**Configuração de uma rede e desenvolvimento de uma aplicação de *download***

**Relatório Final**



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

**Turma 2 / Grupo 1:**

António Pedro Araújo Fraga – 201303095

Luís Miguel da Costa Oliveira – 201304515

Miguel Guilherme Perestrelo Sampaio Pereira – 201305998

22 de dezembro de 2015

Sumário

Este relatório tem o propósito de cimentar o trabalho realizado e serve como material de apoio ao projeto que se trata de configurar e estudar uma rede, utilizando comandos de configuração do *router* e do *switch* e de desenvolver uma aplicação capaz de fazer *download* de um ficheiro através de um FTP.

O trabalho foi terminado com sucesso. A aplicação desenvolvida realiza a transferência do ficheiro sem erros e a foi possível configurar corretamente a rede.

Índice

[1. Introdução 3](#_Toc438515204)

[2. Aplicação de *download* 4](#_Toc438515205)

[2.1. Arquitetura 4](#_Toc438515206)

[2.2. Resultados 6](#_Toc438515207)

[3. Estudo da configuração da rede 7](#_Toc438515208)

[3.1. Configurar um IP de rede 7](#_Toc438515209)

[3.2. Implementar duas LANs virtuais no *switch* 8](#_Toc438515210)

[3.3. Configurar um *router* em Linux 8](#_Toc438515211)

[3.4. Configurar um *router* comercial e implementar NAT 8](#_Toc438515212)

[3.5. DNS 9](#_Toc438515213)

[3.6. Ligações TCP 9](#_Toc438515214)

[4. Conclusão 9](#_Toc438515215)

[Referências 10](#_Toc438515216)

[Anexos 11](#_Toc438515217)

[Código da aplicação 11](#_Toc438515218)

[Comandos de configuração 22](#_Toc438515219)

[Logs 24](#_Toc438515220)

# Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Redes de Computadores e tem como metas fundamentais o desenvolvimento de uma aplicação capaz de realizar o *download* de um ficheiro utilizando o *File Transfer Protocol* (FTP) e a configuração e análise de uma rede.

Para a configuração da rede seguiram-se os passos descritos no guião que envolviam a configuração do IP de cada máquina, das LANs virtuais dentro do *switch*, do *router* com a implementação de NAT e, por fim, do DNS.

Estando a rede configurada, testou-se a aplicação de *download*. A aplicação foi desenvolvida em C recorrendo à utilização de *sockets* para a comunicação com o servidor, enviando comandos e recebendo respostas.

Este relatório começa por descrever como foi desenvolvida a aplicação e como ela está estruturada. Depois, serão abordadas cada uma das experiências realizadas ao longo das aulas práticas, explicando o objetivo de cada uma e os comandos utilizados para a realizar a configuração. Por último, serão apresentadas algumas conclusões.

# Aplicação de *download*

## Arquitetura

A aplicação está dividida em duas partes, a primeira parte trata do processamento da *string* que é passada como argumento e guarda todos os seus componentes numa estrutura *url* através da função *init\_url*. Esses componentes fazem parte da informação necessária para que a transferência ocorra, sendo que entre eles estão a *string* de utilizador e palavra-passe assim como a *string* que contém o *hostname* e o caminho do ficheiro a transferir.

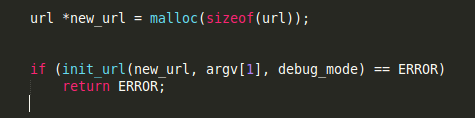


Figura 1 - Chamamento de init\_url

A segunda parte tem como objetivo estabelecer duas ligações (ligação A e ligação B) com o *host* definido anteriormente, fazendo uso de várias tarefas para o conseguir. A ligação A será a ligação de controlo, por isso é inicializada uma estrutura *connection* que guardará os dados necessários.

http://i.imgur.com/BffD1QK.png

Figura 2 - Ligação A

A primeira coisa a fazer é resolver o IP a que o *host* está associado através da função *get\_ip*. Só assim será possível estabelecer uma ligação. Esta função contém maioritariamente código de um exemplo de obtenção de IP a partir da *string* de *host* que estava presente no *moodle* da unidade curricular. Além desse código, é definida a porta a usar na ligação de controlo, a porta 21. A *string* que contém o IP e a porta é guardada na estrutura *connection* da ligação A.

