Lab 2

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Grupo 3:
Francisco Rodrigues - 201305627
João Nogueira - up201303882
Marta Lopes - 201208067

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

20 de Dezembro de 2015

1 Sumário

Este relatório tem como objetivo explicar o segundo projeto da Unidade Curricular de Redes de Computadores bem como analisar os resultados obtidos na realização das experiências especificadas no enunciado do mesmo.

Conteúdo

2 Introdução

Este projeto encontra-se dividido em duas grandes partes. Em primeiro lugar, é-nos pedido que desenvolvamos uma aplicação de *download* que proceda à transferência de um ficheiro e que implemente o protocolo *FTP*. Em segundo lugar, é-nos pedido que configuremos e estudemos uma Rede de Computadores seguindo a estrutura das experiências abaixo enumeradas:

1. Configuração de um *IP* de rede;
2. Configuração de duas Redes *LAN* virtuais num *switch*;
3. Configuração de um *router* em *Linux*;
4. Configuração de um *router* comercial implementando *NAT*;
5. *DNS*;
6. Conexões *TCP*.

3 Parte 1 - Aplicação de download

Como referido anteriormente, a primeira parte deste tranalho consiste numa aplicação que transfere um ficheiro utilizando o protocolo *FTP* descrito no ficheiro RFC959. Como método de *input* é utilizada a sintaxe mostrada na figura abaixo como descrito no ficheiro RFC1738.

```
joao@joao-VirtualBox:~/Documents/Github/RCOM-code/ftpDownloader$ ./download ftp://
PROGRAM: Entering anonymous Mode
ERROR - Wrong paramater -> URL | Expected something like: ftp://[<user>:<password>@]<host>/<url-path>
```

Figura 1: Input

A aplicação desenvolvida permite que seja feito um download em modo anónimo. Para tal basta não colocar os caracteres '@' e ':' e não colocar nome de utilizador e password. Neste caso a aplicação irá assumir o utilizador *anonymous* e a palavra-passe vazia.

3.1 Arquitetura

A *UrlStruct* é a estrutura definida respponsável por guardar a informação necessária que depende do *input* do utilizador.

```
1  typedef struct{
2      char * user;
3      char * password;
4      struct hostent * h;
5      char * urlPath;
6      char * hostIp;
7  }urlStruct;
```

urlStruct

Ao correr o programa é chamada a função *getUrlInfo* que é responsável por pegar na *string* que o utilizador forneceu como argumento e interpretar toda a informação necessária.

```
1  #include "utilities.h"
2
3  void getUrlInfo(char * completeString, urlStruct * url);
```

Url Header

Depois de interpretar a informação introduzida pelo utilizador, e após verificar que esta informação é válida é chamada a função *startConection* responsável por ligar o cliente *FTP* ao servidor através de um *socket*. Com a ligação estabelecida é então necessário chamar função *getControl*, responsável por enviar a informação necessária para o *login* e por enviar o comando ***PASV***, o que vai permitir que haja comunicação em ambos os sentidos.

```
int getControl(FTP * ftp, urlStruct * url, FTP * receiverFtp) {
    if(sendAndReceiveControl(CMD_USER, ftp, receiverFtp, url) != 0) {
        return -1;
    }
    if (strlen(url->password) != 0) {
        msg("Entering in anonymous mode");
    }
    if(sendAndReceiveControl(CMD_PASS, ftp, receiverFtp, url) != 0) {
        return -1;
    }
    if(sendAndReceiveControl(CMD_PASSV, ftp, receiverFtp, url) != 0) {
        return -1;
    }
    return 0;
}
```

getControl

É também feita uma nova conexão através da função *startReceiverConection* para permitir a receção do ficheiro. pedido pelo utilizador. Por fim é enviado o comando ***RETR*** e recebido o ficheiro a ser guardado. A função *receiveFile* é responsável por enviar o comando, receber o ficheiro e escrevê-lo no disco.

Terminada a receção do ficheiro resta apenas fechar os *sockets* abertos e libertar a memória alocada para terminar o programa.

