



FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO
PORTO

EIC0032

REDES DE COMPUTADORES

T2 - Redes de Computadores

Autores:

João Miguel
Miguel Barraca
Pedro Fernandes

Números de estudante:

up201604241@fe.up.pt
up201609149@fe.up.pt
up201603846@fe.up.pt

Dezembro 2018

Conteúdo

1	Introdução	2
2	Aplicação <i>download</i>	2
2.1	Arquitetura	2
2.2	Exemplo de um <i>download</i> com sucesso	3
3	Configuração e análise de redes	4
3.1	Experiência 1 - Configurar uma rede IP	4
3.2	Experiência 2 - Implementar duas VLANs num <i>switch</i>	5
3.3	Experiência 3 - Configurar um <i>router</i> em Linux	6
3.4	Experiência 4 - Configurar um <i>router</i> comercial e implementar NAT	7
3.5	Experiência 5 - DNS	7
3.6	Experiência 6 - Conexões TCP	7
4	Conclusão	7
5	Anexos	8
5.1	Código da aplicação <i>download</i>	8
5.2	Comandos de configuração	17
5.3	<i>Logs</i> registados	19

1 Introdução

No âmbito da Unidade Curricular Rede de Computadores (RCOM), foi-nos proposto o desenvolvimento de um projeto composto por duas partes: uma primeira que visou a criação de uma aplicação de *download* e uma segunda que teve como objetivo a configuração de uma rede.

No que diz respeito à primeira parte do trabalho, procedemos ao desenvolvimento da aplicação em linguagem de programação C, capaz de fazer o *download* de ficheiros de um servidor FTP. Numa segunda fase, realizamos a configuração de uma rede e a análise de cada uma das experiências.

Ao longo do presente relatório, pretendemos explicitar a base teórica que sustentou a elaboração deste projeto, levar a cabo uma análise do trabalho prático realizado e da aprendizagem que resultou da concretização do mesmo.

2 Aplicação *download*

2.1 Arquitetura

A aplicação tem o objetivo de fazer o *download* de um ficheiro recorrendo ao protocolo FTP (*File Transfer Protocol*). Para tal, recebe como argumento um link FTP, que deve conter os seguintes campos:

- Nome de utilizador
- Palavra-passe
- Endereço do anfitrião
- Caminho do ficheiro

Caso os dados do utilizador não sejam fornecidos, o programa pedi-los-á posteriormente. Após a análise do argumento, o programa cria a ligação através de duas funções cujo código é já fornecido:

- `char *getServerIp(info_t info)`
- `int createSocketTCP(char *server_ip, int server_port)`

Em caso de conexão bem sucedida, o programa segue os seguintes passos:

1. Enviar USER e PASS
2. Entrar em modo passivo pasv
3. Criar novo socket TCP
4. Enviar comando RETR
5. Transferir o ficheiro
6. Terminar a conexão com QUIT

Esta estrutura é apoiada nas seguintes funções, cuja documentação pode ser encontrada nos anexos:

- `int sendCommand(int socketFd, char *command, char *argument)`
- `int readServerReply(int socketFd, char *reply)`
- `void createFile(int fd, char *filename)`

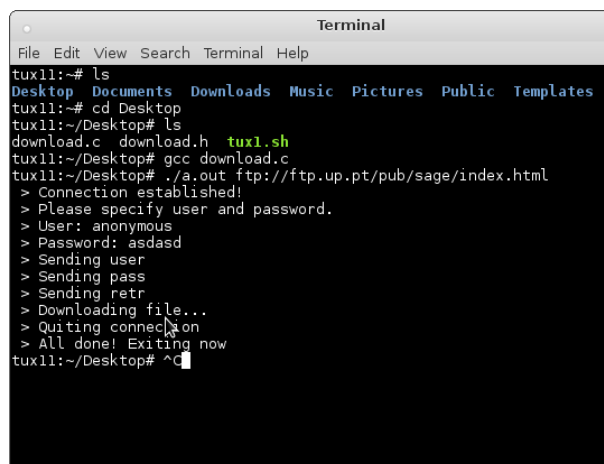
E nas seguintes estruturas:

- `enum state_t`
Representa o estado da leitura da resposta aos comandos enviados.
- `enum reply_type_t`
Representa o tipo da resposta do servidor aos comandos enviados.
- `struct info_t`
Guarda informação fornecida no argumento do programa.

