Redes de Computadores (RCOM)
Turma 5, Grupo 2
2020/2021

Rede de Computadores
2º Trabalho Laboratorial

João Carlos Machado Rocha Pires (up201806079)
João Luís Azevedo Ferreira (up201806716)

18-12-2020

Contents

1. Summary

This report was written to support the second laboratory work made in the Computer Networks curricular unit. The subject of this work was Computer Networks. That said, our job was completed with success by programming a download application capable of transfer files through File Transfer Protocol (FTP) and configuring a network using CISCO equipment.

2. Introduction

In the next pages, we'll discuss all the parts of this work, starting by the Download Application and then the Network Configuration and Analysis. In this last part, we'll focus on each of the experiments made, from the first to the sixth, analysing, for each one, the network architecture, the experiment objectives, the main configuration command and the logs captured that were relevant for the learning objectives.

3. Part 1 - Download Application

This first part was the development of a download application in the C programming language. This application receives the URL of a file as an argument and then proceeds with the download of the same.

The URL is of the following type **ftp://[<user>:<password>@]<host>/<url-path>**, which means the username and the passwords are optional, and the URL syntax is adopted as described in RFC1738.

The FTP application protocol was implemented as described in RFC959.

3.1 Download Application Architecture

Our download application starts by analysing the URL provided as an argument and parsing its elements (username and password, if provided, host and file path) using a state machine. By doing this, we also make sure we're checking whether the URL contains both username and password, just one of them or even none.

Then, we call the function *getHostInfo*, which receives the host name and puts all the information related to in the struct hostent *h, which is a global variable. Then, the function *establishConnection* is called, which assures three steps: server address handling, opening of a TCP socket and connection to the server.

Next, depending of the situation, the application behaves following this scenarios:

A) If the URL contains both username and password, the application saves them to be sent in the commands "user <username>" and "pass <password>" through the socket.

B) If the URL contains only the username, the application starts by asking the password and then behaves like the scenario A).

C) If the URL contains only the password, the application starts by asking the username and then behaves like the scenario A).

D) If the URL doesn't contain neither username and password, the application asks if the user wants to proceed using the anonymous mode or if the user wants to set a username and password. The, the behavior is the same as in scenario A).

The next step of our application is to use another state machine to check the information that receives and send the commands only when the right time comes. The sequence of commands is "user <username>", "pass <password>" and "pasv".

Then, before sending the command "retr <file-path>", we call the function *establishConnection2*, which is similar to the *establishConnection*, but the main difference is that it receives the server port, which is retrieved by the function *getServerPort*.

Finally, we open a FILE with the name retrieved by the function *getFileName*, which receives the file path and retrieves the name with the extension, and write to it all the bytes collected in the read process using the second socket.

We've used two .c files (download.c and functions.c) and one header (functions.h). The main process is held in download.c, while the auxiliary functions mentioned in the previous paragraphs are all in the functions (.c and .h).

3.2 Report of a successful Download

Our application works as expected. We submitted the download application to a set of experiments/tests which could assure that it behaves as it's supposed. The only thing that our application is not capable of is to download a .mp4 or any other video file.

To see some examples of a single download, there are some screenshots in the annexes.

4. Part 2 - Network Configuration and Analysis

Introduction note: As we worked in the station 3, we'll from now on mention substitute all y's by 3 (e.g. tuxy3 →tux33 and so on).

4.1 Experiment 1 - Configure an IP Network

4.1.1 Network architecture, experiment objectives, main configuration command

In terms of the architecture, the tux33.eth0 and the tux34.eth0 are connected to the switch. The serial port of tux33, s0, is connected to the RS232 →Cisco. Then, the Cisco →RS232 is connected to the switch.

The main objective is to connect tux33 and tux34 through the switch.

4.1.2 Analysis of the logs captured that are relevant for the learning objectives

1) What are the ARP packets and what are they used for?

The ARP (Address Resolution Protocol) packets are a type of message sent in order to obtain the MAC address associated to a given IP address.

2) What are the MAC and IP addresses of ARP packets and why?

There are 2 ARP packets, the request and the reply. The request contains it's own IPv4 and MAC addresses and it will "ask" who has a certain IPv4 address. The reply will then fill in the missing MAC address associating it with it's own IP.