As funções acima referidas e outras auxiliares estão definifdas abaixo, bem como nos anexos.

```
#include "url.h"

typedef struct
{
    int socketFd; // file descriptor to control socket
    int dataSocketFd; // file descriptor to data socket

    int passvAnswer[6];
    int port;
    char ip[MAX_STRING_DEBUG_SIZE];

} FTP;

int startConection(urlStruct * url, FTP * ftp);
int showResponse(FTP * ftp);
int sendAndReceiveControl(int cmd, FTP * ftp, FTP * receiverFtp, urlStruct * url);
int receivePassvAnswer(FTP * ftp);
int getControl(FTP * ftp, urlStruct * url, FTP * receiverFTP);
int startReceiverConection(urlStruct * url, FTP * ftp);
int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp);
```

conection.h

Durante o desenvolvimento da aplicação foi implementado um modo de *debug* que é ativo ao alterar a Macro *DEBUG* de 0 para 1. Este modo faz com que haja mais impressões na consola, o que permite controlar com maior exatidão o modo como a aplicação está a funcionar.

```
#define DEBUG_MODE 0
#define MAX_STRING_DEBUG_SIZE 100
#define PORT_FTP 21

#define CMD_USER 0
#define CMD_PASS 1
#define CMD_PASSV 2
```

Macros

3.2 Resultados

Esta aplicação foi testada com diversos ficheiros, tanto em modo anónimo como em modo não anónimo. A transferência dos vários ficheiros foi verificada tendo sido o máximo ficheiro testado um ficheiro de vídeo com cerca de 200MB.

Em caso de erro, para além da aplicação terminar é impresso na consola o erro em causa, de modo a que o utilizador tenha o máximo controlo possível sobre o sucedido.

4 Parte 2 - Configuração de Redes

5 Conclusões

6 Anexos

6.1 Headers

6.1.1 conection.h

```
#include "url.h"

typedef struct
{
    int socketFd; // file descriptor to control socket
    int dataSocketFd; // file descriptor to data socket

    int passvAnswer[6];
    int port;
    char ip[MAX_STRING_DEBUG_SIZE];

} FTP;

int startConection(urlStruct * url, FTP * ftp);
int showResponse(FTP * ftp);
int sendAndReceiveControl(int cmd, FTP * ftp, FTP * receiverFtp, urlStruct *
    url);
int receivePassvAnswer(FTP * ftp);
int getControl(FTP * ftp, urlStruct * url, FTP * receiverFTP);
int startReceiverConection(urlStruct * url, FTP * ftp);
int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp);
```

Anexo 1 - conection.h

6.1.2 url.h

```
#include "utilities.h"

void getUrlInfo(char * completeString, urlStruct * url);
```

Anexo 2 - url.h

6.1.3 utilities.h

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

#define DEBUG_MODE 0
#define MAX_STRING_DEBUG_SIZE 100
```

```
#define PORT_FTP 21

#define CMD_USER 0
#define CMD_PASS 1
#define CMD_PASSV 2


typedef struct{
    char * user;
    char * password;
    struct hostent * h;
    char * urlPath;
    char * hostIp;
}urlStruct;

void debug(char * msg1, char * msg);
void msg(char * m);
void stringMsg(char * m, char * m2);
void getName(char * url, char ** filename);
void responseMsg(char * m, char * m2);
```

Anexo 3 - utilities.h

6.2 *.c files

6.2.1 main.c

```
#include "conection.h"

int main(int argc, char *argv[])
{
    debug("", "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

    urlStruct * url = malloc(sizeof(urlStruct));
    urlStruct * receiverUrl = malloc(sizeof(urlStruct));

    if (argc != 2) {
        fprintf(stderr,"usage: getip address\n");
        exit(1);
    }
    getUrlInfo(argv[1], url);

    FTP ftp;
    FTP receiverFtp;

    if (startConection(url, &ftp) < 0) {
        msg("Error starting connection\n");
        return -1;
    }

    if (getControl(&ftp, url, &receiverFtp) < 0) {
        msg("ERROR - getControl()");
        return -1;
    }

```

