

Desenvolvimento de uma aplicação de download e configuração e estudo de uma rede

Relatório do 2º trabalho laboratorial



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Grupo:

Rui Emanuel Cabral de Almeida Quaresma up201503005

Tiago Duarte Carvalho up201504461

João Dias Conde Azevedo up201503256

Faculdade de Engenharia da Universidade do Porto

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

22 de dezembro de 2017

ÍNDICE

1 Sumário	3
2 Introdução.....	3
3 Parte 1 – Aplicação de Download	4
3.1 Arquitetura	4
3.2 Resultados de um download bem sucedido	5
4 Parte 2 – Configuração da rede e análise	6
4.1 Experiência 1 – Configuração de um IP de rede	6
4.2 Experiência 2 – Implementação de duas LANs virtuais no switch	7
4.3 Experiência 3 - Configurar um Router em Linux	7
4.4 Experiência 4 – Configurar um Router comercial e implementar o NAT	8
4.5 Experiência 5 – DNS	8
4.6 Experiência 6 – Ligações TCP	8
5 Conclusão	9
6 Bibliografia.....	9
7 ANEXOS.....	10
7.1 Código da aplicação	10
7.1.1 URL.h	10
7.1.2 URL.C.....	11
7.1.3 FTP.H.....	15
7.1.5 MAIN.C	21
7.2 Comandos de configuração	25
Experiência 1 – Configuração de um ip de rede.....	25
Experiência 2 – implementação de duas vlans virtuais no switch	25
Experiência 3 – Configuração de um router em linux	26
Experiência 4 – configuração de um router comercial e implementação do NAT	26

1 SUMÁRIO

Este trabalho, desenvolvido no contexto da unidade curricular de Redes de Computadores do Mestrado Integrado de Engenharia Informática, tem como objetivo o desenvolvimento de uma aplicação de download segundo o protocolo FTP e a configuração de uma rede.

O trabalho foi concluído com sucesso, dado que é possível fazer download de ficheiros de diferentes tamanhos usando a aplicação desenvolvida, depois de configurada a rede que permitia o acesso à Internet.

2 INTRODUÇÃO

O objetivo deste segundo projeto de Redes de Computadores era permitir aos estudantes ter um contacto físico com as diversas componentes necessárias à configuração de uma rede. Isto foi alcançado nas várias aulas laboratoriais, sendo que estas se dividiram em 6 experiências.

Ao mesmo tempo era pretendido que fosse desenvolvida uma aplicação que através de uma ligação TCP conseguisse fazer a ligação entre um cliente e um servidor (servidor FTP). Esta ligação é realizada a partir de sockets.

Este relatório subdivide-se nas seguintes secções:

- **Introdução**, onde é feita uma breve introdução sobre o objetivo do projeto;
- **Parte 1 – Aplicação de Download**, em que é enunciada a arquitetura da mesma, sendo também apresentados resultados de um download bem sucedido e a sua análise;
- **Parte 2 – Configuração da rede e análise**, em que para cada experiência é apresentada a sua arquitetura, objetivos experimentais, comandos de configuração principais e análise dos logs feitos durante a realização da experiência;
- **Conclusão**, síntese das secções apresentadas e uma breve reflexão sobre os objetivos académicos alcançados;
- **Bibliografia**, sites consultados pelo grupo para a realização do projeto;
- **Anexos**, contendo o código da aplicação de download, comandos de configuração e os logs feitos.

3 PARTE 1 – APLICAÇÃO DE DOWNLOAD

A primeira parte deste projeto consistia no desenvolvimento de uma aplicação que fosse capaz de fazer download de um ficheiro a partir de um servidor, como por exemplo, `ftp://ftp.up.pt/...`

Esta foi desenvolvida na linguagem de programação C em ambiente Linux.

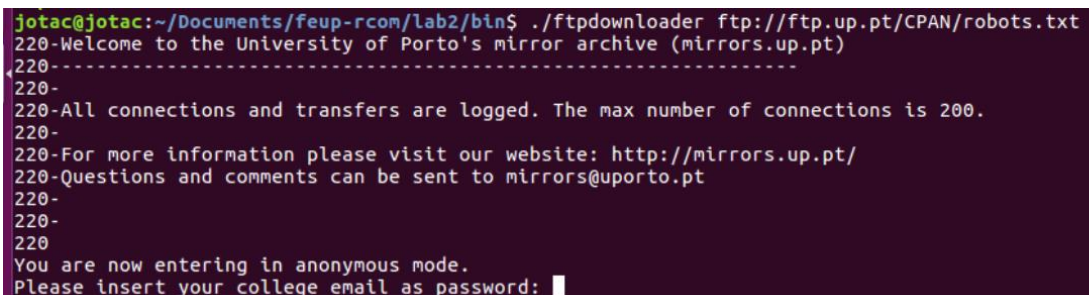
Para tal, foi necessário consultar o documento RFC959 de forma a compreender melhor como implementar o protocolo FTP. Foi também necessário consultar o documento RFC1738 para compreender o uso de URL's e a sua sintaxe.