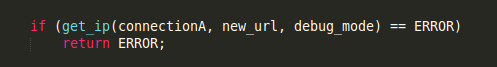


Figura 3 - Chamamento de get\_ip

Depois dessas informações estarem guardadas é possível estabelecer uma ligação com o *host* através da função *connect\_to* que também contém o código disponibilizado no *moodle* de um exemplo de conexão a um determinado IP. Depois de uma tentativa de conexão com o *host*, no caso da ligação de controlo, é recebida uma resposta com um determinado código; a função *read\_from\_host* verifica se o código recebido é o código esperado. Esta função retorna o descritor de ficheiro da ligação aberta, neste caso a ligação A.

Quando a ligação estiver completa é altura de o programa se autenticar fazendo uso do utilizador e palavra-passe guardados anteriormente na estrutura *url*. O programa chama assim a função *log\_in\_host*, que envia uma mensagem com a informação da *string* de utilizador através do método *send\_to\_host*; é recebida a resposta e o código dessa mesma resposta é verificado. Repete-se o mesmo procedimento para enviar a informação sobre a *string* da *password* e no fim deve receber-se o código correspondente à resposta de um uma autenticação bem sucedida.

http://i.imgur.com/ca1cGOJ.png

Figura 4 - Chamamento de log\_in\_host

É então enviado o comando para que o servidor de FTP transfira dados em modo passivo. É executada a função *pasv\_host*, que envia a mensagem e interpreta a reposta do *host*, para que o IP e a porta a serem utilizados pela ligação B possam ser guardados. Esta resposta é interpretada pelo método *get\_pasv\_from\_host* e estes dados são guardados numa nova estrutura *connection*, estrutura essa que se refere à ligação B.

http://i.imgur.com/XEyfJFd.png

Figura 5 - Chamamento de pasv\_host

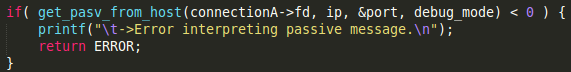


Figura 6 - Chamamento de get\_pasv\_from\_host

Já que o servidor entrou no modo passivo e existem todos os dados de ligação da conexão B (calculados e interpretados anteriormente), é executada a função *connect\_to*, para que haja uma ligação de transferência de ficheiro. Quando a ligação estiver completa é enviada uma mensagem a partir da ligação A que define o caminho do ficheiro dentro do servidor; a resposta do servidor define se o ficheiro existe ou não.

http://i.imgur.com/dxMf0jo.png

Figura 7 - Chamamento de função def\_path

Os dados para que a transferência ocorra estão enviados. Então, a ligação B começa a transferir o ficheiro (*download\_from\_host*), abrindo um ficheiro para escrita e escrevendo os dados recebidos a cada 1024 *bytes*, fechando-o quando este estiver terminado. A ligação B é fechada e passa-se à desconexão da ligação A, enviando-se uma mensagem de terminação ao servidor e libertando a memória alocada anteriormente.