```
29      if (startReceiverConection(receiverUrl, &receiverFtp) < 0) {
30          msg("Error starting receiver connection");
31          return -1;
32      }
33
34      if (receiveFile(url, &ftp, &receiverFtp) < 0) {
35          msg("Error receiving file");
36          return -1;
37      }
38
39      close(receiverFtp.socketFd);
40      close(ftp.socketFd);
41      msg("Sockets Closed, Terminating...");
42
43      free(url->user);
44      free(url->password);
45      free(url->urlPath);
46      free(url);
47
48      return 0;
49  }
```

Anexo 4 - main.c

6.2.2 conection.c

```
1   #include "conection.h"
2
3   int startConection(urlStruct * url, FTP * ftp) {
4       debug("###################### DEBUG START CONNECTION ####################", "
            BEGIN");
5       int socket_fd;
6       struct sockaddr_in server_addr;
7
8       // Configuring server address
9       bzero((char*)&server_addr, sizeof(server_addr));
10      server_addr.sin_family = AF_INET;
11      server_addr.sin_addr.s_addr = inet_addr(url->hostIp);                // 32
            bit Internet address network byte ordered
12      server_addr.sin_port = htons(PORT_FTP);                             // server
            TCP port (21) must be network byte ordered
13
14      // Opening the control TCP socket
15      socket_fd = socket(AF_INET, SOCK_STREAM, 0);
16      if(socket_fd < 0) {
17          printf("Error opening control TCP socket\n");
18          return -1;
19      } else {
20          char temp[MAX_STRING_DEBUG_SIZE];
21          sprintf(temp, "%d", socket_fd);
22          debug("Opened control TCP socket with Fd    ", temp);
23      }
24
25      // Connecting to the server...
26      int connectRet;
```

```
27      if((connectRet = connect(socket_fd, (struct sockaddr *)&server_addr, sizeof
            (server_addr))) < 0) {
28          perror("connect()");
29          printf("Error connecting to the server to open the control connection\n
                ");
30          return -1;
31      } else {
32          char temp[MAX_STRING_DEBUG_SIZE];
33          sprintf(temp, "%d", connectRet);
34          debug("Connected to server with return      ", temp);
35      }
36
37      ftp->socketFd = socket_fd;
38
39      msg("Connected");
40      debug("#################### DEBUG START CONNECTION ####################", "
            END");
41
42      return 0;
43  }
44
45  int showResponse(FTP * ftp) {
46      char answerFromServer[MAX_STRING_DEBUG_SIZE] = "";
47      if(read(ftp->socketFd, answerFromServer, MAX_STRING_DEBUG_SIZE) > 0) {
48          responseMsg("Server response", answerFromServer);
49          return 0;
50      }
51      msg("Could not read response form server");
52      return -1;
53  }
54
55  int receivePassvAnswer(FTP * ftp) {
56      char passvAnswer[MAX_STRING_DEBUG_SIZE];
57      if(read(ftp->socketFd, passvAnswer, MAX_STRING_DEBUG_SIZE) > 0) {
58          responseMsg("Server response", passvAnswer);
59          if(6 != sscanf(passvAnswer, "%*[^(](%d,%d,%d,%d,%d,%d)\n", &(ftp->
                passvAnswer[0]), &(ftp->passvAnswer[1]), &(ftp->passvAnswer[2]), &(
                ftp->passvAnswer[3]), &(ftp->passvAnswer[4]), &(ftp->passvAnswer[5])
                ))
60          {
61              stringMsg("Could not read the 6 bytes from the server response",
                    passvAnswer);
62              return -1;
63          }
64          return 0;
65      }
66
67      msg("Could not read response form server socket - PASSV");
68      return -1;
69  }
70
71  int getControl(FTP * ftp, urlStruct * url, FTP * receiverFtp) {
72      if(sendAndReceiveControl(CMD_USER, ftp, receiverFtp, url) != 0) {
73          return -1;
74      }
75
```