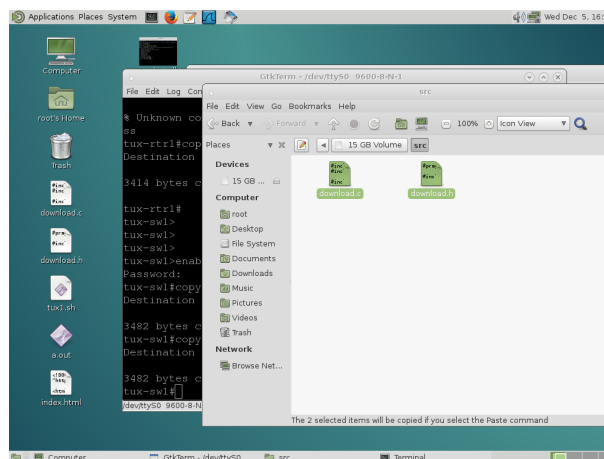
Ao longo do programa são impressas mensagens para informar o utilizador do avanço do processo.

2.2 Exemplo de um *download* com sucesso

Após efetuar todas as experiências, o programa foi corrido com o argumento `ftp://ftp.up.pt/pub/sage/index.html`. O resultado é o que se pode observar nas imagens:



```
Terminal
File Edit View Search Terminal Help
tux11:~# ls
Desktop Documents Downloads Music Pictures Public Templates
tux11:~# cd Desktop
tux11:~/Desktop# ls
download.c download.h tux1.sh
tux11:~/Desktop# gcc download.c
tux11:~/Desktop# ./a.out ftp://ftp.up.pt/pub/sage/index.html
> Connection established!
> Please specify user and password.
> User: anonymous
> Password: asdasd
> Sending user
> Sending pass
> Sending retr
> Downloading file...
> Quitting connection
> All done! Exiting now
tux11:~/Desktop# ^C
```



3 Configuração e análise de redes

3.1 Experiência 1 - Configurar uma rede IP

A primeira experiência teve como objetivo ligar o *tux1* ao *tux4*, usando um *switch*.

1. O que são os pacotes ARP e para que são usados?

Os dispositivos precisam de endereços MAC (endereços físicos) para comunicarem entre si dentro da LAN, pelo que cada interface tem associada ao seu endereço IP um endereço MAC.

Quando um dispositivo (*host* ou *router*) pretende comunicar com outro dispositivo, apenas conhece o endereço IP com que quer comunicar, utilizando o protocolo ARP para obter o endereço MAC associado a este.

Neste protocolo são utilizados pacotes ARP (*ARP request* e *ARP reply*) para que um dispositivo obtenha o endereço MAC de outro dispositivo com quem pretende estabelecer uma comunicação.

De cada vez que um dispositivo precisa obter o endereço MAC de outro dispositivo, envia um pacote de pedido (*ARP request*) a todos os dispositivos, indicando o seu próprio endereço MAC e o endereço IP do dispositivo com que pretende comunicar. Como resposta, o dispositivo com o endereço IP indicado no pedido envia para o primeiro um pacote de resposta (*ARP reply*) com o endereço MAC pedido, tornando-se possível a comunicação entre os dois dispositivos.

Cada dispositivo tem uma tabela ARP que guarda associações entre endereços IP e respetivos endereços MAC. Cada endereço MAC obtido por um dispositivo é guardado na sua tabela para possíveis comunicações futuras. Em resumo, o ARP (*Address Resolution Protocol*) e os pacotes ARP são utilizados para obter o endereço MAC (endereço físico/da camada de ligação) associado a um dado endereço IP (endereço IPv4 de uma interface).

2. O que são os endereços MAC e IP dos pacotes ARP e porquê?

O pacote ARP de pedido (*APR request*) contém os endereços IP e MAC do dispositivo que faz o pedido, para que o recetor saiba qual o dispositivo que pretende obter resposta. Para além disso, contém o endereço IP do dispositivo com que se pretende estabelecer comunicação, para que este envie o seu endereço MAC como resposta.

O pacote ARP de resposta (*APR reply*) transporta o endereço MAC solicitado até ao dispositivo que fez o pedido.

Quando se executa o comando `ping` do *tux1* para o *tux4*, o *tux1* tenta comunicar com o *tux4*. De acordo o protocolo descrito acima, o *tux1* envia um pacote de pedido com os próprios endereços IP e MAC (172.16.10.1 e 00:21:5a:61:28:9c) e o endereço IP do *tux4* (172.16.10.254), com quem pretende estabelecer conexão. Por sua vez, o *tux4*, reconhecendo que o pedido é para si, envia o pacote de resposta com o seu endereço MAC (00:22:64:a6:a4:f8) para o *tux1*.

3. Que pacotes gera o comando `ping`?

O comando `ping` começa por gerar os pacotes APR (de pedido e resposta), como referido na questão 2. De seguida são gerados pacotes do protocolo ICMP.