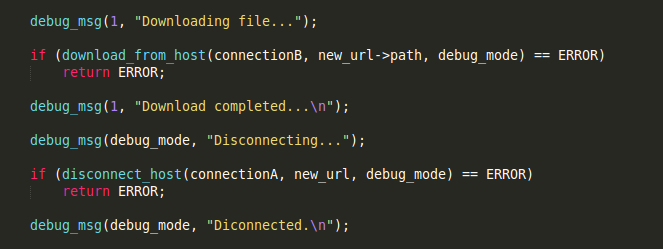


Figura 8 - Terminação da ligação

## Resultados

Para uma melhor análise de resultados o grupo decidiu implementar um modo de *debug* que pode ou não ser ativado. Este modo *debug* imprime várias informações sobre toda a execução do programa desenvolvido. Uma transferência só é bem-sucedida se todas as respostas por parte do servidor contiverem o código de resposta positivo em relação à mensagem enviada pelo programa. Se por qualquer motivo o código de resposta não for aquele que é suposto ser então o programa termina a sua execução e imprime a respetiva mensagem de erro, uma possibilidade seria a falha de autenticação. Esta verificação é feita pelo método *read\_from\_host* que recebe como argumento o código que é suposto receber em cada chamada. De seguida apresenta-se o resultado de uma execução com sucesso com o modo *debug* ativado.

Initializing a url struct.

->Getting url strings...

->Completed!

Url struct initialized.

Getting ip by host name.

->Getting host ip by name...

->Completed!

Host is valid, and it returned a valid ip.

Connecting to A '193.136.37.8' through port 21...

->Creating a socket...

->Socket creatd.

->Connecting...

->Connected!

->Receiving response message...

->Message Code: 220

->Message received!

Connection was successfull.

Logging in.

->Sending user to host...

->User sent!

->Receiving message from host...

->Message Code: 331

->Message received!

->Sending passoword to host...

->Password sent!

->Receiving messagem from host...

->Message Code: 230

->Message received!

Logged in messages were sent.

Entering passive mode...

->Sending passive message to host...

->Passive message sent!

->Interpreting passive message from host...

->Passive Message: 227 Entering Passive Mode (193,136,37,8,251,201)

->Interpreted IP: 193.136.37.8

->Interpreted Port: 64457

->Completed!

Completed!

Connecting to B '193.136.37.8' through port 64457...

->Creating a socket...

->Socket created.

->Connecting...

->Connected!

Connected!

Sending path...

->File: pub/CPAN/RECENT-1M.json

->Sending 'retr' command to host...

->Command sent!

->Receiving message from host...

->Message Code: 150

->Message received!

Path was sent!

Downloading file...

->Creating file with defined path...

->Created!

->Downloading...

->Completed!

Download completed...

Disconnecting...

->Sending 'quit' command to host...

->Closing socket...

Disconnected.

# Estudo da configuração da rede

## Configurar um IP de rede

O objetivo desta experiência foi configurar os endereços de IP de dois computadores para que estes conseguissem comunicar. Depois de configurar as portas eth0 e adicionar as rotas necessárias, utilizou-se o comando *ping* para verificar a existência de uma ligação entre os dois computadores.

O *Address Resolution Protocol* (ARP) é um protocolo utilizado para a resolução de endereços da camada de rede (endereços IP) em endereços da camada de ligação de dados (endereços Ethernet). Para enviar uma trama para um computador na rede, o emissor tenta descobrir o endereço MAC correspondente ao endereço IP, difundindo em *broadcast* um pacote ARP que contém o endereço de IP e espera uma resposta com o endereço MAC que lhe corresponde.

O comando *ping* gera pacotes do protocolo ICMP. Para distinguir os pacotes ARP, IP e ICMP, é necessário verificar o cabeçalho da trama Ethernet, sendo que os pacotes IP contêm, ainda, informação acerca do tamanho da trama.

A interface *loopback* é uma interface de rede virtual que o computador utiliza para realizar testes de diagnóstico. Esta interface permite ter um endereço de IP no *router* que está sempre ativo, não dependendo de uma interface física.

Para esta experiência, utilizou-se o comando *ifconfig* para configurar os endereços de IP nos dois computadores.

## Implementar duas LANs virtuais no *switch*

O objetivo da experiência foi criar duas LANs virtuais no *switch*, uma com os computadores 1 e 4 e a outra com o computador 2. Assim, o computador 2 deixaria de ter acesso aos computadores 1 e 4, visto que se encontram em sub-redes diferentes.