```
76
77      if (strlen(url->password) != 0) {
78          msg("Entering in anonymous mode");
79      }
80
81      if(sendAndReceiveControl(CMD_PASS, ftp, receiverFtp, url) != 0) {
82          return -1;
83      }
84
85      if(sendAndReceiveControl(CMD_PASSV, ftp, receiverFtp, url) != 0) {
86          return -1;
87      }
88
89      return 0;
90  }
91
92  int sendAndReceiveControl(int cmdSelector, FTP * ftp, FTP * receiverFtp,
        urlStruct * url) {
93      debug("##################### DEBUG SEND AND RECEIVE CONTROL
            ####################", "BEGIN");
94      char cmd[MAX_STRING_DEBUG_SIZE];
95      switch(cmdSelector) {
96          case CMD_USER:
97              strcpy(cmd, "user \0");
98              strcat(cmd, url->user);
99              break;
100         case CMD_PASS:
101             strcpy(cmd, "pass \0");
102             strcat(cmd, url->password);
103             break;
104         case CMD_PASSV:
105             strcpy(cmd, "pasv \0");
106             break;
107         default:
108             break;
109     }
110     strcat(cmd, "\n");
111     debug("Command to Send", cmd);
112     if(write(ftp->socketFd, cmd, strlen(cmd)) < 0) {
113         perror("write()");
114         return -1;
115     } else {
116         debug("Command Sent", cmd);
117     }
118     sleep(1);
119     switch(cmdSelector) {
120         case CMD_PASSV:
121             if(0 == receivePassvAnswer(ftp)) {
122                 // Parse new IP address
123                 receiverFtp->port = ftp->passvAnswer[4] * 256 + ftp->
                        passvAnswer[5];
124                 memset(receiverFtp->ip, 0, MAX_STRING_DEBUG_SIZE); // clearing
                        the array, "just in case"
125                 sprintf(receiverFtp->ip, "%d.%d.%d.%d", ftp->passvAnswer[0],
                        ftp->passvAnswer[1], ftp->passvAnswer[2], ftp->passvAnswer
                        [3]);
```

```
126                 stringMsg("IP address to receive file", (char *) &receiverFtp->
                        ip);
127                 char temp[MAX_STRING_DEBUG_SIZE];
128                 sprintf(temp, "%d", receiverFtp->port);
129                 stringMsg("Port to receive file      ", temp);
130 
131                 debug("#################### DEBUG SEND AND RECEIVE CONTROL
                        ####################", "END");
132                 return 0;
133             }
134 
135             msg("Could not receive the 'passvAnswer' response");
136             return -1;
137         default:
138             showResponse(ftp);
139             break;
140     }
141     debug("#################### DEBUG SEND AND RECEIVE CONTROL
            ####################", "END");
142     return 0;
143 }
144 
145 int startReceiverConection(urlStruct * url, FTP * ftp) {
146     debug("#################### DEBUG START RECEIVER CONECTION
            ####################", "BEGIN");
147     struct sockaddr_in server_addr;
148     char *host_ip;
149     int socket_fd;
150 
151     if (NULL == (url->h = gethostbyname(ftp->ip))) {
152         msg("Could not get host");
153         return -1;
154     }
155 
156     host_ip = inet_ntoa(*((struct in_addr *)url->h->h_addr));
157 
158     stringMsg("Host name  ", url->h->h_name);
159     stringMsg("IP Address ", host_ip);
160 
161     // Configuring server address
162     bzero((char*)&server_addr,sizeof(server_addr));
163     server_addr.sin_family = AF_INET;
164     server_addr.sin_addr.s_addr = inet_addr(host_ip);   //32 bit Internet
            address network byte ordered
165     server_addr.sin_port = htons(ftp->port);                //server TCP port
            must be network byte ordered | this is the new port received from the
            control TCP connection!
166 
167     // Opening the receiver TCP socket
168     socket_fd = socket(AF_INET, SOCK_STREAM,0);
169     if (socket_fd < 0) {
170         msg("Error opening receiver TCP socket");
171         return -1;
172     }
173 
174     int connectRet;
```