4. Quais são os endereços MAC e IP dos pacotes `ping`?

Sendo o comando `ping` executado do *tux1* para o *tux4*, os endereços origem correspondem aos endereços do *tux1* (IP = 172.16.10.1 e MAC = 00:21:5a:61:28:9c) e os endereços destino correspondem aos endereços IP e MAC do *tux4* (IP = 172.16.10.254 e MAC = 00:22:64:a6:a4:f8).

5. Como determinar se um quadro *Ethernet* recebido é ARP, IP, ICMP?

O tipo de trama recebido é especificado pelo valor do *Ethernet header*. O valor 0x800 indica que se trata de um IP e o valor 0x806 indica que se trata de um ARP. Se o valor *Ethernet header* for 0x0800, o valor *header* do IP indica se o tipo de protocolo é ICMP (caso IP *header* seja igual a 1, caso contrário está a ser usado outro protocolo).

6. Como determinar o tamanho de um quadro recebido?

O tamanho de uma frame pode ser verificado no *wireshark*.

7. Qual é a interface *loopback* e porque é importante?

É uma interface de rede virtual (endereço IP 127.0.0.1) que permite a um *host* ou *router* comunicar consigo mesmo. Trata-se de uma interface com várias utilidades, entre as quais testar se a conectividade do *router* ou *host* está a funcionar corretamente (ou diagnosticar problemas).

3.2 Experiência 2 - Implementar duas VLANs num *switch*

Na segunda experiência, começamos por criar duas LAN's virtuais: VLAN10 e VLAN11. Posteriormente, associamos os *tux1* e 4 à VLAN10 e associamos o *tux2* à VLAN11.

1. Como configurar a *vlan10*?

- 1) Criar uma VLAN com valor 0, usando os seguintes comandos no GTKTerm:

- `configure terminal`
- `vlan y0`
- `end`

- 2) Adicionar à VLAN criada as portas .1 e .254, que correspondem às portas do *tux1* e do *tux4*, respetivamente. Assim, estes *tux* são ligados à VLAN0, criando-se uma sub-rede. Para tal utilizam-se os comandos:

- `configure terminal`
- `interface fastethernet 0/1 (interface fastethernet 0/254)`
- `switchport mode access`
- `switchport access vlan y0`
- `end`

2. Quantos domínios de transmissão existem? Como se pode concluir a partir dos registos?

Apesar de se executar `ping` do *tux1* para o *tux4*, `ping` do *tux1* para o *tux2*, e `ping` a partir do *tux2*, apenas existem 2 domínios de transmissão. O *tux1* obtém resposta do *tux4*, mas não obtém resposta do *tux2*, assim como o *tux2* não obtém resposta de qualquer outro *tux* (ver figuras 2 e 3).

Tal se justifica pelo facto de ainda não existir nenhuma ligação física entre o *tux2* e os restantes *tux1* e *tux4*, visto que o *tux2* se encontra numa sub-rede diferente da sub-rede dos restantes e *vlan1* e *vlan0* não têm qualquer ligação.

3.3 Experiência 3 - Configurar um *router* em Linux

A terceira experiência teve como propósito a configuração do *tux4* como um *router*, permitindo o estabelecimento de ligação entre os *tux1* e *tux2*, ao estabelecer uma ligação entre a VLAN01 e a VLAN11.

1. Que rotas existem nos *tuxes*? Qual o seu significado?

O *tux1* tem uma rota para a VLAN0 (172.16.y0.0) através da *gateway* 172.16.y0.1 e o *tux2* tem uma rota para a VLAN1 (172.16.y1.0) através da *gateway* 172.16.y1.1. Estes *tuxes*, juntamente com as suas rotas, representam sub-redes diferentes. Assim, o *tux1* e o *tux2* apenas poderão comunicar com o auxílio de um *router*.

O *tux4* tem duas rotas, uma para a VLAN0 pela *gateway* 172.16.y0.254 e outra para a VLAN1 através da *gateway* 172.16.y1.253. O *tux4*, usando estas rotas, simula um *router* que estabelece comunicação entre as duas sub-redes do parágrafo anterior, uma vez que estabelece comunicação entre as VLAN0 e VLAN1, permitindo, desta maneira, que o *tux1* comunique com o *tux2*.

2. Que informação contém uma entrada da tabela de encaminhamento?

- *Destination*: destino da rota
- *Gateway*: IP da interface seguinte
- *Interface*: eth0/etho1

3. Que mensagens ARP e endereços MAC associados são observados, e porquê?

Quando se executa **ping** do *tux1* para o *tux2*, o *tux1* tenta comunicar com o *tux2* (comunicação possibilitada agora pelo *tux4*, que simula o *router*). Para tal, o *tux1* começa por tentar comunicar com o *tux4*, intermediário, que, por sua vez, permite que este comunique com o *tux2*.