3.1 Arquitetura

A aplicação foi dividida em duas camadas: uma camada destinada ao cliente FTP e outra destinada ao processamento do URL. Estas são compostas por estruturas responsáveis por guardar os dados que são precisos para a execução das respetivas funções.

A única ligação entre a camada FTP e a camada URL é a que ocorre quando o cliente FTP usa os atributos do URL, formado na sua camada, para realizar as ações de comunicação necessárias e a respetiva transferência do ficheiro.

De forma a iniciar a aplicação, é necessário aceder à linha de comandos e fornecer um link para o download do ficheiro. De seguida, por exemplo no modo anónimo, é pedida como password um e-mail válido.

A terminal window with a dark background and light-colored text. The prompt is 'jotac@jotac:~/Documents/feup-rcom/lab2/bin\$'. The command entered is './ftpdnloader ftp://ftp.up.pt/CPAN/robots.txt'. The output shows a series of '220-' responses from the server: '220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)', '220-All connections and transfers are logged. The max number of connections is 200.', '220-For more information please visit our website: http://mirrors.up.pt/', '220-Questions and comments can be sent to mirrors@uporto.pt', and '220-You are now entering in anonymous mode.'. The final prompt is 'Please insert your college email as password: ' followed by a cursor.

```
jotac@jotac:~/Documents/feup-rcom/lab2/bin$ ./ftpdnloader ftp://ftp.up.pt/CPAN/robots.txt
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-You are now entering in anonymous mode.
Please insert your college email as password: █
```

Tendo este link, a camada URL é responsável por guardar na sua estrutura os dados fornecidos (username, password, host, path e filename), retirando-os do link. É de seguida criado o atributo ip, sendo que ao atributo port será dado o valor 21, que é o número da porta de controlo do protocolo FTP.

As funções da camada URL consistem nas seguintes:

- initURL que é responsável por instanciar objeto url e alocar memória para os seus atributos;

- `parseURL` que é responsável pelo processamento do link recebido e guardar a informação lá contida nos atributos do objeto url;

- `getIpByHost` que é responsável por obter o IP a partir do hostname recebido. Utiliza a função `gethostbyname` que retorna uma estrutura `hostnet`, que por sua vez é usada na função `inet_ntoa` depois de aplicado um cast para uma estrutura do tipo `in_addr`. Por fim o que é retornado é um `char*` que representa o ip.

A estrutura FTP apenas contém um descritor de ficheiro para o socket de dados e outro para o socket de controlo.

Depois de processado o URL, começa-se por ligar o cliente ao servidor FTP. Isto é feito através de um socket TCP. Segundo o que está estipulado no protocolo FTP foi estabelecida uma ordem pela qual os comandos seriam enviados. A ordem foi a seguinte:

- `USER user`, em que é enviado o username (feito na função `ftpLogin`);
- `PASS password`, em que é enviada a password (feito na função `ftpLogin`);
- `PASV`, em que se passa para modo passivo o que permite uma comunicação entre o servidor e o cliente FTP de forma mútua. É de seguida feita uma nova conexão do socket sendo que a porta é processada com informação recebida do servidor que será guardada no descritor de dados do cliente (feito na função `ftpPasv`);
- `CWD path`, em que é enviado o diretório onde está guardado o ficheiro (feito na função `ftpCWD`);
- `RETR filename`, em que é feito um pedido ao servidor para que envie o ficheiro para fazer download (feito na função `ftpRetr`).

No fim o envio destes comandos é chamada a função `ftpDownload` para que se inicie o download do ficheiro pretendido.

3.2 Resultados de um download bem sucedido

Foram realizados testes com ficheiros de tamanhos diferentes de forma a confirmar o bom funcionamento da aplicação.

Estes foram realizados usando o servidor FTP da UP com ficheiros presentes em `ftp://ftp.up.pt/CPAN/...`

Apresentamos a seguir um print de um download bem sucedido de um ficheiro chamado `robots.txt` e a confirmação do mesmo através da execução do comando `ls`.

```

220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
You are now entering in anonymous mode.
Please insert your college email as password: up201503256@fe.up.pt
Bytes send: 16
Info: USER anonymous

331 Please specify the password.
Bytes send: 33
Info: PASS ♦[M♦up201503256@fe.up.pt

230 Login successful.
Bytes send: 6
Info: PASV

227 Entering Passive Mode (193,137,29,15,221,237).
IP: 193.137.29.15
PORT: 56813
Bytes send: 11
Info: CWD CPAN/

250-The Comprehensive Perl Archive Network (http://www.cpan.org/)
250-master site has been from the very beginning (1995) hosted at FUNET,
250-the Finnish University NETwork.
250-
250-
250 Directory successfully changed.
Bytes send: 17
Info: RETR robots.txt

150 Opening BINARY mode data connection for robots.txt (78 bytes).
226 Transfer complete.
Bytes send: 6
Info: QUIT

jotac@jotac:~/Documents/feup-rcom/lab2/bin$ ls
ftpdnloader robots.txt

