP Redirect packet sent by the router was accepted by tux2, which then started using this more efficient route to network 172.16.50.0/24 than the one it knew.

In the end, we enabled NAT on the router, making the gigabitethernet 0/1 the interface connected to the inside network and the gigabitethernet 0/2 the one conneted to the outside network and allowing access to the networks 172.16.50.0/24 and 172.16.51.0/24. This allowed us to communicate with machines outside our local network through their IP address by translating a machine local IP address to a public IP address by mapping its datagrams IP address and port to a port on the device making the translation and them making the translation backwards when receiving and answer.

3.5 Experience 5 - DNS

This experience was centered around using a DNS to access other machines by its domain names. To do that we set IP address 172.16.1.1 as the nameserver for each of the Tux machines.

After that we sent some pings to a domain name. Before issuing the ICMP Echo packet the machine sent an UDP packet with a DNS query to 172.16.1.1 asking for the IP address corresponding to the domain name, which was sent by the nameserver as an answer.

3.6 Experience 6 - TCP Connection

In this experience we tested our ftp download application on the network we were configuring.

We started by running it in tux1 and were able to download the requested file, which led us to conclude that both the download application and the network were well configured.

The download application opens 2 TCP connections, one connected to the ftp server port 21 to send the control information and another one connected to the data port provided by the server. These TCP connections are divided in 3 phases: the connection establishement, which is a 3-way handshake, the data transfer, in which the connection data is transmitted, and the connection termination, which is a 4-way handshake. When transfering the data the TCP connetions relies on the a variation of the Go-Back-N ARQ mechanism in order to maintain the integrity of the transmited data. The TCP datagram contains a sequence number field, an acknowledgment number field and an window size field, which are needed to use this ARQ mechanism. In order to achieve a high and stable throughput, the TCP connection has a congestion control mechanism based on ACK's received, which increase additively the throughput, and timeouts occured, which decrease multiplicatively the throughput.

In the end we opened two instances of the download application, one on tux1 and another on tux2, and analysed the impact of it in the throughput of the connection on tux1. We concluded that the throughput diminished almost to half, as expected.

4 Conclusion

After concluding this project, the group considers they understood all the topics asked and managed to solve all the problems proposed, both from the FTP application part and the network configuration and analysis part. All group elements consider they've understood the basics of implementing a computer network, and we had no trouble developing the application and studying the RFC's needed to develop it.

We'd like to thank our teacher for helping us several times during our classes. Without that support, this project would have been much more difficult to complete, especially some key concepts of the computer network part.

All in all, we all consider the group's objectives were successfully completed.

Appendices

A Application Source code

ftpClient.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <strings.h>
#include <string.h>
#include <errno.h>
#include <arpa/inet.h>
#include <libgen.h>

#include "ftpClient.h"

int getIpAdress(const char* server_addr, char* buf){
        struct hostent *h;

         if ((h=gethostbyname(server_addr)) == NULL)
                return -1;

    strcpy(buf, inet_ntoa(*((struct in_addr *)h->h_addr)));

    #ifdef DEBUG
    printf("\nDEBUG: Server info:\n");
    printf("DEBUG: Host name  : %s\n", h->h_name);
    printf("DEBUG: IP Address : %s\n\n", buf);
    #endif
```

```
    return 0;
}

ftp_t* ftp_init(const char* server, const char* username, const char*
    password, const char* path_to_file){

        ftp_t* ftp = malloc(sizeof(ftp_t));
        memset(ftp, 0, sizeof(ftp_t));
        char server_address[IP_MAX_SIZE];

        if(getIpAdress(server, server_address) < 0){
                printf("Error while resolving address %s\n", server);
                return NULL;
        }

        strcpy(ftp->server_address, server_address);
        strcpy(ftp->username, username);
        strcpy(ftp->password, password);
        strcpy(ftp->path_to_file, path_to_file);

        return ftp;
}

int ftp_connect_socket(const char* ip, const int port){
        int     sockfd;
        struct  sockaddr_in server_addr;

        #ifdef DEBUG
        printf("DEBUG: Starting server address handling\n");
        #endif

        /*server address handling*/
        memset((char*)&server_addr,0,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 */

    #ifdef DEBUG
        printf("DEBUG: Finished service handling\n");
        #endif

        /*open a TCP socket*/
        if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
                perror("socket()");
                        return -1;
    }