```
175     if((connectRet = connect(socket_fd, (struct sockaddr *)&server_addr, sizeof
            (server_addr))) < 0) {
176         perror("connect()");
177         msg("Error connecting to the server to open the control connection");
178         return -1;
179     } else {
180         char temp[MAX_STRING_DEBUG_SIZE];
181         sprintf(temp, "%d", connectRet);
182         debug("Connected to server with return      ", temp);
183     }
184 
185 
186     ftp->socketFd = socket_fd;
187     ftp->dataSocketFd = socket_fd;
188     msg("Connected to receiver");
189     debug("#################### DEBUG START RECEIVER CONECTION
            ####################", "END");
190 
191     return 0;
192 }
193 
194 int receiveFile(urlStruct * url, FTP * ftp, FTP * receiverFtp) {
195     debug("#################### DEBUG RECEIVE FILE ####################", "
            BEGIN");
196     msg("Receiving File...");
197     char cmd[MAX_STRING_DEBUG_SIZE] = "";
198     strcpy(cmd, "retr ");
199     strcat(cmd, url->urlPath);
200     strcat(cmd, "\n");
201     stringMsg("Command to be sent", cmd);
202     if(write(ftp->socketFd, cmd, strlen(cmd)) < 0){
203         msg("ERROR - retr command could not be sent");
204         return -1;
205     }
206     stringMsg("Command sent", cmd);
207 
208     char * filename;
209     getName(url->urlPath, &filename);
210     debug("Filename is", filename);
211 
212     FILE* file;
213     int bytes;
214 
215     if (!(file = fopen(filename, "w"))) {
216         msg("ERROR: Cannot open file.");
217         return -1;
218     }
219 
220     char temp[MAX_STRING_DEBUG_SIZE];
221     sprintf(temp,"%d",  receiverFtp->dataSocketFd);
222     debug("Receiver Data Socket fd", temp);
223 
224     char buf[1024];
225     while ((bytes = read(receiverFtp->dataSocketFd, buf, sizeof(buf)))) {
226         if (bytes < 0) {
227             msg("ERROR: Nothing was received.");
```

```
228             return -1;
229         }
230 
231         if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
232             msg("ERROR: Cannot write data in file.\n");
233             return -1;
234         }
235         debug("In read cycle", "Read 1 byte");
236     }
237 
238     fclose(file);
239 
240     msg("File received");
241     debug("#################### DEBUG RECEIVE FILE ####################", "END"
        );
242 
243     return 0;
244 }
```

Anexo 5 - conection.c

6.2.3 url.c

```
1  #include "url.h"
2  
3  void getUrlInfo(char * completeString, urlStruct * url) {
4      debug("#################### DEBUG URL INFO ####################", "BEGIN");
5  
6      if(strncmp(completeString, "ftp://", 6)) {
7          printf("Wrong Url on argument, expected begining like: 'ftp://'\n");
8          exit(1);
9      }
10     //######################### debug code #########################
11     char debugString[MAX_STRING_DEBUG_SIZE];
12     char debugString_2[MAX_STRING_DEBUG_SIZE];
13     char debugString_3[MAX_STRING_DEBUG_SIZE];
14     char debugString_4[MAX_STRING_DEBUG_SIZE];
15     char debugString_5[MAX_STRING_DEBUG_SIZE];
16     //######################### debug code #########################
17 
18     char * at = strchr(completeString, '@');
19     if (at == NULL) {
20         msg("Entering anonymous Mode");
21     }
22 
23     char * toTwoPoints = strchr(completeString + 6, ':');
24     char * slashAfterAt = strchr(completeString + 7, '/');
25 
26     if (slashAfterAt == NULL) {
27         printf("ERROR - Wrong paramater -> URL | Expected something like: ftp
           ://[<user>:<password>@]<host>/<url-path>\n");
28         exit(1);
29     }
30 
31     int lengthOfUserAndPassword;
32     int lengthOfUser;
```