Para configurar o *switch*, entrou-se na sua consola de configuração e executaram-se os seguintes comandos:

* *vlan i*, para indicar que se estaria a atuar sobre a VLAN identificada por *i*;
* *interface fastethernet 0/j*, para adicionar a porta *j* à VLAN;
* *switchport mode access;*
* *switchport access vlan i.*

Existem dois domínios de *broadcast* que correspondem a cada umas das VLANs criadas.

## Configurar um *router* em Linux

Esta experiência tinha como objetivo configurar o computador 4 para funcionar como um *router* que permitisse a comunicação entre as duas VLANs criadas.

Para o efeito, foi necessário configurar porta eth1 do computador 4 com um IP na mesma gama que o computador 2. De seguida, adicionaram-se as rotas corretas aos computadores com o comando *route add*. No computador 1 adicionou-se a rota que indica que os pacotes devem ser reencaminhados para o endereço IP do computador 4, sendo feito o mesmo para o computador 2, mas com IP do computador 4 da sub-rede 1.

Assim, é possível fazer *ping* do computador 2 para o 1. Os pedidos são reencaminhados para o computador 4 que está ligado à sub-rede de ambos os computadores, conseguindo comunicar com os dois.

Analisando os *logs*, é possível perceber que, quando é feito um ping do computador 1 ao computador 2, o pacote ICMP contém como endereço de destino, o endereço MAC do computador 4. Na resposta do computador 2, o pacote contém como endereço de origem o endereço MAC do computador 4.

Para além do que foi feito na experiência 2, foi também preciso adicionar a porta do *switch* ligada a eth1 à VLAN com o computador 2, utilizando os comandos de configuração do *switch*.

## Configurar um *router* comercial e implementar NAT

O objetivo da experiência foi configurar um *router* comercial com NAT implementado.

O NAT – *Network Address Translation* – possibilita que os computadores de uma rede interna, como as que foram criadas, tenham acesso ao exterior. Para tal, é gerado um número de 16 bits, utilizando-se uma *hash table*, e escrevendo-o no campo da porta de origem. Na resposta, realiza-se o processo inverso, para que o *router* saiba para qual computador deve enviar a resposta.

Configurou-se o *router* definindo as rotas internas e externas com o comando *ip route* na consola de configuração do *router*. De seguida, definiu-se o computador 4 como *default gateway* do computador 1 e o *router* como *default gateway* dos computadores 2 e 4. Assim, os pacotes enviados pelo computador 1 seguem para o computador 4 e depois para o *router* ou para o computador 2.

## DNS

O objetivo desta experiência foi configurar o *Domain Name System* (DNS), para ser possível aceder a redes externas.

Para configurar o DNS, basta editar o ficheiro *resolv.conf*, indicando os parâmetros do DNS fornecido no guião.

Quando se faz *ping* a um servidor externo, é enviado um pacote de DNS que pede o IP do servidor. Em resposta, chega outro pacote DNS que contém a informação pedida.

## Ligações TCP

Depois de a rede estar completamente configurada, procedeu-se ao teste da aplicação desenvolvida.

O teste foi feito com recurso à transferência de um ficheiro através de um servidor FTP. A transferência foi bem-sucedida, mostrando que a configuração da rede foi feita corretamente.

O *Transmission Control Protocol* (TCP) utiliza o mecanismo *Automatic Repeat Request* (ARQ) que é um método de controlo de erros na transmissão de dados que utiliza *acknowledgments* (mensagens enviadas pelo recetor indicando que a trama de dados foi recebida corretamente) e *timeouts* (tempo permitido para esperar por um *acknowledgment*), de forma a garantir uma transmissão confiável através serviço não confiável. Se não for recebido um *acknowledgment* antes do *timeout*, a trama é retransmitida até ser recebido um *acknowledgment*.

Para fazer o controlo de congestão, o TCP mantém uma janela de congestão que consiste numa estimativa do número de octetos que a rede consegue encaminhar, não enviando mais octetos do que o mínimo da janela definida pelo recetor e pela janela de congestão.

Uma vez que a taxa de transferência é distribuída de forma igual para cada ligação, a transferência de dados em simultâneo leva a uma queda na taxa de transmissão.