Primeiramente, o *tux1* começa por enviar um pacote de pedido APR com os seus endereços IP e MAC (172.16.10.1 e 00:21:5a:61:28:9c) e o endereço IP do *tux4* (172.16.10.254). O *tux4* envia de volta um pacote de resposta com o seu endereço MAC (00:22:64:a6:a4:f8) para o *tux1*, estabelecendo-se a comunicação entre estes.

Estando agora o *tux1* ligado ao *tux4*, pode enviar um pedido APR com os seus endereços e o endereço IP 172.16.11.253 (rota de ligação do *tux4* com a *vlan1*), obtendo o endereço MAC associado ao último.

Por último, o *tux1* envia finalmente o pedido com os seus endereços e o endereço do *tux2* (172.16.11.1). O *tux2* envia o seu endereço MAC para o *tux1*, estabelecendo-se a ligação entre o *tux1* e o *tux2*.

4. Que pacotes ICMP são observados e porquê?

São observados tanto pacotes de pedidos como de resposta, visto que nesta experiência, ao contrário da anterior, todos os *tuxes* podem comunicar entre si.

5. Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?

Os endereços associados são os endereços IP e MAC do *tux* origem e os endereços IP e MAC do *tux* destino.

3.4 Experiência 4 - Configurar um *router* comercial e implementar NAT

1. Como configurar uma rota estática num *router* comercial?
2. Quais são os caminhos seguidos pelos pacotes nas experiências feitas e porquê?
3. Como configurar NAT num *router* comercial?
4. O que faz NAT?

3.5 Experiência 5 - DNS

1. Como configurar o serviço DNS num anfitrião?
2. Que pacotes são trocados por DNS e que informação é transportada?

3.6 Experiência 6 - Conexões TCP

1. Quantas conexões TCP são abertas pela aplicação FTP?
2. Em que conexão é transportada a informação de controlo FTP?
3. Quais são as fases de uma conexão FTP?
4. Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos registos?
5. Como funciona o mecanismo de congestionamento de controlo TCP? Quais são os campos relevantes? Como é que a taxa de transferência da conexão de dados evoluiu ao longo do tempo? Está de acordo com o mecanismo de congestionamento de controlo TCP?
6. A taxa de transferência de uma conexão de dados TCP é perturbada pelo aparecimento de uma segunda conexão TCP? Como?

4 Conclusão

Consideramos que, com o desenvolvimento deste projeto, fomos capazes de cumprir, com sucesso, os objetivos inicialmente propostos. Acrescentamos ainda que este trabalho permitiu a exploração e assimilação de uma série de conceitos, fundamentais para o conhecimento e domínio das redes de computadores.

Referências

- [1] File Transfer Protocol Standard,
<https://tools.ietf.org/html/rfc959>

5 Anexos

5.1 Código da aplicação *download*

download.h

```
#pragma once

#include <stdbool.h>

#define MAX_BUF_SIZE 100
#define MAX_REPLY_SIZE 400
#define SOCKET_BUF_SIZE 1000
#define REPLY_CODE_SIZE 3
#define SERVER_PORT 21

typedef struct
{
    char serverName[MAX_BUF_SIZE];
    char filePath[MAX_BUF_SIZE];
    char fileName[MAX_BUF_SIZE];
    char user[MAX_BUF_SIZE];
    char pass[MAX_BUF_SIZE];
} info_t;

typedef enum
{
    READ_CODE,
    READ_LINE,
    READ_MULT_LINE,
    WAIT_FOR_PORT,
    FIRST_PORT_BYTE,
    SECOND_PORT_BYTE,
    END
} state_t;

typedef enum
{
    POSITIVE_PRE = 1,
    POSITIVE_INT,
    POSITIVE_COMP,
    TRANS_NEGATIVE_COMP,
    PERM_NEGATIVE_COMP
} reply_type_t;

/**
 * @brief Prints a message that shows how to run the program.
 *
 * @param argv array of arguments passed from the command line
 */
```

```
* @return always return 1
*/
int usage(char *argv[]);

/**
 * @brief Parses the argument passed to the program, retrieving user information.
 *
 * @param argument argument from the command line, supposedly an FTP link
 * @param info structure that holds user and server info
 *
 * @return true if argument was successfully read, false otherwise
 */
bool parseArgument(char *argument, info_t *info);

/**
 * @brief Gets a server ip from a host name
 *
 * @param name host name
 *
 * @return server ip
 */
char *getServerIp(const char* name);

/**
 * @brief Creates a TCP socket, returning its respective file descriptor.
 *
 * @param server_ip
 * @param server_port
 *
 * @return socket's file descriptor
 */
int createSocketTCP(char *server_ip, int server_port);

/**
 * @brief Reads the server reply to an FTP command, returning it through the reply argument.
 *
 * @param socketFd socket's file descriptor
 * @param reply numeric descriptor of the server reply
 */
void readServerReply(int socketFd, char *reply);

/**
 * @brief Gets the server port after the program issues pasv command.
 *
 * @param socketFd socket's file descriptor
 *
 * @return the server port
 */
int getServerPort(int socketFd);
```

```

/**
 * @brief Sends and FTP command along with its argument (if applicable) and reads the server reply
 *
 *
 * @param socketFd socket's file descriptor
 * @param command FTP command
 * @param argument command's argument, if any
 *
 * @return 0 for POSITIVE_INT, 1 for POSITIVE_COMP and -1 for PERM_NEGATIVE_COMP
 */
int sendCommand(int socketFd, char *command, char *argument);

/**
 * @brief Called after sending RETR command, reads data from the socket and creates a local file.
 *
 * @param fd second socket's file descriptor
 * @param filename name of the file to be retrieved
 */
void createFile(int fd, char *filename);