```

4 PARTE 2 – CONFIGURAÇÃO DA REDE E ANÁLISE

4.1 Experiência 1 – Configuração de um IP de rede

Esta experiência consistiu na configuração de IPs em máquinas diferentes permitindo que comunicassem entre si. Primeiro foi necessário configurar os IPs das portas eth0 das duas máquinas e a criação das rotas necessárias. De seguida foi enviado o sinal *ping* de uma máquina para a outra, de forma a confirmar a ligação entre as duas.

Para o primeiro passo (configuração dos IPs) foi usado o comando `ifconfig` com o IP de cada máquina. Para a criação das rotas foi utilizado o comando `route`. Depois disso foi usado o comando `ping` para enviar o sinal “ping” de uma máquina para a outra.

Usando o comando `arp -a` e `route -n` foi possível confirmar que a tabela que continha os pedidos ARP e a tabela que continha as rotas estavam de acordo com o pretendido.

4.2 Experiência 2 – Implementação de duas LANs virtuais no switch

Esta experiência consistiu na criação de duas LANs virtuais no switch. A primeira constituída pelas máquinas 1 e 4, enquanto que a outra constituída apenas pela máquina 2. Isto faz com que as máquinas 1 e 4 fiquem numa sub-rede diferente da da máquina 2.

Para configurar o switch foi necessário aceder à consola de configuração. De seguida foi executado o comando `vlan y0` e `vlan y1`, em que `y0` e `y1` representavam o número da vlan. Depois de configuradas as VLANs foi necessário adicionar as portas do switch às respectivas VLANs. Os comandos usados foram `interface fastethernet 0/x`, em que `x` é o identificador da porta do switch, `switchport mode access` e `switchport access VLAN z`, em que `z` é o identificador da VLAN criada.

```

*Mar  2 04:29:17.516: %SYS-5-CONFIG_I: Configured from console by
console
tux-sw3#show vlan

VLAN Name                Status    Ports
-----
1    default                active    Gi0/1, Gi0/2
30   VLAN0030                active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                           Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                           Fa0/9, Fa0/10, Fa0/11, Fa0/12
31   VLAN0031                active    Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                           Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                           Fa0/21, Fa0/22, Fa0/23, Fa0/24
1002 fddi-default            act/unsup
1003 token-ring-default    act/unsup
1004 fddinet-default        act/unsup
1005 trnet-default         act/unsup

VLAN Type  SAID      MTU   Parent RingNo BridgeNo Stp  BrdgMode Transl Trans2
-----
1    enet     100001    1500  -      -      -      -      0      0
30   enet     100030    1500  -      -      -      -      0      0
31   enet     100031    1500  -      -      -      -      0      0
1002 fddi     101002    1500  -      -      -      -      0      0
1003 tr      101003    1500  -      -      -      -      0      0
1004 fdnet   101004    1500  -      -      -      -      0      0

VLAN Type  SAID      MTU   Parent RingNo BridgeNo Stp  BrdgMode Transl Trans2
-----
1005 trnet  101005    1500  -      -      -      -      0      0

Remote SPAN VLANs
-----

```

4.3 Experiência 3 - Configurar um Router em Linux

Esta experiência consistiu na configuração da máquina 4 para servir de router entre as duas sub-redes previamente criadas na experiência 2. Para isso foi feita a configuração da interface `eth1` da máquina 4 com um IP da mesma gama do IP da máquina 2, adicionando de seguida esta interface à sub-rede da máquina 2. Depois disto foi adicionada uma rota à máquina 1 para a máquina 4 e outra à máquina 2 para a máquina 4. Desta forma passou a poder fazer-se ping da máquina 2 a partir da máquina 1, sendo este realizado através da máquina 4.

4.4 Experiência 4 – Configurar um Router comercial e implementar o NAT

Esta experiência consistiu na configuração de um router comercial com NAT implementado.

O NAT permitia possibilitar a comunicação entre as máquinas da rede criada com redes externas. Isto era necessário dado que os IPs atuais eram privados não sendo por isso reconhecidos fora da rede. O NAT serve para reescrever os IPs da rede interna para que possam aceder a redes externas.

Para a configuração do router foi necessário configurar a interface interna no processo de NAT.

4.5 Experiência 5 – DNS

Esta experiência consistiu na configuração do DNS de forma a poder aceder à Internet através da rede criada.

Isto foi feito indo a todas as máquinas da rede interna e editar o ficheiro `resolv.conf` acrescentando “nameserver 172.16.2.1” que representa o IP do servidor a aceder.