3) What packets does the ping command generate?

Ping first generates ARP packets and then ICMP (Internet Control Message Protocol) packets.

4) What are the MAC and IP addresses of the ping packets?

Request Packet	Reply Packet
Sender IP: 172.16.30.1	Sender IP: 172.16.30.254
Sender MAC: 00:21:5a:5a:7d:b7	Sender MAC: 00:21:5a:5a:74:3e
Target IP: 172.16.30.254	Target IP: 172.16.30.1
Target MAC: 00:00:00:00:00:00	Target MAC: 00:21:5a:5a:7d:b7

(See figure 7.1 and figure 7.2)

5) How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

In Wireshark, by analysing the Type parameter in the Ethernet tree, we can check whether the Ethernet frame is ARP (0x0806) or IP (0x0800). Then, if it's an IP one, we can check the Protocol parameter in the Internet Protocol tree and see if it's a Ethernet frame ICMP (1).

6) How to determine the length of a receiving frame?

The length of a receiving frame can be seen in the Length column in Wireshark or in the Frame Length field in the details of a frame.

7) What is the loopback interface and why is it important?

The loopback interface is used to identify the device and, as the loopback address never changes, it is the best way to do it. It's also important because any traffic that a computer program sends on the loopback network is addressed to the same computer.

4.2 Experiment 2 - Implement two virtual LANs in a switch

4.2.1 Network architecture, experiment objectives, main configuration command

In this experiment, we start from the architecture used in the first one and add a connection between tux32.eth0 and the switch.

The main objective is to connect the vlan30 (tux33.eth0 and tux34.eth0) to the vlan31 (tux32.eth0) using the switch.

4.2.2 Analysis of the logs captured that are relevant for the learning objectives

1) How to configure vlan30?

To configure vlan30, we must start by opening the gtkterm at tux33 and pressing enter until we are able to start writing. Then, enter the following sequence of commands:

» en

» configure terminal (or conf t)

» vlan 30

» end (or Ctrl+C)

Then, to add the tux33.eth0 and tux34.eth0 ports to the vlan30, do the following:

» configure terminal (or conf t)

» interface fastethernet 0/P (where P is the number of the port in the switch where the cable from tux33.eth0 and tux34.eth0 are connected to, accordingly. In our case, 5 and 7)

» switchport mode access

» switchport access vlan 30

» end (or Ctrl+C)

2) How many broadcast domains are there? How can you conclude it from the logs?

There are as many broadcast domains, in this case, as the number of vlans. We can conclude that there are 2 broadcast domains, one with the tux33 and the tux34 and the other with the tux32, which is unreachable (at this moment) after doing ping -b (ping broadcast).

4.3 Experiment 3 - Configure a Router in Linux

4.3.1 Network architecture, experiment objectives, main configuration command

In this experiment, we start with all the connections used for the previous experiment and add the connection between tux34.eth1 and the switch.

The main objective of this experiment is to make tux34 work like a Linux router and then establish a connection between vlan30 and vlan31 (between tux32 and tux33).

4.3.2 Analysis of the logs captured that are relevant for the learning objectives

1) What routes are there in the tuxes? What are their meaning?

The routes in the tuxes are the following:

- For tux32, there's a route to vlan31 (172.16.31.0) using the gateway 172.16.31.1 and a route to vlan30 (172.16.30.0) using the gateway 172.16.30.254
- For tux33, there's a route to vlan30 (172.16.30.0) using the gateway 172.16.30.1 and a route to vlan31 (172.16.31.0) using the gateway 172.16.31.253
- For tux34, there's a route to vlan30 (172.16.30.0) using the gateway 172.16.30.254 and a route to vlan31 (172.16.31.0) using the gateway 172.16.31.253.

The routes indicate where the respective tux can go.

2) What information does an entry of the forwarding table contain?

An entry of the forwarding table contains information about Destination, Gateway, Genmask, Flags, Metric, Ref(erence), Use and Iface (interface).

An example can be found in the annexes.

3) What ARP messages, and associated MAC addresses, are observed and why?