```

download.c

```

#include <arpa/inet.h>
#include <netinet/in.h>

#include <sys/socket.h>
#include <sys/types.h>

#include <ctype.h>
#include <errno.h>
#include <netdb.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <unistd.h>

#include "download.h"

int usage(char *argv[])
{
    printf("Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
    return 1;
}

bool parseArgument(char *argument, info_t *info)
{
    char *sep;

```

```
char* lastSep;
int index1 = 6, index2;

if (strncmp("ftp://", argument, 6) != 0)
    return false;

if ((sep = strchr(argument + 6, ':')) != NULL)
{
    int index;

    index = (int)(sep - argument);
    strncpy(info->user, argument + 6, index - 6);
    info->user[index - 6] = '\0';

    if ((sep = strchr(argument, '@')) == NULL)
        return false;

    int new_index = (int)(sep - argument);
    index++;
    strncpy(info->pass, argument + index, new_index - index);
    info->pass[new_index - index] = '\0';

    index1 = ++new_index;
}
else if ((sep = strchr(argument, '@')) != NULL)
    return false;
else
    strncpy(info->user, "placeholder", 11);

if ((sep = strchr(argument + 6, '/')) == NULL)
    return false;

index2 = (int)(sep - argument);

strncpy(info->serverName, argument + index1, index2 - index1);
info->serverName[index2 - index1] = '\0';
index2++;
strncpy(info->filePath, argument + index2, strlen(argument) - index2);
info->filePath[strlen(argument) - index2] = '\0';

lastSep = strrchr(argument, '/');
strcpy(info->fileName, lastSep + 1);
info->fileName[strlen(lastSep)] = '\0';

return true;
}

char *getServerIp(const char* name)
{
```

```
    struct hostent *h;

    if ((h = gethostbyname(name)) == NULL)
    {
        perror("gethostbyname");
        exit(1);
    }

    return inet_ntoa(*(struct in_addr *)h->h_addr_list[0]);
}

int createSocketTCP(char *server_ip, int server_port)
{
    int socketFd;
    struct sockaddr_in server_addr;

    /*server address handling*/
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr =
        inet_addr(server_ip); /*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 ((socketFd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket()");
        exit(0);
    }

    /*connect to the server*/
    if (connect(socketFd, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
        0)
    {
        perror("connect()");
        exit(0);
    }

    return socketFd;
}

void readServerReply(int socketFd, char *reply)
{
    char c;
    int res = 0, i = 0;
    state_t state = READ_CODE;

    while (state != END)
```

```
{
    if ((res = read(socketFd, &c, 1)) <= 0)
        continue;

    switch (state)
    {
    case READ_CODE:
        if (c == ' ')
        {
            state = READ_LINE;
            i = 0;
        }
        else if (c == '-')
        {
            state = READ_MULT_LINE;
            i = 0;
        }
        else if (isdigit(c))
        {
            reply[i++] = c;
        }
        break;
    case READ_LINE:
        if (c == '\n')
            state = END;
        break;
    case READ_MULT_LINE:
        if (c == reply[i])
        {
            i++;
        }
        else if (i == 3 && c == ' ')
        {
            state = READ_LINE;
        }
        break;
    case END:
        break;
    default:
        break;
    }
}

int getServerPort(int socketFd)
{
    int res = 0;
    state_t state = WAIT_FOR_PORT;
    char c;
```

```
char first_byte[4], second_byte[4];
int numCommas = 0, i = 0;

while (state != END)
{
    if ((res = read(socketFd, &c, 1)) <= 0)
        continue;

    switch (state)
    {
        case READ_CODE:
            if (c == ',')
            {
                if (i != 3)
                {
                    printf("> Error receiving response code\n");
                    return -1;
                }