4.6 Experiência 6 – Ligações TCP

Esta experiência consistiu na compilação e execução da aplicação de download desenvolvida na primeira parte deste projeto.

Para o teste da aplicação foi usado um servidor FTP da UP e executado o download de um ficheiro presente em `ftp://ftp.up.pt/CPAN/...`

Verificou-se que o download foi realizado com sucesso o que veio confirmar a correta configuração da rede.

5 CONCLUSÃO

Cremos que o desenvolvimento de uma aplicação de download, bem como a configuração de uma rede foram conseguidos com sucesso.

Foi também alcançada uma aprendizagem dos diversos conceitos necessários à realização deste projeto.

A nível da configuração da rede, conseguimos compreender quais os componentes físicos necessários bem como a ligação entre eles. Tendo também ganho noção das ligações lógicas essenciais à correta comunicação entre componentes.

A nível da aplicação de download, conseguimos perceber melhor o funcionamento do protocolo FTP e sockets, bem como a syntax do URL.

6 BIBLIOGRAFIA

Manuel Ricardo Lab 2 – Computer Networks. [link](#)

J. Postel, J. Reinolds File Transfer Protocol [link](#)

T. Berners-Lee Uniform Resource Locators [link](#)

7 ANEXOS

7.1 Código da aplicação

7.1.1 URL.H

```
1  #pragma once
2  #include <string.h>
3  #include <netdb.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <regex.h>
7  #include <errno.h>
8
9  #include <sys/types.h>
10 #include <sys/socket.h>
11
12 #include <arpa/inet.h>
13
14 #include <netinet/in.h>
15
16 typedef char url_content[256];
17
18 typedef struct URL {
19     url_content user; // string to user
20     url_content password; // string to password
21     url_content host; // string to host
22     url_content ip; // string to IP
23     url_content path; // string to path
24     url_content filename; // string to filename
25     int port; // integer to port
26 } url;
27
28 void initURL(url* url);
29 int parseURL(url* url, const char* str); // Parse a string with the url to create the URL structure
30 int getIpByHost(url* url); // gets an IP by host name
31
32 char* processElementUntilChar(char* str, char chr);
```

7.1.2 URL.C

```

1  #include "URL.h"
2
3  void initURL(url* url) {
4      memset(url->user, 0, sizeof(url_content));
5      memset(url->password, 0, sizeof(url_content));
6      memset(url->host, 0, sizeof(url_content));
7      memset(url->path, 0, sizeof(url_content));
8      memset(url->filename, 0, sizeof(url_content));
9      url->port = 21;
10 }
11
12 const char* regExpression =
13     "ftp://([([A-Za-z0-9]*:([A-Za-z0-9])*@)*([A-Za-z0-9.~-])+/(([A-Za-z0-9/~._-])+";
14
15 const char* regExprAnony = "ftp://([A-Za-z0-9.~-])+/(([A-Za-z0-9/~._-])+";
16
17 int parseURL(url* url, const char* urlStr) {
18     char* tempURL, *element, *activeExpression;
19     regex_t* regex;
20     size_t nmatch = strlen(urlStr);
21     regmatch_t pmatch[nmatch];
22     int userPassMode;
23
24     element = (char*) malloc(strlen(urlStr));
25     tempURL = (char*) malloc(strlen(urlStr));
26
27     memcpy(tempURL, urlStr, strlen(urlStr));
28
29     if (tempURL[6] == '[') {
30         userPassMode = 1;
31         activeExpression = (char*) regExpression;
32     } else {

```

```

33     userPassMode = 0;
34     activeExpression = (char*) regExprAnony;
35 }
36
37     regex = (regex_t*) malloc(sizeof(regex_t));
38
39     int reti;
40     if ((reti = regcomp(regex, activeExpression, REG_EXTENDED)) != 0) {
41         perror("URL format is wrong.");
42         return 1;
43     }
44
45     if ((reti = regexec(regex, tempURL, nmatch, pmatch, REG_EXTENDED)) != 0) {
46         perror("URL could not execute.");
47         return 1;
48     }
49
50     free(regex);
51
52     // removing ftp:// from string
53     strcpy(tempURL, tempURL + 6);
54
55     if (userPassMode) {
56         //removing [ from string
57         strcpy(tempURL, tempURL + 1);
58
59         // saving username
60         strcpy(element, processElementUntilChar(tempURL, ':'));
61         memcpy(url->user, element, strlen(element));
62
63         //saving password