    #ifdef DEBUG
    printf("DEBUG: Finished opening TCP Socket \n");
    #endif

    printf("Connecting to %s:%d\n", ip, port);
        /*connect to the server*/
    if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(
       server_addr)) < 0){
        perror("connect()");
                return -1;
        }

    #ifdef DEBUG
        printf("DEBUG: Connected\n");
        #endif

        return sockfd;
}
```

```
int ftp_connect(ftp_t* ftp){
        int sockfd = ftp_connect_socket(ftp->server_address, FTP_PORT
            );

        if(sockfd < 0){
                printf("Error while connecting to server\n");
                return -1;
        }

        ftp->socketfd = sockfd;

        char answer[MAX_STRING_SIZE];
        ftp_read_answer(ftp, answer, MAX_STRING_SIZE);

        #ifdef DEBUG
        printf("DEBUG: Finished connecting to the server\n");
        #endif

        return sockfd;
}

int ftp_send_command(ftp_t* ftp, const char* command, const int size)
    {
        #ifdef DEBUG
        printf("DEBUG: Sending command %s with size %d\n", command,
            size);
        #endif

        int bytesSent = write(ftp->socketfd, command, size);

        if(bytesSent <= 0){
                printf("Error while sending command to server (no
                    information sent)\n");
```

```
                return -1;
        }

        if(bytesSent != size){
                printf("Error while sending command to server (
                    information partially sent)\n");
                return -1;
        }

        #ifdef DEBUG
                printf("DEBUG: Sent command %s, %d bytes written\n",
                    command, bytesSent);
        #endif

        return 0;
}

int ftp_read_answer(ftp_t* ftp, char* answer, const int size){
        int bytesRead;
        memset(answer, 0, size);
        bytesRead = read(ftp->socketfd, answer, size);
        #ifdef DEBUG
        printf("DEBUG: %s", answer);
        #endif
        return bytesRead;
}

int ftp_login_host(ftp_t* ftp){

        char command[6 + strlen(ftp->username)];
        char answer[MAX_STRING_SIZE];

        sprintf(command,"user %s\n", ftp->username);
        if(ftp_send_command(ftp, command, strlen(command)) < 0){
```

```
            printf("Error while sending login information\n");
            return -1;
        }
        if(ftp_read_answer(ftp, answer, MAX_STRING_SIZE) < 0){
            printf("Error while receiving answer from login
                information \n");
            return -1;
        }

        sprintf(command,"pass %s\n", ftp->password);

        if(ftp_send_command(ftp, command, strlen(command)) < 0){
            printf("Error while sending login information\n");
            return -1;
        }

        if(ftp_read_answer(ftp, answer, MAX_STRING_SIZE) < 0){
            printf("Error while receiving answer from login
                information \n");
            return -1;
        }

        int replyCode;
        sscanf(answer, "%d", &replyCode);
        if(replyCode == LOGIN_FAIL){ /* Failed to login */
            printf("Wrong username/password combination.\n");
            return -1;
        }

        printf("Logged in to server %s.\n",ftp->server_address);

        return 0;
}
```

```
int ftp_set_passive_mode(ftp_t* ftp){
        char answer[MAX_STRING_SIZE];
        int ip1, ip2, ip3, ip4;
        int port1, port2;
        if(ftp_send_command(ftp, "pasv\n", strlen("pasv\n")) < 0){
                printf("Error while switching to passive mode\n");
                return -1;
        }
        if(ftp_read_answer(ftp, answer, MAX_STRING_SIZE) < 0){
                printf("Error while receiving answer from switching
                   to passive mode \n");
                return -1;
        }

        #ifdef DEBUG
                printf("%s", answer);
        #endif

        if ((sscanf(answer, "227 Entering Passive Mode (%d,%d,%d,%d,%
           d,%d)", &ip1, &ip2, &ip3, &ip4, &port1, &port2)) < 0){
                printf("Error while parsing pasv answer from the
                   server.\n");
                return -1;
        }

        char ip[15];
        int port = 256*port1+port2;
        sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

        #ifdef DEBUG
        printf("IP: %s Port: %d\n", ip, port);
        #endif

        int datafd = ftp_connect_socket(ip, port);
```

```
        if(datafd < 0){
                printf("Error while connecting to data socket\n");
                return -1;
        }

        ftp->datafd = datafd;
        return 0;
}

int ftp_download_file(ftp_t* ftp){

        char filename[MAX_STRING_SIZE];
        char buf[DATA_PACKET_SIZE];
        sprintf(filename, "%s", basename(ftp->path_to_file));
        int bytesRead = 0;
        int totalBytes = 0;