                i = 0;
                state = WAIT_FOR_PORT;
            }
            else
            {
                i++;
            }
            break;
            break;
        case WAIT_FOR_PORT:
            if (c == ',')
                numCommas++;

            if (numCommas == 4)
                state = FIRST_PORT_BYTE;
            break;
        case FIRST_PORT_BYTE:
            if (c == ',')
            {
                state = SECOND_PORT_BYTE;
                i = 0;
            }
            else
            {
                first_byte[i++] = c;
            }
            break;
        case SECOND_PORT_BYTE:
            if (c == ')')
                state = END;
            else
```

```
        {
            second_byte[i++] = c;
        }
        break;
    case END:
        break;
    default:
        break;
    }
}

return atoi(first_byte) * 256 + atoi(second_byte);
}

int sendCommand(int socketFd, char *command, char *argument)
{
    char reply[REPLY_CODE_SIZE];
    reply_type_t type;

    write(socketFd, command, strlen(command));
    if (argument != NULL)
        write(socketFd, argument, strlen(argument));
    write(socketFd, "\n", 1);

    while (true)
    {
        readServerReply(socketFd, reply);

        type = reply[0] - '0';

        switch (type)
        {
            case POSITIVE_PRE:
                break;
            case POSITIVE_INT:
                return 0;
            case POSITIVE_COMP:
                return 1;
            case TRANS_NEGATIVE_COMP:
                write(socketFd, command, strlen(command));
                if (argument != NULL)
                    write(socketFd, argument, strlen(argument));
                write(socketFd, "\n", 1);
                break;
            case PERM_NEGATIVE_COMP:
                close(socketFd);
                return -1;
            default:
                break;
        }
    }
}
```



```
    }
}

void createFile(int fd, char *filename)
{
    FILE *file = fopen(filename, "wb+");

    char fileData[SOCKET_BUF_SIZE];
    int nbytes;
    while ((nbytes = read(fd, fileData, SOCKET_BUF_SIZE)) > 0)
    {
        nbytes = fwrite(fileData, nbytes, 1, file);
    }

    fclose(file);
}

int main(int argc, char *argv[])
{
    info_t info;
    char *server_ip;
    char reply[MAX_REPLY_SIZE];
    int fd1, fd2, res, port;

    if (argc != 2 || !parseArgument(argv[1], &info))
        return usage(argv);

    server_ip = getServerIp(info.serverName);

    fd1 = createSocketTCP(server_ip, SERVER_PORT);
    readServerReply(fd1, reply);

    if (reply[0] == '2')
        printf(" > Connection established!\n");
    else
    {
        printf(" > Couldn't connect! Exiting.\n");
        exit(1);
    }

    if(strncmp(info.user, "placeholder", 11) == 0){
        printf(" > Please specify user and password.\n");
        printf(" > User: ");
        scanf("%s", info.user);
        printf(" > Password: ");
        scanf("%s", info.pass);
    }
}
```

```
printf(" > Sending user\n");
res = sendCommand(fd1, "user ", info.user);

if (res == 0 || res == 1)
{
    printf(" > Sending pass\n");
    res = sendCommand(fd1, "pass ", info.pass);
}
else
{
    printf(" > Error sending username! Exiting.\n");
    exit(1);
}

write(fd1, "pasv\n", 5);
port = getServerPort(fd1);

fd2 = createSocketTCP(server_ip, port);

printf(" > Sending retr\n");
res = sendCommand(fd1, "retr ", info.filePath);

if (res == 0)
{
    printf(" > Downloading file...\n");
    createFile(fd2, info.fileName);
}

printf(" > Quitting connection\n");
write(fd1, "quit\n", 5);

close(fd1);
close(fd2);

printf(" > All done! Exiting now\n");
return 0;
}
```

5.2 Comandos de configuração

De forma a facilitar a configuração dos *tuxes*, foram desenvolvidos os seguintes *bash scripts*.