As we said in the second question of experiment 1, the ARP messages are used to obtain an unknown MAC address of a certain IP. That said, in this case, the ARP messages observed are the ones that associate a MAC address to an IP when we do ping of a certain network interface.

4) What ICMP packets are observed and why?

The ICMP packets observed are ICMP Request and Reply packets. This indicates that the routes were added with success, allowing all the tuxes to reach each other.

5) What are the IP and MAC addresses associated to ICMP packets and why?

The IP and MAC addresses associated to ICMP packets are the same as the origin and destination IPs and MACs from the tuxes. That means that an ICMP packet will have an origin IP and MAC equal to the origin tux IP and MAC and the same occurs for the destination IP and MAC.

4.4 Experiment 4 - Configure a Commercial Router and Implement NAT

4.4.1 Network architecture, experiment objectives, main configuration command

In this experiment, we start from the architecture used in the third experiment and add a router to vlan1 which will be connected to the internet.

The goal of the experiment is to configure the router and implement NAT which will allow certain computers to have internet access.

4.4.2 Analysis of the logs captured that are relevant for the learning objectives

1) How to configure a static route in a commercial router?

To configure a static route you must add an entry of it to the routing table. To do so you must open GTKTerm and introduce the following commands:

»conf t

»ip route [ip route][mask][gateaway]

»exit

2) What are the paths followed by the packets in the experiments carried out and why?

The packets will follow the paths specified in the routing table if they match. Otherwise they will use the default route (0.0.0.0)

3) How to configure NAT in a commercial router? To configure NAT we first defined a NAT inside/outside interface. We then define a NAT pool and associate it with an access list that permits a defined range of IP addresses.

4) What does NAT do?

Network Address Translation maps multiple local private addresses to a public one. This method allows IP conservation and offers better security to the network.

4.5 Experiment 5 - DNS

4.5.1 Network architecture, experiment objectives, main configuration command

The goal of this experiment is to connect the network to a DNS server which will be responsible for translating domain names to a numeric IP address.

4.5.2 Analysis of the logs captured that are relevant for the learning objectives

1) How to configure the DNS service at an host?

To configure the DNS service we must modify the resolv.conf file by adding the Domain Name and it's IP address of "nameservers" for resolution.

2) What packets are exchanged by DNS and what information is transported

The host sends the desired hostname to the server asking for it's IP address. The server then replies with the corresponding IP address.

4.6 Experiment 6 - TCP connections

4.6.1 Network architecture, experiment objectives, main configuration command

The objective of the last experiment is to use the download application inside our computer network which will allow us to monitor the behavior of the TCP protocol

4.6.2 Analysis of the logs captured that are relevant for the learning objectives

1) How many TCP connections are opened by your ftp application?

The ftp application opened 2 TCP connections.

2) In what connection is transported the FTP control information?

The FTP control is transported in a TCP connection which remains opened the whole session called control connection.

3) What are the phases of a TCP connection?

Connection establishment, data transfer and connection termination.

4) How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?

The ARQ TCP mechanism uses the sliding window method where each packet is assigned a unique consecutive sequence number. The main TCP fields observed are Acknowledgement numbers which signify receipt of message, Sequence numbers which are used to keep track of how much data has been sent and window size which represents the amount of data that each packet can contain

5) How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according the TCP congestion control mechanism?

TCP uses slow start and congestion window, to achieve congestion avoidance.This strategies determine how much data can be transferred at any time. They are dictated by an additive increase/-multiplicative decrease (AIMD) approach which means the congestion window will keep growing one by one each ACK until a timeout occurs. This timeout will then reset the congestion window.

6) Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?

The appearance of a simultaneous TCP connection will result in a drop of data transfer speed since the bandwith will be split equally between both connections.

5. Conclusion

This laboratory work was concluded with success by configuring the computer network and the download application. We both worked the same and, even with the COVID-19 restrictions, we always managed to be in touch during the classes and outside them in order to do our best and to keep the element outside the presential class informed of what was happening. That said, it was the opportunity to put in practice what we've been learning in the theoretical classes and to extend our knowledge in the subject.