# Conclusão

Com o desenvolvimento deste trabalho, foi possível interiorizar os conceitos necessários e perceber melhor como funciona algo que está presente no dia-a-dia de todos.

Por outro lado, a aplicação de *donwload* e o seu desenvolvimento permitiu-nos perceber como funcionam as transferências por FTP e o próprio protocolo.

Podemos concluir que o projeto foi terminado com sucesso, visto que o grupo conseguiu implementar tudo o que era proposto, dando uma perspetiva diferente de como funcionam os dispositivos utilizados.

# Referências

1. Address Resolution Protocol, https://en.wikipedia.org/wiki/Address\_Resolution\_Protocol
2. Transmission Control Protocol, https://en.wikipedia.org/wiki/Transmission\_Control\_Protocol
3. Automatic repeat request, https://en.wikipedia.org/wiki/Automatic\_repeat\_request
4. Controlo de congestão,  
   http://www.gsd.inesc-id.pt/~ler/docencia/prd0304/handouts09.pdf

# Anexos

## Código da aplicação

main.c

#include "utilities.h"

#include "url.h"

#include "connection.h"

int main **(**int argc**,** char**\*\*** argv**)** **{**

**if** **(**argc **!=** 3**)** **{**

printf**(**"\nUsage Error!\n"**);**

printf**(**"\n\nUsage:\n"**);**

printf**(**"| name |\t\t url\t\t\t| debug mode |\n"**);**

printf**(**" ./app ftp://[<user>:<password>@]<host>/<url-path> <ON/OFF>\n\n"**);**

**return** ERROR**;**

**}**

**if** **(**strcmp**(**argv**[**2**],** "ON"**)** **!=** 0 **&&** strcmp**(**argv**[**2**],** "OFF"**)** **!=** 0**)** **{**

printf**(**"Error!\n"**);**

printf**(**"\nPlease give a valid debug mode: ON/OFF\n"**);**

**return** ERROR**;**

**}**

int debug\_mode **=** strcmp**(**argv**[**2**],** "ON"**)** **==** 0 **?** 1 **:** 0**;**

debug\_msg**(**debug\_mode**,** "Initializing a url struct."**);**

url **\***new\_url **=** malloc**(sizeof(**url**));**

**if** **(**init\_url**(**new\_url**,** argv**[**1**],** debug\_mode**)** **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Url struct initialized.\n"**);**

debug\_msg**(**debug\_mode**,** "Getting ip by host name."**);**

connection **\*** connectionA **=** malloc**(sizeof(**connection**));**

**if** **(**get\_ip**(**connectionA**,** new\_url**,** debug\_mode**)** **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Host is valid, and it returned a valid ip.\n"**);**

char **\*** host\_info **=** malloc**(**30 **\*** **sizeof(**char**));**

strcpy**(**host\_info**,** "Connecting to A '"**);**

strcat**(**host\_info**,** connectionA**->**ip**);**

strcat**(**host\_info**,** "' through port "**);**

char portA\_str**[**15**];**

sprintf**(**portA\_str**,** "%d"**,** connectionA**->**port**);**

strcat**(**host\_info**,** portA\_str**);**

strcat**(**host\_info**,** "..."**);**

debug\_msg**(**debug\_mode**,** host\_info**);**

connectionA**->**fd **=** connect\_to**(**connectionA**->**ip**,** connectionA**->**port**,** debug\_mode**);**