        FILE* file = fopen(filename, "w");
        if(file == NULL){
                printf("Couldn't open file %s for writing\n",
                   filename);
                return -1;
        }

        while((bytesRead = read(ftp->datafd, buf, sizeof(buf))) != 0)
           {
                printf("Downloading file ... %.2f%% complete.\n",
                   (100.0 * totalBytes) / ftp->file_size);
                if(bytesRead < 0){
                        printf("Error in reading file from server\n")
                           ;
                        return -1;
                }
                totalBytes += bytesRead;
```

```
                if(fwrite(buf,sizeof(char),bytesRead,file) == 0)
                        if(ferror(file) != 0){
                                printf("Error in writing to file %s\n
                                    ", filename);
                                return -1;
                        }
        }
        printf("File downloaded \n");
        fclose(file);
        return totalBytes;
}

int ftp_retr_file(ftp_t* ftp){
        char command[6+strlen(ftp->path_to_file)];
        char answer[MAX_STRING_SIZE];
        sprintf(command, "retr %s\n", ftp->path_to_file);

        printf("Requesting file %s\n",basename(ftp->path_to_file));

        if(ftp_send_command(ftp, command, 6+strlen(ftp->path_to_file)
            ) < 0){
                printf("Error in requesting file from server\n");
                return -1;
        }
        if(ftp_read_answer(ftp, answer, MAX_STRING_SIZE) < 0){
                printf("Error in reading answer from file request
                    from server\n");
                return -1;
        }

        int replyCode;
        sscanf(answer, "%d", &replyCode);
        if(replyCode == OPEN_FILE_FAIL){ /* Failed to find file */
                printf("Couldn't find file %s in server.\n", ftp->
```

```
                path_to_file);
            return -1;
        }
        else if(replyCode == OPEN_FILE_SUCCESS){
            /* Find the file size to export it */
            char* beg = strrchr(answer,'(');
            char sizeInfo[MAX_STRING_SIZE];
            strcpy(sizeInfo, beg);
            int size;
            sscanf(sizeInfo, "(%d bytes).", &size);
            ftp->file_size = size;
            #ifdef DEBUG
            printf("DEBUG : Size Info: %d bytes\n", size);
            #endif
        }

        #ifdef DEBUG
            printf("%s\n", answer);
        #endif

        return 0;
}

int ftp_disconnect(ftp_t* ftp){
        char answer[MAX_STRING_SIZE];
        if(ftp_read_answer(ftp, answer, MAX_STRING_SIZE) < 0){
            printf("Error in disconnecing from server\n");
            return -1;
        }

        if(ftp_send_command(ftp, "quit\n", strlen("quit\n")) < 0){
            printf("Error in sending quit command\n");
            return -1;
        }
```

```
        if(close(ftp->socketfd) < 0){
                printf("Failed to close ftp socket.\n");
                return -1;
        }

        if(close(ftp->datafd) < 0){
                printf("Failed to close ftp data socket.\n");
                return -1;
        }

        printf("Disconnected from server.\n");
        return 0;
}

void ftp_delete(ftp_t* ftp){
        free(ftp);
}
```

ftpClient.h:

```
#ifndef __tcpClient
#define __tcpClient

#define FTP_PORT 21
#define MAX_STRING_SIZE 255
#define IP_MAX_SIZE      15
#define OPEN_FILE_FAIL 550
#define LOGIN_FAIL 530
#define OPEN_FILE_SUCCESS 150
#define DATA_PACKET_SIZE 8192
//#define DEBUG

typedef struct {
```

```
        char server_address[IP_MAX_SIZE]; //NNN.NNN.NNN.NNN
        int socketfd;
        int datafd;
        char username[MAX_STRING_SIZE];
        char password[MAX_STRING_SIZE];
        char path_to_file[MAX_STRING_SIZE];
        int file_size;
} ftp_t;

int getIpAdress(const char* server_addr, char* buf);
ftp_t* ftp_init(const char* server, const char* username, const char*
    password, const char* path_to_file);
int ftp_connect_socket(const char* ip, const int port);
int ftp_connect(ftp_t* ftp);
int ftp_send_command(ftp_t* ftp, const char* command, const int size)
   ;
int ftp_read_answer(ftp_t* ftp, char* answer, const int size);
int ftp_login_host(ftp_t* ftp);
int ftp_set_passive_mode(ftp_t* ftp);
int ftp_download_file(ftp_t* ftp);
int ftp_retr_file(ftp_t* ftp);
int ftp_disconnect(ftp_t* ftp);
void ftp_delete(ftp_t* ftp);