6. References

1. Loopback documentation

7. Annexes

7.1 Download Application

download.c

```c
#include "functions.h"

int main(int argc, char *argv[])
{

    if (argc != 2)
    {
        fprintf(stderr, "usage: invalid number of parameters\n");
        exit(1);
    }

    char *url = argv[1];

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

    int index = 0;

    char *host = malloc(50);
    char *filepath = malloc(100);
    char *user = malloc(100);
    char *password = malloc(100);

    int state = FTP;
    int type = 0; // 0 - both provided, 1 - only user, 2 - only password, 3 - none
    int passwordFound = 0;
    int userFound = 0;

    for (long int i = 0; i < strlen(url); i++) {
        switch (state) {
            case FTP:
                if ((url[i] == '/') && (url[i + 1] == '/')) {
                    state = USER;
                    i++;
                }
                break;
            case USER:
                if (url[i] == '/') {
                    strcpy(host, user);
                    state = PATH;
                    index = 0;
                }
```

```
44                  else if (url[i] == ':') {
45                      if (index != 0) userFound = 1;
46                      state = PASSWORD;
47                      index = 0;
48                  } else {
49                      user[index] = url[i];
50                      index++;
51                  }
52                  break;
53              case PASSWORD:
54                  if (url[i] == '@') {
55                      if (index != 0) passwordFound = 1;
56                      state = HOST;
57                      index = 0;
58                  } else {
59                      password[index] = url[i];
60                      index++;
61                  }
62                  break;
63              case HOST:
64                  if (url[i] == '/') {
65                      state = PATH;
66                      index = 0;
67                  } else {
68                      host[index] = url[i];
69                      index++;
70                  }
71                  break;
72              case PATH:
73                  filepath[index] = url[i];
74                  index++;
75                  break;
76              default:
77                  break;
78          }
79      }
80
81      type = passwordFound == 1 && userFound == 1 ? 0 : passwordFound == 0 && userFound
    == 1 ? 1 : passwordFound == 1 && userFound == 0 ? 2 : 3;
82
83      getHostInfo(host);
84
85      establishConnection(inet_ntoa(*((struct in_addr *)h->h_addr)));
86
87      char *messages[3];
88      messages[2] = "pasv\n";
89
90      if (type == 3) {        // case ftp://host/path
91
92
93          // ask if wants anonymous mode
94          write(STDOUT_FILENO, "Do you want to user anonymous mode? [y/N] ", 42);
95          char choice[256];
96          fgets(choice, sizeof(choice), stdin);
```

```
97          char finalchoice;
98          sscanf(choice, "%c", &finalchoice);
99
100         if (finalchoice == 'y') {
101
102             messages[0] = "user anonymous";
103             messages[1] = "pass 123456";
104
105         } else {
106             // ask password and user
107
108             write(STDOUT_FILENO, "Please provide an user: ", 24);
109             char finalUser[256];
110             fgets(finalUser, sizeof(finalUser), stdin);
111             char *finalU = malloc(strlen("user ") + strlen(finalUser) + 1);
112             strcpy(finalU, "user ");
113             strcat(finalU, finalUser);
114             messages[0] = finalU;
115
116             write(STDOUT_FILENO, "Please provide a password: ", 27);
117             char finalPassword[256];
118             fgets(finalPassword, sizeof(finalPassword), stdin);
119             char *finalP = malloc(strlen("pass ") + strlen(finalPassword) + 1);
120             strcpy(finalP, "pass ");
121             strcat(finalP, finalPassword);
122             messages[1] = finalP;
123
124         }
125
126     } else if (type == 2) { // case ftp://:password@host/path
127
128         char *finalPassword = malloc(strlen("pass ") + strlen(password) + 1);
129         strcpy(finalPassword, "pass ");
130         strcat(finalPassword, password);
131         messages[1] = finalPassword;
132
133         // ask user
134
135         write(STDOUT_FILENO, "Please provide an user: ", 24);
136         char finalUser[256];
137         fgets(finalUser, sizeof(finalUser), stdin);
138         char *finalU = malloc(strlen("user ") + strlen(finalUser) + 1);
139         strcpy(finalU, "user ");
140         strcat(finalU, finalUser);
141         messages[0] = finalU;
142
143     } else if (type == 1) { // case ftp://user:@host/path
144
145         char *finalUser = malloc(strlen("user ") + strlen(user) + 1);
146         strcpy(finalUser, "user ");
147         strcat(finalUser, user);
148         messages[0] = finalUser;
149
150         // ask password
```

```
151
152         write(STDOUT_FILENO, "Please provide a password: ", 27);
153         char finalPassword[256];
154         fgets(finalPassword, sizeof(finalPassword), stdin);