```

```

64     strcpy(element, processElementUntilChar(tempURL, '@'));
65     memcpy(url->password, element, strlen(element));
66     strcpy(tempURL, tempURL + 1);
67 }
68
69 //saving host
70 strcpy(element, processElementUntilChar(tempURL, '/'));
71 memcpy(url->host, element, strlen(element));
72
73 //saving url path
74 char* path = (char*) malloc(strlen(tempURL));
75 int startPath = 1;
76 while (strchr(tempURL, '/')) {
77     element = processElementUntilChar(tempURL, '/');
78
79     if (startPath) {
80         startPath = 0;
81         strcpy(path, element);
82     } else {
83         strcat(path, element);
84     }
85
86     strcat(path, "/");
87 }
88 strcpy(url->path, path);
89
90 // saving filename
91 strcpy(url->filename, tempURL);
92
93 free(tempURL);
94 free(element);

```

```

95
96 ▾ /*printf("\n%s\n%s\n%s\n%s\n%s\n", url->user, url->password, url->host,
97     url->path, url->filename);*/
98
99     return 0;
100 }
101
102 ▾ int getIpByHost(url* url) {
103     struct hostent* h;
104
105 ▾ if ((h = gethostbyname(url->host)) == NULL) {
106     perror("gethostbyname");
107     return 1;
108 }
109
110 ▾ // printf("Host name : %s\n", h->h_name);
111 // printf("IP Address : %s\n", inet_ntoa(*(struct in_addr *) h->h_addr));
112
113     char* ip = inet_ntoa(*(struct in_addr *) h->h_addr);
114     strcpy(url->ip, ip);
115
116     return 0;
117 }
118
119 ▾ char* processElementUntilChar(char* str, char chr) {
120     // using temporary string to process substrings
121     char* tempStr = (char*) malloc(strlen(str));
122
123     // calculating length to copy element
124     int index = strlen(str) - strlen(strcpy(tempStr, strchr(str, chr)));
125
126     tempStr[index] = '\0'; // termination char in the end of string
127
127     strncpy(tempStr, str, index);
128     strcpy(str, str + strlen(tempStr) + 1);
129
130     return tempStr;
131 }
132

```

7.1.3 FTP.H

```
1  #include <string.h>
2  #include <netdb.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <regex.h>
6  #include <errno.h>
7  #include <unistd.h>
8
9  #include <sys/types.h>
10 #include <sys/socket.h>
11
12 #include <arpa/inet.h>
13
14 #include <netinet/in.h>
15
16 typedef struct FTP
17 {
18     int control_socket_fd; // file descriptor to control socket
19     int data_socket_fd; // file descriptor to data socket
20 } ftp;
21
22 int ftpConnect(ftp* ftp, const char* ip, int port);
23 int ftpLogin(ftp* ftp, const char* user, const char* password);
24 int ftpCWD(ftp* ftp, const char* path);
25 int ftpPasv(ftp* ftp);
26 int ftpRetr(ftp* ftp, const char* filename);
27 int ftpDownload(ftp* ftp, const char* filename);
28 int ftpDisconnect(ftp* ftp);
29
30 int ftpSend(ftp* ftp, const char* str, size_t size);
31 int ftpRead(ftp* ftp, char* str, size_t size);
32
```

7.1.4 FTP.C

```

1  #include "FTP.h"
2
3  static int connectSocket(const char* ip, int port) {
4      int sockfd;
5      struct sockaddr_in server_addr;
6
7      // server address handling
8      bzero((char*) &server_addr, sizeof(server_addr));
9      server_addr.sin_family = AF_INET;
10     server_addr.sin_addr.s_addr = inet_addr(ip); /*32 bit Internet address network byte ordered*/
11     server_addr.sin_port = htons(port); /*server TCP port must be network byte ordered */
12
13     // open an TCP socket
14     sockfd = socket(AF_INET, SOCK_STREAM, 0);
15
16     // connect to the server
17     connect(sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr));
18
19     return sockfd;
20 }
21
22 int ftpConnect(ftp* ftp, const char* ip, int port) {
23     int sockfd;
24     char rd[1024];
25
26     sockfd = connectSocket(ip, port);
27
28     ftp->control_socket_fd = sockfd;
29     ftp->data_socket_fd = 0;
30
31     ftpRead(ftp, rd, sizeof(rd));
32