```
33      int lengthOfPassword;
34      int lengthOfHost;
35      int lengthOfUrlPath;
36
37      if (at != NULL) {
38          lengthOfUserAndPassword = (int) (at - completeString - 6);
39          lengthOfUser = (int) (toTwoPoints - completeString - 6);
40          lengthOfPassword = lengthOfUserAndPassword - lengthOfUser - 1;
41          lengthOfHost = (int) (slashAfterAt - at - 1);
42          lengthOfUrlPath = strlen(completeString) - (9 + lengthOfUser +
                lengthOfPassword + lengthOfHost);
43      } else {
44          lengthOfUserAndPassword = 0;
45          lengthOfUser = (int) 0;
46          lengthOfPassword = 0;
47          lengthOfHost = (int) (slashAfterAt - completeString - 6);
48          lengthOfUrlPath = strlen(completeString) - (7 + lengthOfUser +
                lengthOfPassword + lengthOfHost);
49      }
50
51      if(lengthOfHost <= 0 || lengthOfUrlPath <= 0) {
52          printf("ERROR - Wrong paramater -> URL | Expected something like: ftp
                ://[<user>:<password>@]<host>/<url-path>\n");
53          exit(1);
54      }
55
56      //######################### debug code ########################
57      sprintf(debugString, "%d", lengthOfUser);
58      sprintf(debugString_2, "%d", lengthOfPassword);
59      sprintf(debugString_3, "%d", lengthOfHost);
60      sprintf(debugString_5, "%d", lengthOfUrlPath);
61      debug("Number of characters of the User     ", debugString);
62      debug("Number of characters of the Password ", debugString_2);
63      debug("Number of characters of the Host     ", debugString_3);
64      debug("Number of characters of the Url Path ", debugString_5);
65      //######################### debug code ########################
66
67      url->password = malloc(sizeof(char) * lengthOfPassword);
68      url->urlPath = malloc(sizeof(char) * lengthOfUrlPath);
69      char hostTemp[MAX_STRING_DEBUG_SIZE];
70      if (at != NULL) {
71          url->user = malloc(sizeof(char) * lengthOfUser);
72          strncpy(url->user, completeString + 6, lengthOfUser);
73          strncpy(url->password, completeString + lengthOfUser + 7,
                lengthOfPassword);
74          strncpy(hostTemp, at + 1, lengthOfHost);
75      } else {
76          url->user = malloc(sizeof(char) * strlen("anonymous"));
77          strncpy(url->user, "anonymous", strlen("anonymous"));
78          strncpy(hostTemp, completeString + 6, lengthOfHost);
79      }
80      strncpy(url->urlPath, slashAfterAt + 1, lengthOfUrlPath);
81      hostTemp[lengthOfHost] = '\0';
82
83      //######################### debug code ########################
84      debug("User field                           ", url->user);
```

```
85      debug("Password field                        ", "<password>");
86      debug("Host field                            ", hostTemp);
87      debug("Url Path field                        ", url->urlPath);
88      //######################### debug code ########################
89
90      if ((url->h=gethostbyname(hostTemp)) == NULL) {
91          herror("gethostbyname");
92          exit(1);
93      }
94
95      sprintf(debugString_4, "%s", inet_ntoa(*((struct in_addr *)url->h->h_addr))
            );
96      url->hostIp = malloc(sizeof(char) * strlen(debugString_4));
97      strncpy(url->hostIp, debugString_4, strlen(debugString_4));
98
99      //######################### debug code ########################
100     debug("Host name                             ", url->h->h_name);
101     debug("IP Address                            ", url->hostIp);
102     //######################### debug code ########################
103
104     debug("#################### DEBUG URL INFO ####################", "END");
105     return;
106 }
```

Anexo 6 - url.c

6.2.4 utilities.c

```
1  #include "utilities.h"
2
3  void debug(char * msg1, char * msg) {
4      if (DEBUG_MODE == 1) {
5          printf("DEBUG: %s: %s\n", msg1, msg);
6      }
7  }
8
9  void msg(char * m) {
10     printf("PROGRAM: %s\n", m);
11 }
12
13 void stringMsg(char * m, char * m2) {
14     printf("PROGRAM: %s: %s\n", m, m2);
15 }
16
17 void getName(char * url, char ** filename) {
18     char * temp = url;
19     int i = 0;
20     while(temp != NULL) {
21         if(i > 0)
22             *filename = temp + 1;
23         else
24             *filename = temp;
25         temp = strchr(*filename, '/');
26         i++;
27     }
28 }
```

```
29
30 void responseMsg(char * m, char * m2) {
31     printf("PROGRAM: %s\n%sRESPONSE END\n", m, m2);
32 }
```

Anexo 7 - utilities.c

6.3 Makefile

```
1 all: main.c utilities.c url.c conection.c
2     gcc -Wall -o download main.c utilities.c url.c conection.c
```

Anexo 8 - Makefile