155         char *finalP = malloc(strlen("pass ") + strlen(finalPassword) + 1);
156         strcpy(finalP, "pass ");
157         strcat(finalP, finalPassword);
158         messages[1] = finalP;
159
160     } else if (type == 0) { // case ftp://user:password@host/path
161
162         char *finalUser = malloc(strlen("user ") + strlen(user) + 1);
163         strcpy(finalUser, "user ");
164         strcat(finalUser, user);
165         messages[0] = finalUser;
166
167         char *finalPassword = malloc(strlen("pass ") + strlen(password) + 1);
168         strcpy(finalPassword, "pass ");
169         strcat(finalPassword, password);
170         messages[1] = finalPassword;
171     }
172
173     char buf = ' ';
174
175     write(sockfd, messages[0], strlen(messages[0])); // user user
176     write(STDOUT_FILENO, "> ", 2);
177     write(STDOUT_FILENO, messages[0], strlen(messages[0]));
178     write(sockfd, "\n", 1); // new line
179     write(STDOUT_FILENO, "\n", 1);
180
181     char * ip = malloc(50);
182
183     int scnd_state = START;
184     while (scnd_state != END)
185     {
186
187         read(sockfd, &buf, 1);
188         write(STDOUT_FILENO, &buf, 1);
189
190         switch (scnd_state)
191         {
192         case START:
193             if (buf == '3')
194             {
195                 read(sockfd, &buf, 1);
196                 write(STDOUT_FILENO, &buf, 1);
197                 if (buf == '3')
198                 {
199                     read(sockfd, &buf, 1);
200                     write(STDOUT_FILENO, &buf, 1);
201                     if (buf == '1')
202                     {
203                         scnd_state = FIRST_ANS;
204                     }
```

```
205                 }
206             }
207             break;
208         case FIRST_ANS:
209             if (buf == '\n')
210             {
211                 write(sockfd, messages[1], strlen(messages[1])); // pass password
212                 write(STDOUT_FILENO, "> ", 2);
213                 write(STDOUT_FILENO, messages[1], strlen(messages[1]));
214                 write(sockfd, "\n", 1);
215                 write(STDOUT_FILENO, "\n", 1);
216                 scnd_state = SEARCH_SCND;
217             }
218             break;
219         case SEARCH_SCND:
220             if (buf == '2')
221             {
222                 read(sockfd, &buf, 1);
223                 write(STDOUT_FILENO, &buf, 1);
224                 if (buf == '3')
225                 {
226                     read(sockfd, &buf, 1);
227                     write(STDOUT_FILENO, &buf, 1);
228                     if (buf == '0')
229                     {
230                         scnd_state = SCND_ANS;
231                     }
232                 }
233             }
234             break;
235         case SCND_ANS:
236             if (buf == '\n')
237             {
238                 write(sockfd, messages[2], strlen(messages[2])); // pasv
239                 write(STDOUT_FILENO, "> ", 2);
240                 write(STDOUT_FILENO, messages[2], strlen(messages[2]));
241                 scnd_state = SEARCH_THIRD;
242             }
243             break;
244         case SEARCH_THIRD:
245             if (buf == '2')
246             {
247                 read(sockfd, &buf, 1);
248                 write(STDOUT_FILENO, &buf, 1);
249                 if (buf == '2')
250                 {
251                     read(sockfd, &buf, 1);
252                     write(STDOUT_FILENO, &buf, 1);
253                     if (buf == '7')
254                     {
255                         scnd_state = GETTING_LAST;
256                     }
257                 }
258             }
```

```
            break;
        case GETTING_LAST:
            if (buf == '(') {
                int i = 0;
                read(sockfd, &buf, 1);
                write(STDOUT_FILENO, &buf, 1);
                do {
                    ip[i] = buf;
                    i++;
                    read(sockfd, &buf, 1);
                    write(STDOUT_FILENO, &buf, 1);
                } while (buf != ')');
                while(buf != '\n') {
                    read(sockfd, &buf, 1);
                    write(STDOUT_FILENO, &buf, 1);
                }
                scnd_state = END;
            }
            break;
        default:
            break;
        }
    }

    int port = getServerPort(ip);

    establishConnection2(inet_ntoa(*((struct in_addr *)h->h_addr)), port);

    write(sockfd, "retr ", 5);
    write(STDOUT_FILENO, "> ", 2);
    write(STDOUT_FILENO, "retr ", 5);
    write(sockfd, filepath, strlen(filepath));
    write(STDOUT_FILENO, filepath, strlen(filepath));
    write(sockfd, "\n", 1);
    write(STDOUT_FILENO, "\n", 1);

    char * filename = getFileName(filepath);
    int file = open(filename, O_CREAT | O_WRONLY, 0777);

    char buffer[256];
    int bytes;
    while ((bytes = read(sockfd2, buffer, sizeof(256)))) {
        write(file, buffer, bytes);
    }

    close(file);

    close(sockfd);
    close(sockfd2);

    return 0;
}
```