tux1.sh

```
#!/bin/bash

if [ $# != 1 ] || [ $1 -lt 1 ] || [ $1 -gt 6 ]; then
    echo "Usage: $0 <stand>"
    exit 1
fi
```

```
ip1="172.16.$10.1/24"
ip2="172.16.$11.0/24"
ip3="172.16.$10.254"

ifconfig eth0 up
ifconfig eth0 $ip1
route add -net $ip2 gw $ip3
route add default gw $ip3

tux2.sh

#!/bin/bash

if [ $# != 1 ] || [ $1 -lt 1 ] || [ $1 -gt 6 ]; then
    echo "Usage: $0 <stand>"
    exit 1
fi

ip1="172.16.$11.1/24"
ip2="172.16.$10.0/24"
ip3="172.16.$11.253"
ip4="172.16.$11.254"

ifconfig eth0 up
ifconfig eth0 $ip1
route add -net $ip2 gw $ip3
route add default gw $ip4

tux4.sh

#!/bin/bash

if [ $# != 1 ] || [ $1 -lt 1 ] || [ $1 -gt 6 ]; then
    echo "Usage: $0 <stand>"
    exit 1
fi

ip1="172.16.$10.254/24"
ip2="172.16.$11.253/24"
ip3="172.16.$11.254"

ifconfig eth0 up
ifconfig eth0 $ip1
ifconfig eth1 up
ifconfig eth1 $ip2
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
route add default gw $ip3
```

5.3 Logs registados

Time	Source	Destination	Protocol	Length	Info
1 0.000000	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
2 1.963187	Cisco_3a:fc:03	Cisco_3a:fc:03	LOOP	60	Reply
3 2.010602	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
4 3.877688	HewlettP_61:28:9c	Broadcast	ARP	42	Who has 172.16.10.254? Tell 172.16.10.1
5 3.877821	HewlettP_a6:a4:f8	HewlettP_61:28:9c	ARP	60	172.16.10.254 is at 00:22:64:a6:a4:f8
6 3.877829	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x08ee, seq=1/256, ttl=64 (reply in 7)
7 3.877961	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x08ee, seq=1/256, ttl=64 (request in 6)
8 4.009769	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
9 4.876681	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x08ee, seq=2/512, ttl=64 (reply in 10)
10 4.876815	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x08ee, seq=2/512, ttl=64 (request in 9)
11 5.875681	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x08ee, seq=3/768, ttl=64 (reply in 12)
12 5.875816	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x08ee, seq=3/768, ttl=64 (request in 11)
13 6.014659	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
14 6.874913	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x08ee, seq=4/1024, ttl=64 (reply in 15)
15 6.875074	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x08ee, seq=4/1024, ttl=64 (request in 14)
16 7.874914	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x08ee, seq=5/1280, ttl=64 (reply in 17)
17 7.875048	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x08ee, seq=5/1280, ttl=64 (request in 16)

Figura 1: Experiência 1 (ping do tux1 para tux4)

Time	Source	Destination	Protocol	Length	Info
1 0.000000	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
2 1.627761	Cisco_3a:fc:03	Cisco_3a:fc:03	LOOP	60	Reply
3 2.004831	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
4 4.009730	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
5 6.014665	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
6 7.253423	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x0be4, seq=1/256, ttl=64 (reply in 7)
7 7.253586	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x0be4, seq=1/256, ttl=64 (request in 6)
8 8.019501	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
9 8.252420	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x0be4, seq=2/512, ttl=64 (reply in 10)
10 8.252547	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x0be4, seq=2/512, ttl=64 (request in 9)
11 9.251421	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x0be4, seq=3/768, ttl=64 (reply in 12)
12 9.251552	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x0be4, seq=3/768, ttl=64 (request in 11)
13 10.024368	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
14 10.250493	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x0be4, seq=4/1024, ttl=64 (reply in 15)
15 10.250619	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x0be4, seq=4/1024, ttl=64 (request in 14)
16 11.250494	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x0be4, seq=5/1280, ttl=64 (reply in 17)
17 11.250641	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x0be4, seq=5/1280, ttl=64 (request in 16)

Figura 2: Experiência 2 (ping do tux1 para o tux4)

Time	Source	Destination	Protocol	Length	Info
34 51.033618	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=1/256, ttl=64 (no response found!)
35 52.040737	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=2/512, ttl=64 (no response found!)
36 52.725722	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
37 53.048735	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=3/768, ttl=64 (no response found!)
38 54.056726	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=4/1024, ttl=64 (no response found!)
39 54.730866	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
40 55.064728	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=5/1280, ttl=64 (no response found!)
41 56.072732	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=6/1536, ttl=64 (no response found!)
42 56.740686	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
43 57.080730	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=7/1792, ttl=64 (no response found!)
44 58.088729	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=8/2048, ttl=64 (no response found!)
45 58.740365	Cisco_3a:fc:03	Spanning-tree-(for...	STP	60	Conf. Root = 32768/10/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8003
46 59.096732	172.16.10.1	172.16.10.255	ICMP	98	Echo (ping) request id=0x0c81, seq=9/2304, ttl=64 (no response found!)