#endif
```

parseURL.c:

```
#include <regex.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>

#include "ftpClient.h"
```

```
#include "parseURL.h"

int validateURLAnon(char* url, int size);

int validateURLAnon(char* url, int size){
        regex_t regex;
        char* regex_auth_anon = "ftp://[A-Za-z0-9._~:?#!$&'()
            *+,:;=-]+/[A-Za-z0-9._~:/?#@!$&'()*+,:;=-]+";
        char regcomp_err[255];

        /* Compile regex */
        int reti = regcomp(&regex, regex_auth_anon, REG_EXTENDED);
        if (reti) {
                regerror(reti, &regex, regcomp_err, sizeof(
                    regcomp_err));
            printf("Error while validating URL\n");
            printf("Error while compiling regex: %s\n",regcomp_err);
            return -1;
        }

        /* Execute regex */
        reti = regexec(&regex, url, 0, NULL, 0);

        /* Validate regex */
        if (!reti) {
            #ifdef DEBUG
            printf("DEBUG: Anon regex matched sucessfully\n");
            #endif
            return 1;
        }
        else if (reti == REG_NOMATCH) {
            printf("Invalid URL!\n");
            return 2;
        }
```

```
        else {
            regerror(reti, &regex, regcomp_err, sizeof(regcomp_err));
            printf("Error while validating URL: %s\n", regcomp_err);
                #ifdef DEBUG
            printf("DEBUG: Error while matching regex: %s\n",
               regcomp_err);
            #endif
            return -1;
        }
}

int validateURL(char* url, int size){
        regex_t regex;
        char* regex_auth = "ftp://[A-Za-z0-9]+:[A-Za-z0-9]+@[A-Za-z0
           -9._~:?#!$&'()*+,;=-]+/[A-Za-z0-9._~:/?#@!$&'()*+,;=-]+"
           ;
        char regcomp_err[255];

        /* Compile regex */
        int reti = regcomp(&regex, regex_auth, REG_EXTENDED);
        if (reti) {
                regerror(reti, &regex, regcomp_err, sizeof(
                   regcomp_err));
            printf("Error while validating URL\n");
            printf("Error while compiling regex: %s\n",regcomp_err);
            return -1;
        }

        /* Execute regex */
        reti = regexec(&regex, url, 0, NULL, 0);

        /* Validate regex */
        if (!reti) {
            #ifdef DEBUG
```

```
            printf("DEBUG: Regex matched sucessfully\n");
            #endif
            return 0;
        }
        else if (reti == REG_NOMATCH) {
            return validateURLAnon(url, size);
        }
        else {
            regerror(reti, &regex, regcomp_err, sizeof(regcomp_err));
            printf("Error while validating URL: %s\n", regcomp_err);
                #ifdef DEBUG
            printf("DEBUG: Error while matching regex: %s\n",
               regcomp_err);
            #endif
            return -1;
        }

}

void parseURL(char* url, int size, char* host, char* user, char*
    password, char* path, int anon){
        // Remove ftp:// from string
        char* beg = url+6;
        #ifdef DEBUG
        printf("DEBUG: Url: %s\n",beg);
        #endif
        if(anon == 0) {
                char* end =  strchr(beg, ':');
                int i = 0;
                while(beg != end){
                        user[i] = *beg;
                        ++i;
                        ++beg;
                }
```

```
            ++beg;
            user[i] = 0;

            end=strchr(beg,'@');
            i = 0;
            while(beg != end){
                    password[i] = *beg;
                    ++i;
                    ++beg;
            }
            ++beg;
            password[i] = 0;
    }

    else {
            strncpy(user, "anonymous", sizeof("anonymous"));
            printf("Please␣specify␣your␣email␣address␣for␣
               authentication:␣");
            scanf("%s", password);
    }

    char* end = strchr(beg, '/');
    int i = 0;
    while(beg != end){
            host[i] = *beg;
            ++i;
            ++beg;
    }
    ++beg;
    host[i] = 0;

    strcpy(path, beg);

    #ifdef DEBUG
```

```
        printf("DEBUG: Host: %s, User: %s, Password: %s, Path: %s\n",
            host, user, password, path);
        #endif
}
```

parseURL.h:

```
#ifndef __parseURL
#define __parseURL

void parseURL(char* url, int size, char* host, char* user, char*
    password, char* path, int anon);
int validateURLAnon(char* url, int size);
int validateURL(char* url, int size);

#endif
```

main.c:

```
#include <stdio.h>
#include <stdlib.h>

#include "ftpClient.h"
#include "parseURL.h"

#ifndef NULL
#define NULL (void*)0
#endif

void printUsage(){
        printf("Usage: Normal: download ftp://<user>:<password>@<host
            >/<url-path>\n");
        printf("Usage: Anon: download ftp://<host>/<url-path>\n");
```

```
}

int main(int argc, char** argv){
        if(argc != 2){
                printUsage();
                return 1;
        }

        int val = validateURL(argv[1], sizeof(argv[1]));
        int anon = 1;
        char user[MAX_STRING_SIZE];
        char password[MAX_STRING_SIZE];
        char host[MAX_STRING_SIZE];
        char path[MAX_STRING_SIZE];
        switch(val){
                case 0:
                        anon = 0;
                case 1:
                        parseURL(argv[1], sizeof(argv[1]), host, user
                            , password, path, anon);
                        break;
                case 2:
                        printUsage();
                        return 1;
                default:
                        return 1;
        }

        ftp_t* ftp = ftp_init(host, user, password, path);
        if(ftp == NULL)
                return 1;

        if(ftp_connect(ftp) < 0)
```

```
        return 1;

    if(ftp_login_host(ftp) < 0)
        return 1;

    if(ftp_set_passive_mode(ftp) < 0)
        return 1;

    if(ftp_retr_file(ftp) < 0)
        return 1;

    if(ftp_download_file(ftp) < 0)
        return 1;

    if(ftp_disconnect(ftp) < 0)
        return 1;

    ftp_delete(ftp);

    return 0;
}
```

Makefile:

```
all:
    gcc -Wall -o download main.c ftpClient.c parseURL.c

clean:
    rm download
```

B Configuration commands

B.1 Configure an IP network

On tux1:

- ifconfig eth0 up
- ifconfig eth0 172.16.50.1/24

On tux4:

- ifconfig eth0 up
- ifconfig eth0 172.16.50.254/24

B.2 Implement two virtual LANs in a switch

On tux2:

- ifconfig eth0 up
- ifconfig eth0 172.16.51.1/24

On switch:

- configure terminal
- vlan 50
- exit
- vlan 51
- exit
- interface fastethernet 0/1
- switchport mode access
- switchport access vlan 50
- exit
- interface fastethernet 0/2
- switchport mode access
- switchport access vlan 51
- exit
- interface fastethernet 0/4
- switchport mode access
- switchport access vlan 50
- exit
- end

B.3 Configure a router in Linux

On tux1:

- route add default gw 172.16.50.254

On tux2:

- route add -net 172.16.50.0/24 gw 172.16.51.253

On tux4:

- ifconfig eth1 up
- ifconfig eth1 172.16.51.253/24
- echo 1 > /proc/sys/net/ipv4/ip_forward
- echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

On switch:

- configure terminal
- interface fastethernet 0/5
- switchport mode access
- switchport access vlan 51
- exit
- end

B.4 Configure a router in Linux

On tux2 and tux4:

- route add default gw 172.16.51.254

On switch:

- configure terminal
- interface gigabitethernet 0/1
- switchport mode access
- switchport access vlan 51
- exit
- end

On router:

- configure terminal
- interface gigabitEthernet 0/0
- ip address 172.16.51.254 255.255.255.0
- no shutdown
- ip nat inside

- exit
- interface gigabitEthernet 0/1
- ip address 172.16.1.59 255.255.255.0
- no shutdown
- ip nat outside
- exit
- ip nat pool ovrld 172.16.1.59 172.16.1.59 prefix 24
- ip nat inside source list 1 pool ovrld overload
- access-list 1 permit 172.16.50.0 0.0.0.255
- access-list 1 permit 172.16.51.0 0.0.0.255
- ip route 0.0.0.0 0.0.0.0 172.16.1.254
- ip route 172.16.50.0 255.255.255.0 172.16.51.253
- end

Disable tux2 redirection acceptance:

- echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
- echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects

Enable tux2 redirection acceptance:

- echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
- echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects

B.5 DNS

On tux1, tux2 and tux4 replace /etc/resolv.conf with:

- search netlab.fe.up.pt
- nameserver 172.16.1.1