functions.c

```c
#include "functions.h"

void establishConnection(char *serverAddr)
{

    /*server address handling*/
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(serverAddr); /*32 bit Internet address network byte ordered*/
    server_addr.sin_port = htons(SERVER_PORT);           /*server TCP port must be network byte ordered */

    /*open an TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket()");
        exit(0);
    }
    /*connect to the server*/
    if (connect(sockfd,
                (struct sockaddr *)&server_addr,
                sizeof(server_addr)) < 0)
    {
        perror("connect()");
        exit(0);
    }

    write(STDOUT_FILENO, "[Connection 1 established w/ success]\n", 38);
}

void establishConnection2(char *serverAddr, int port)
{

    /*server address handling*/
    bzero((char *)&server_addr2, sizeof(server_addr2));
    server_addr2.sin_family = AF_INET;
    server_addr2.sin_addr.s_addr = inet_addr(serverAddr); /*32 bit Internet address network byte ordered*/
    server_addr2.sin_port = htons(port);           /*server TCP port must be network byte ordered */

    /*open an TCP socket*/
    if ((sockfd2 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket()");
        exit(0);
    }
    /*connect to the server*/
    if (connect(sockfd2,
                (struct sockaddr *)&server_addr2,
                sizeof(server_addr2)) < 0)
```

```
49      {
50          perror("connect()");
51          exit(0);
52      }
53
54      write(STDOUT_FILENO, "[Connection 2 established w/ success]\n", 38);
55  }
56
57  char *getFileName(char *filepath) {
58      int lastBarIndex = 0;
59      for (int i = 0; i < strlen(filepath); i++) {
60          if (filepath[i] == '/') {
61              lastBarIndex = i+1;
62          }
63      }
64
65      char * filename = malloc(50);
66      int k = 0;
67      for (int i = lastBarIndex; i < strlen(filepath); i++, k++) {
68          filename[k] = filepath[i];
69      }
70
71      return filename;
72  }
73
74  /*
75  struct hostent {
76    char    *h_name;       Official name of the host.
77      char    **h_aliases;  A NULL-terminated array of alternate names for the host.
78    int     h_addrtype;     The type of address being returned; usually AF_INET.
79      int     h_length;     The length of the address in bytes.
80    char    **h_addr_list;  A zero-terminated array of network addresses for the host.
81                            Host addresses are in Network Byte Order.
82  };
83
84  #define h_addr h_addr_list[0] The first address in h_addr_list.
85  */
86  void getHostInfo(char *hostName)
87  {
88
89      if ((h = gethostbyname(hostName)) == NULL)
90      {
91          herror("gethostbyname");
92          exit(1);
93      }
94  }
95
96  int getServerPort(char * arr) {
97      int i = 0;
98      char *p = strtok (arr, ",");
99      char *array[6];
100
101     while (p != NULL)
102     {
```

```
103         array[i++] = p;
104         p = strtok (NULL, ",");
105     }
106 
107     int port = 256*atoi(array[4]) + atoi(array[5]);
108     return port;
109 }
```