```



```

33     return 0;
34 }
35
36 int ftpLogin(ftp* ftp, const char* user, const char* password) {
37     char sd[1024];
38
39     // username
40     sprintf(sd, "USER %s\r\n", user);
41
42     ftpSend(ftp, sd, strlen(sd));
43
44     ftpRead(ftp, sd, sizeof(sd));
45
46     // cleaning buffer
47     memset(sd, 0, sizeof(sd));
48
49     // password
50     sprintf(sd, "PASS %s\r\n", password);
51     ftpSend(ftp, sd, strlen(sd));
52
53     ftpRead(ftp, sd, sizeof(sd));
54
55     return 0;
56 }
57
58 int ftpCWD(ftp* ftp, const char* path) {
59     char cwd[1024];
60
61     sprintf(cwd, "CWD %s\r\n", path);
62     ftpSend(ftp, cwd, strlen(cwd));
63
64     ftpRead(ftp, cwd, sizeof(cwd));
65
66     return 0;
67 }
68
69 int ftpPasv(ftp* ftp) {
70     char pasv[1024] = "PASV\r\n";
71
72     ftpSend(ftp, pasv, strlen(pasv));
73
74     ftpRead(ftp, pasv, sizeof(pasv));
75
76     // starting process information
77     int ipPart1, ipPart2, ipPart3, ipPart4;
78     int port1, port2;
79     sscanf(pasv, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)", &ipPart1, &ipPart2, &ipPart3, &ipPart4, &port1, &port2);
80
81     // cleaning buffer
82     memset(pasv, 0, sizeof(pasv));
83
84     // forming ip
85     sprintf(pasv, "%d.%d.%d.%d", ipPart1, ipPart2, ipPart3, ipPart4);
86
87     // calculating new port
88     int portResult = port1 * 256 + port2;
89
90     printf("IP: %s\n", pasv);
91     printf("PORT: %d\n", portResult);
92
93     ftp->data_socket_fd = connectSocket(pasv, portResult);
94 }

```

```
95     return 0;
96 }
97
98 int ftpRetr(ftp* ftp, const char* filename) {
99     char retr[1024];
100
101     sprintf(retr, "RETR %s\r\n", filename);
102     if (ftpSend(ftp, retr, strlen(retr))) {
103         printf("ERROR: Cannot send filename.\n");
104         return 1;
105     }
106
107     if (ftpRead(ftp, retr, sizeof(retr))) {
108         printf("ERROR: None information received.\n");
109         return 1;
110     }
111
112     return 0;
113 }
114
115 int ftpDownload(ftp* ftp, const char* filename) {
116     FILE* file;
117     int bytes;
118
119     if (!(file = fopen(filename, "w"))) {
120         printf("ERROR: Cannot open file.\n");
121         return 1;
122     }
123
124     char buf[1024];
125     while ((bytes = read(ftp->data_socket_fd, buf, sizeof(buf)))) {
126         if (bytes < 0) {
```

```
127     printf("ERROR: Nothing was received from data socket fd.\n");
128     return 1;
129 }
130
131 if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
132     printf("ERROR: Cannot write data in file.\n");
133     return 1;
134 }
135 }
136
137 fclose(file);
138 close(ftp->data_socket_fd);
139
140 return 0;
141 }
142
143 int ftpDisconnect(ftp* ftp) {
144     char disc[1024];
145
146     if (ftpRead(ftp, disc, sizeof(disc))) {
147         printf("ERROR: Cannot disconnect account.\n");
148         return 1;
149     }
150
151     sprintf(disc, "QUIT\r\n");
152     if (ftpSend(ftp, disc, strlen(disc))) {
153         printf("ERROR: Cannot send QUIT command.\n");
154         return 1;
155     }
156
157     if (ftp->control_socket_fd)
```

```
158     close(ftp->control_socket_fd);
159
160     return 0;
161 }
162
163 int ftpSend(ftp* ftp, const char* str, size_t size) {
164     int bytes;
165
166     bytes = write(ftp->control_socket_fd, str, size);
167
168     printf("Bytes send: %d\nInfo: %s\n", bytes, str);
169
170     return 0;
171 }
172
173 int ftpRead(ftp* ftp, char* str, size_t size) {
174     FILE* fp = fdopen(ftp->control_socket_fd, "r");
175
176     do {
177         memset(str, 0, size);
178         str = fgets(str, size, fp);
179         printf("%s", str);
180     } while (!('1' <= str[0] && str[0] <= '5') || str[3] != ' ');
181
182     return 0;
183 }
184
```

7.1.5 MAIN.C

```
1  #include "URL.h"
2  #include "FTP.h"
3
4  void printUsage(char* argv0) {
5      printf("\nLogin Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv0);
6      printf("Anonymous Usage: %s ftp://<host>/<url-path>\n\n", argv0);
7  }
8
9  int main(int argc, char** argv){
10
11      if(argc != 2){
12          printf("Wrong number of arguments\n");
13          printUsage(*argv);
14          return 1;
15      }
16
17      //url process
18      url url;
19      initURL(&url);
20
21      if(parseURL(&url,argv[1]) != 0){
22          perror("Error parsing URL\n");
23          return 1;
24      }
25
26      //get hostname IP
27      getIpByHost(&url);
28
29      //process FTP client
30      ftp ftp;
31
```

```

32
33 //connect
34 if(ftpConnect(&ftp, url.ip, url.port) != 0){
35     perror("Error connecting to FTP\n");
36     return 1;
37 }
38
39 //user and pw
40 const char* user = strlen(url.user) ? url.user : "anonymous";
41 char* password;
42 if (strlen(url.password)) {
43     password = url.password;
44 } else {
45     char buf[100];
46     printf("You are now entering in anonymous mode.\n");
47     printf("Please insert your college email as password: ");
48     while (strlen(fgets(buf, 100, stdin)) < 14)
49         printf("\nIncorrect input, please try again: ");
50     password = (char*) malloc(strlen(buf));
51     strncat(password, buf, strlen(buf) - 1);
52 }
53
54
55 //login
56 if(ftpLogin(&ftp,user,password) != 0){
57     perror("Error logging in ftp\n");
58     return 1;
59 }
60
61 //enter passive mode
62 if(ftpPasv(&ftp) != 0){
63     perror("Error entering PASV mode\n");