Figura 3: Experiência 2 (ping do tux1 para o tux2)

Time	Source	Destination	Protocol	Length	Info
1 0.000000	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
2 0.201418	Cisco_3a:fc:03	Cisco_3a:fc:03	LOOP	60	Reply
3 1.023770	Cisco_3a:fc:03	CDP/VTP/DTP/PAGP/UD...	CDP	435	Device ID: tux-sw1 Port ID: FastEthernet0/1
4 2.010071	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
5 2.705453	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1087, seq=1/256, ttl=64 (reply in 6)
6 2.705611	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1087, seq=1/256, ttl=64 (request in 5)
7 3.704454	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1087, seq=2/512, ttl=64 (reply in 8)
8 3.704612	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1087, seq=2/512, ttl=64 (request in 7)
9 4.009731	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
10 4.703452	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1087, seq=3/768, ttl=64 (reply in 11)
11 4.703581	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1087, seq=3/768, ttl=64 (request in 10)
12 5.702454	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1087, seq=4/1024, ttl=64 (reply in 13)
13 5.702584	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1087, seq=4/1024, ttl=64 (request in 12)
14 6.014628	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
15 6.702147	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1087, seq=5/1280, ttl=64 (reply in 16)
16 6.702279	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1087, seq=5/1280, ttl=64 (request in 15)
17 7.702147	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x1087, seq=6/1536, ttl=64 (reply in 18)
18 7.702281	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1087, seq=6/1536, ttl=64 (request in 17)

Figura 4: Experiência 3 (ping do tux1 para o tux2, part1)

Time	Source	Destination	Protocol	Length	Info
24 12.029298	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
25 12.729434	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x108e, seq=1/256, ttl=64 (reply in 26)
26 12.729593	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x108e, seq=1/256, ttl=64 (request in 25)
27 13.728436	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x108e, seq=2/512, ttl=64 (reply in 28)
28 13.728594	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x108e, seq=2/512, ttl=64 (request in 27)
29 14.039292	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
30 14.727435	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x108e, seq=3/768, ttl=64 (reply in 31)
31 14.727565	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x108e, seq=3/768, ttl=64 (request in 30)
32 15.726439	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x108e, seq=4/1024, ttl=64 (reply in 33)
33 15.726570	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x108e, seq=4/1024, ttl=64 (request in 32)
34 16.039055	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
35 16.726153	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x108e, seq=5/1280, ttl=64 (reply in 36)
36 16.726293	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x108e, seq=5/1280, ttl=64 (request in 35)
37 17.726146	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x108e, seq=6/1536, ttl=64 (reply in 38)
38 17.726278	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x108e, seq=6/1536, ttl=64 (request in 37)

Figura 5: Experiência 3 (ping do tux1 para o tux2, part2)

42 20.665222	172.16.10.1	172.16.11.1	ICMP	98	Echo (ping) request id=0x1092, seq=1/256, ttl=64 (reply in 43)
43 20.665629	172.16.11.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1092, seq=1/256, ttl=63 (request in 42)
44 21.664222	172.16.10.1	172.16.11.1	ICMP	98	Echo (ping) request id=0x1092, seq=2/512, ttl=64 (reply in 45)
45 21.664487	172.16.11.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1092, seq=2/512, ttl=63 (request in 44)
46 22.053783	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
47 22.663225	172.16.10.1	172.16.11.1	ICMP	98	Echo (ping) request id=0x1092, seq=3/768, ttl=64 (reply in 48)
48 22.663496	172.16.11.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1092, seq=3/768, ttl=63 (request in 47)
49 23.662227	172.16.10.1	172.16.11.1	ICMP	98	Echo (ping) request id=0x1092, seq=4/1024, ttl=64 (reply in 49)
50 23.662491	172.16.11.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1092, seq=4/1024, ttl=63 (request in 50)
51 24.058651	Cisco_3a:fc:03	Spanning-tree-(for-...	STP	60	Conf. Root = 32768/10/fc:fb:3a:fc:00 Cost = 0 Port = 0x8003
52 24.662155	172.16.10.1	172.16.11.1	ICMP	98	Echo (ping) request id=0x1092, seq=5/1280, ttl=64 (reply in 53)
53 24.662395	172.16.11.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1092, seq=5/1280, ttl=63 (request in 52)
54 25.662151	172.16.10.1	172.16.11.1	ICMP	98	Echo (ping) request id=0x1092, seq=6/1536, ttl=64 (reply in 55)
55 25.662392	172.16.11.1	172.16.10.1	ICMP	98	Echo (ping) reply id=0x1092, seq=6/1536, ttl=63 (request in 54)

Figura 6: Experiência 3 (ping do tux1 para o tux2, part3)