functions.h

```
1  #ifndef FUNCTIONS_H
2  #define FUNCTIONS_H
3  
4  #pragma once
5  
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <errno.h>
11 #include <netdb.h>
12 #include <netinet/in.h>
13 #include <arpa/inet.h>
14 #include <sys/socket.h>
15 #include <unistd.h>
16 #include <signal.h>
17 #include <fcntl.h>
18 
19 #define SERVER_PORT 21
20 
21 #define h_addr2 h_addr_list[1] The first address in h_addr_list.
22 
23 #define START 0
24 #define END 1
25 #define FIRST_ANS 2
26 #define SEARCH_SCND 3
27 #define SCND_ANS 4
28 #define SEARCH_THIRD 5
29 #define GETTING_LAST 6
30 
31 #define FTP 0
32 #define USER 1
33 #define PASSWORD 2
34 #define HOST 3
35 #define PATH 4
36 
37 int sockfd;
38 struct sockaddr_in server_addr;
39 
40 int sockfd2;
41 struct sockaddr_in server_addr2;
42 
43 struct hostent *h;
44 
```

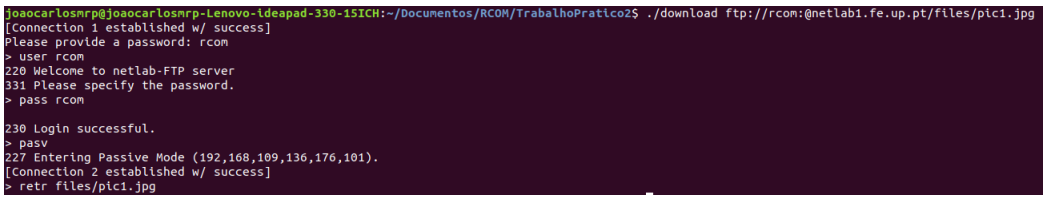
```
45 void establishConnection(char *serverAddr);
46 void establishConnection2(char *serverAddr, int port);
47 char *getFileName(char *filepath);
48 void getHostInfo(char *hostName);
49 int getServerPort(char * arr);
50
51 #endif //FUNCTIONS_H
```

Makefile

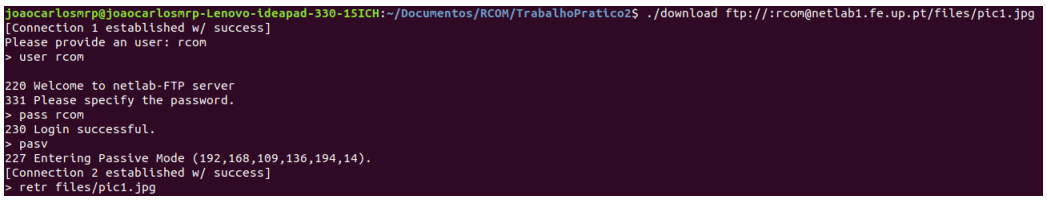
```
1 # Makefile
2
3 simpledu: download.c functions.c functions.h
4   gcc -Wall -o download download.c functions.c
5
6 clean:
7   rm download download.o
```