```

```
64     return 1;
65 }
66
67 //change directory
68 if(ftpCWD(&ftp,url.path) != 0){
69     perror("Error changing directory\n");
70     return 1;
71 }
72
73 //begin transmitting
74 if(ftpRetr(&ftp,url.filename)!= 0){
75     perror("Error retrieving file\n");
76     return 1;
77 }
78
79 //download filename
80 if(ftpDownload(&ftp,url.filename)!= 0){
81     perror("Error downloading file\n");
82     return 1;
83 }
84
85 //disconnect from FTP
86 if(ftpDisconnect(&ftp)!= 0){
87     perror("Error disconnecting from ftp\n");
88     return 1;
89 }
90
91 return 0;
92 }
93
```


7.2 Comandos de configuração

EXPERIÊNCIA 1 – CONFIGURAÇÃO DE UM IP DE REDE

```
ifconfig eth0 down
ifconfig eth0 up
ifconfig eth0 172.16.50.1/24
ifconfig eth0 down
ifconfig eth0 up
ifconfig eth0 172.16.51.1/24
ifconfig eth0 up
ifconfig eth0 down
ifconfig eth0 172.16.50.254/24
```

EXPERIÊNCIA 2 – IMPLEMENTAÇÃO DE DUAS VLANS VIRTUAIS NO SWITCH

```
configure terminal
vlan 50
end
configure terminal
vlan 51
end
--Ports da VLAN 50
configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan 50
end
configure terminal
interface fastethernet 0/3
switchport mode access
switchport access vlan 50
end
--Ports da VLAN 51
configure terminal
interface fastethernet 0/2
switchport mode access
switchport access vlan 51
end
configure terminal
interface fastethernet 0/4
switchport mode access
switchport access vlan 51
end
configure terminal
interface fastethernet 0/5
```

```
switchport mode access
switchport access vlan 51
end
```

EXPERIÊNCIA 3 – CONFIGURAÇÃO DE UM ROUTER EM LINUX

TUX 1:

```
route add -net 172.16.51.0/24 gw 172.16.50.254
route add -net default gw 172.16.50.254
```

TUX 2:

```
route add -net 172.16.50.0/24 gw 172.16.51.253
```

EXPERIÊNCIA 4 – CONFIGURAÇÃO DE UM ROUTER COMERCIAL E IMPLEMENTAÇÃO DO NAT

conf t

```
interface fastethernet 0/0
ip address 172.16.51.254 255.255.255.0
no shutdown
ip nat inside
exit
interface fastethernet 0/1
ip address 172.16.1.29 255.255.255.0
no shutdown
ip nat outside
exit
ip nat pool ovrld 172.16.1.29 172.16.1.29 prefix 24
ip nat inside source list 1 pool ovrld overload
access-list 1 permit 172.16.10.0 0.0.0.255
access-list 1 permit 172.16.11.0 0.0.0.255
ip route 0.0.0.0 0.0.0.0 172.16.2.254
ip route 172.16.50.0 255.255.255.0 172.16.51.253
end
```

Nota: Experiências realizadas na sala I320

7.3 Logs tirados

The screenshot shows the Wireshark 1.12.1 interface. The main pane displays a list of packets captured on the interface eth0 and eth1. The packets are filtered by the expression 'Expression... Clear Apply Save'. The bottom pane shows the packet details for the selected packet (No. 9139), displaying the raw data and its hexadecimal representation.