**if** **(**connectionA**->**fd **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Connection was successfull.\n"**);**

debug\_msg**(**debug\_mode**,** "Logging in."**);**

**if** **(**log\_in\_host**(**connectionA**,** new\_url**,** debug\_mode**)** **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Logged in messages were sent.\n"**);**

debug\_msg**(**debug\_mode**,** "Entering passive mode..."**);**

connection **\*** connectionB **=** malloc**(sizeof(**connection**));**

**if** **(**pasv\_host**(**connectionA**,** new\_url**,** debug\_mode**,** connectionB**)** **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Completed!\n"**);**

char **\*** data\_host\_info **=** malloc**(**30 **\*** **sizeof(**char**));**

strcpy**(**data\_host\_info**,** "Connecting to B '"**);**

strcat**(**data\_host\_info**,** connectionA**->**ip**);**

strcat**(**data\_host\_info**,** "' through port "**);**

char portB\_str**[**15**];**

sprintf**(**portB\_str**,** "%d"**,** connectionB**->**port**);**

strcat**(**data\_host\_info**,** portB\_str**);**

strcat**(**data\_host\_info**,** "..."**);**

debug\_msg**(**debug\_mode**,** data\_host\_info**);**

connectionB**->**fd **=** connect\_to**(**connectionB**->**ip**,** connectionB**->**port**,** debug\_mode**);**

**if** **(**connectionB**->**fd **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Connected!\n"**);**

debug\_msg**(**debug\_mode**,** "Sending path..."**);**

**if** **(**def\_path**(**connectionA**,** new\_url**->**path**,** debug\_mode**)** **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**debug\_mode**,** "Path was sent!\n"**);**

debug\_msg**(**1**,** "Downloading file..."**);**

**if** **(**download\_from\_host**(**connectionB**,** new\_url**->**path**,** debug\_mode**)** **==** ERROR**)**

**return** ERROR**;**

debug\_msg**(**1**,** "Download completed...\n"**);**

debug\_msg(debug\_mode, "Disconnecting...");

if (disconnect\_host(connectionA, new\_url, debug\_mode) == ERROR)

return ERROR;

debug\_msg(debug\_mode, "Diconnected.\n");

return OK;

}

connection.h

#pragma once

#include "utilities.h"

#include "url.h"

**typedef** struct Connection **{**

int fd**;**

char **\*** ip**;**

int port**;**

**}** connection**;**

int get\_ip**(**connection **\*** connection**,** url**\*** url**,** int debug\_mode**);**

int connect\_host**(**connection **\*** connection**,** url **\*** url**,** int debug\_mode**);**

int connect\_to**(**char **\*** ip**,** int port**,** int debug\_mode**);**

int log\_in\_host**(**connection **\*** connection**,** url **\*** url**,** int debug\_mode**);**

int pasv\_host**(**connection **\*** connectionA**,** url **\*** url**,** int debug\_mode**,** connection **\*** connectionB**);**

int get\_pasv\_from\_host**(**int connection\_fd**,** char**\*** ip**,** int **\*** port**,** int debug\_mode**);**

int def\_path**(**connection **\*** connectionA**,**char **\*** path**,** int debug\_mode**);**

int download\_from\_host**(**connection **\*** connectionB**,** char**\*** path**,** int debug\_mode**);**

int disconnect\_host**(**connection **\*** connectionA**,** url **\*** url**,** int debug\_mode**);**

int send\_to\_host**(**int connection\_fd**,** const char**\*** msg**);**

int read\_from\_host**(**int connection\_fd**,** char**\*** msg**,** int debug\_mode**,** char **\*** code**);**

connection.c

#include "connection.h"

int get\_ip**(**connection **\*** connection**,** url**\*** url**,** int debug\_mode**)** **{**

struct hostent**\*** h**;**

debug\_sub\_msg**(**debug\_mode**,** "Getting host ip by name..."**);**

**if** **((**h **=** gethostbyname**(**url**->**host**))** **==** **NULL)** **{**

herror**(**"Error, could not execute gethostbyname()"**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Completed!"**);**

char**\*** ip **=** inet\_ntoa**(\*((**struct in\_addr **\*)** h**->**h\_addr**));**