7.2 Pictures

Download Application - Username and Password Provided



Download Application - Only Username Provided



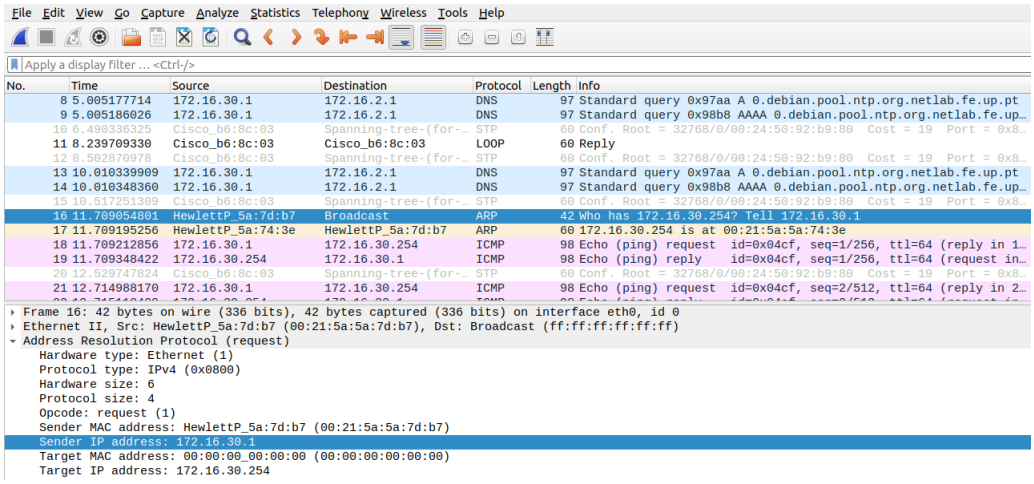
Download Application - Only Password Provided

```
joaocarlosmrp@joaocarlosmrp-Lenovo-ideapad-330-15ICH:~/Documentos/RCOM/TrabalhoPratico2$ ./download ftp://ftp.up.pt/pub/parrot/last-sync.txt
[Connection 1 established w/ success]
Do you want to user anonymous mode? [y/N] y
> user anonymous
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------------
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-
220
331 Please specify the password.
> pass 123456
230 Login successful.
> pasv
227 Entering Passive Mode (193,137,29,15,195,138).
[Connection 2 established w/ success]
> retr pub/parrot/last-sync.txt
```

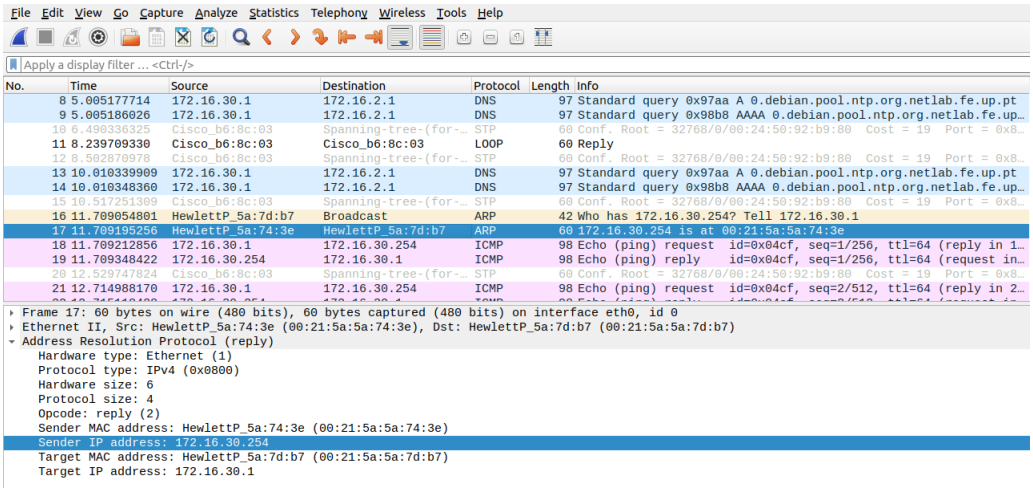
Download Application - None Provided (Anonymous)

```
joaocarlosmrp@joaocarlosmrp-Lenovo-ideapad-330-15ICH:~/Documentos/RCOM/TrabalhoPratico2$ ./download ftp://netlab1.fe.up.pt/files/pic1.jpg
[Connection 1 established w/ success]
Do you want to user anonymous mode? [y/N] N
Please provide an user: rcom
Please provide a password: rcom
> user rcom

220 Welcome to netlab-FTP server
331 Please specify the password.
> pass rcom

230 Login successful.
> pasv
227 Entering Passive Mode (192,168,109,136,156,214).
[Connection 2 established w/ success]
> retr files/pic1.jpg
```

Download Application - None Provided (Non-Anonymous)



EXP1 - IP and MAC addresses of request



EXP1 - IP and MAC addresses of reply

```
root@gnu32:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.31.254   0.0.0.0         UG    0      0        0 eth0
172.16.2.0      172.16.31.254   255.255.255.0   UG    0      0        0 eth0
172.16.30.0     172.16.31.253   255.255.255.0   UG    0      0        0 eth0
172.16.31.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

EXP3 - Forwarding table