No.	Time	Source	Destination	Protocol	Length	Info
9112	486.2399910	172.16.31.253	93.184.220.29	TCP	66	[TCP Keep-Alive] 41556-80 [ACK] Seq=431 Ack=789 Win=32128 Len=0 TSval=178442776 TSecr=2990848099
9113	486.2839010	93.184.220.29	172.16.31.253	TCP	66	[TCP Keep-Alive ACK] 80-41556 [ACK] Seq=789 Ack=432 Win=145920 Len=0 TSval=2990850611 TSecr=1784
9114	486.2840580	93.184.220.29	172.16.31.253	TCP	66	[TCP Keep-Alive ACK] 80-41555 [ACK] Seq=789 Ack=432 Win=145920 Len=0 TSval=2990850611 TSecr=1784
9115	487.1993550	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) request id=0x5729, seq=1/256, ttl=64 (reply in 9116)
9116	487.1996970	172.16.30.1	172.16.30.1	ICMP	98	Echo (ping) reply id=0x5729, seq=1/256, ttl=63 (request in 9115)
9117	487.1994000	Kye_25:26:0a	Broadcast	ARP	42	Who has 172.16.31.1? Tell 172.16.31.253
9118	487.1995170	Hewlett-61:30:63	Kye_25:26:0a	ARP	60	172.16.31.1 is at 00:21:5a:81:30:63
9119	487.1995320	172.16.31.253	172.16.31.1	ICMP	98	Echo (ping) request id=0x5729, seq=1/256, ttl=63 (reply in 9120)
9120	487.1996860	172.16.31.1	172.16.31.253	ICMP	98	Echo (ping) reply id=0x5729, seq=1/256, ttl=64 (request in 9119)
9121	488.0471720	172.16.31.253	172.16.1.1	DNS	87	Standard query 0xa14f PTR 125.253.30.192.in-addr.arpa
9122	488.0481290	172.16.1.1	172.16.31.253	DNS	245	Standard query response 0xa14f PTR lb-192-30-253-125-iad.github.com
9123	488.1420690	Cisco 3a:fa:91	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8011
9124	488.1599830	172.16.31.253	151.101.16.133	TCP	66	[TCP Keep-Alive] 53978-443 [ACK] Seq=745 Ack=905598 Win=422912 Len=0 TSval=178443256 TSecr=12945
9125	488.1983960	172.16.31.253	172.16.31.1	ICMP	98	Echo (ping) request id=0x5729, seq=2/512, ttl=63 (reply in 9126)
9126	488.1985370	172.16.31.1	172.16.31.253	ICMP	98	Echo (ping) reply id=0x5729, seq=2/512, ttl=64 (request in 9125)
9127	488.2045660	151.101.16.133	172.16.31.253	TCP	66	[TCP Keep-Alive ACK] 443-53978 [ACK] Seq=905598 Ack=746 Win=31232 Len=0 TSval=1294523190 TSecr=
9128	488.0448810	Cisco 3a:fa:86	Spanning-tree-(for-br	STP	60	Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
9129	488.0471400	172.16.30.1	172.16.1.1	DNS	87	Standard query 0xa14f PTR 125.253.30.192.in-addr.arpa
9130	488.0481620	172.16.1.1	172.16.30.1	DNS	245	Standard query response 0xa14f PTR lb-192-30-253-125-iad.github.com
9131	488.1983810	172.16.30.1	172.16.31.1	ICMP	98	Echo (ping) request id=0x5729, seq=2/512, ttl=64 (reply in 9132)
9132	488.1985500	172.16.31.1	172.16.30.1	ICMP	98	Echo (ping) reply id=0x5729, seq=2/512, ttl=63 (request in 9131)
9133	488.3481020	172.16.31.253	172.16.1.1	DNS	87	Standard query 0xf9b6 PTR 124.253.30.192.in-addr.arpa
9134	488.3489080	172.16.1.1	172.16.31.253	DNS	245	Standard query response 0xf9b6 PTR lb-192-30-253-124-iad.github.com
9135	488.3490790	172.16.31.253	172.16.1.1	DNS	87	Standard query 0x7624 PTR 174.214.58.216.in-addr.arpa
9136	488.3498710	172.16.1.1	172.16.31.253	DNS	303	Standard query response 0x7624 PTR mad01s26-in-f14.1e100.net PTR mad01s26-in-f174.1e100.net
9137	488.3500450	172.16.31.253	172.16.1.1	DNS	86	Standard query 0x3963 PTR 36.205.58.216.in-addr.arpa
9138	488.3508460	172.16.1.1	172.16.31.253	DNS	300	Standard query response 0x3963 PTR ml04s24-in-f4.1e100.net PTR ml04s24-in-f36.1e100.net
9139	488.3509940	172.16.31.253	172.16.1.1	DNS	87	Standard query 0x8f1 PTR 162.210.58.216.in-addr.arpa

The bottom pane shows the packet details for the selected packet (No. 9139), displaying the raw data and its hexadecimal representation.

```

0000  68 ef bd d6 aa b8 00 c0 df 25 26 0a 08 00 45 00  h.....%...E.
0010  00 34 77 51 40 00 40 06 4f 7b ac 10 1f fd 97 65  .4wQ8.@O(.....e
0020  10 85 d2 83 01 bb 6c 4a b1 be b7 3a 3e c7 80 10  .....l).....>...
0030  01 dd 74 1e 00 00 01 01 08 0a 0a a0 f7 40 4d 26  ..t.....@%$
0040  f4 cf
  
```

The status bar at the bottom shows the capture progress on eth0, with 9263 packets displayed (100.0%). The profile is set to Default.