connection**->**ip **=** malloc**(**strlen**(**ip**));**

strcpy**(**connection**->**ip**,** ip**);**

connection**->**port **=** 21**;**

**return** OK**;**

**}**

int connect\_to**(**char **\*** ip**,** int port**,** int debug\_mode**)** **{**

struct sockaddr\_in server\_addr**;**

bzero**((**char**\*)&(**server\_addr**),sizeof((**server\_addr**)));**

server\_addr**.**sin\_family **=** AF\_INET**;**

server\_addr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**ip**);**

server\_addr**.**sin\_port **=** htons**(**port**);**

debug\_sub\_msg**(**debug\_mode**,** "Creating a socket..."**);**

int fd**;**

**if** **((**fd **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **<** 0**)** **{**

perror**(**"\t->Error, could not execute socket()"**);**

exit**(**ERROR**);**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Socket created."**);**

debug\_sub\_msg**(**debug\_mode**,** "Connecting..."**);**

**if** **(**connect**(**fd**,** **(**struct sockaddr **\*)&(**server\_addr**),** **sizeof(**server\_addr**))** **<** 0**){**

perror**(**"\t->Error, could not execute connect()"**);**

exit**(**ERROR**);**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Connected!"**);**

**if** **(**port **==** 21**)** **{**

char **\*** msg **=** malloc**(**5 **\*** **sizeof(**char**));**

debug\_sub\_msg**(**debug\_mode**,** "Receiving response message..."**);**

**if** **(**read\_from\_host**(**fd**,** msg**,** debug\_mode**,** "220"**)** **==** ERROR**)** **{**

printf**(**"\nNot a valid connect message!\n\n"**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Message received!"**);**

free**(**msg**);**

**}**

**return** fd**;**

**}**

int log\_in\_host**(**connection **\*** connection**,** url **\*** url**,** int debug\_mode**)** **{**

char **\*** user **=** malloc**(sizeof(**url**->**user**)** **+** 5 **\*** **sizeof(**char**));**

sprintf**(**user**,** "user %s\r\n"**,** url**->**user**);**

debug\_sub\_msg**(**debug\_mode**,** "Sending user to host..."**);**

**if** **(**send\_to\_host**(**connection**->**fd**,** user**)** **==** ERROR**)** **{**

printf**(**"\t->Error sending a message to host."**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "User sent!"**);**

debug\_sub\_msg**(**debug\_mode**,** "Receiving message from host..."**);**

**if** **(**read\_from\_host**(**connection**->**fd**,** user**,** debug\_mode**,** "331"**)** **==** ERROR**)** **{**

printf**(**"\nNot a valid user message!\n\n"**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Message received!"**);**

char **\*** password **=** malloc**(sizeof(**url**->**password**)** **+** 5 **\*** **sizeof(**char**));**

sprintf**(**password**,** "pass %s\r\n"**,** url**->**password**);**

debug\_sub\_msg**(**debug\_mode**,** "Sending password to host..."**);**

**if** **(**send\_to\_host**(**connection**->**fd**,** password**)** **==** ERROR**)** **{**

printf**(**"\t->Error sending a message to host."**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Password sent!"**);**

debug\_sub\_msg**(**debug\_mode**,** "Receiving message from host..."**);**

**if** **(**read\_from\_host**(**connection**->**fd**,** password**,** debug\_mode**,** "230"**)** **==** ERROR**)** **{**

printf**(**"\nLog in failed!\n\n"**);**

**return** ERROR**;**

**}**

free**(**password**);**

debug\_sub\_msg**(**debug\_mode**,** "Message received!"**);**

**return** OK**;**

**}**

int pasv\_host**(**connection **\*** connectionA**,** url **\*** url**,** int debug\_mode**,** connection **\*** connectionB**)** **{**

char **\*** pasv **=** malloc**(**7 **\*** **sizeof(**char**));**

sprintf**(**pasv**,** "pasv \r\n"**);**

debug\_sub\_msg**(**debug\_mode**,** "Sending password to host..."**);**

**if** **(**send\_to\_host**(**connectionA**->**fd**,** pasv**)** **==** ERROR**)** **{**

printf**(**"\t->Error sending a message to host."**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Password sent!"**);**

debug\_sub\_msg**(**debug\_mode**,** "Interpreting passive message from host..."**);**

char **\*** ip **=** malloc**(**50 **\*** **sizeof(**char**));**

int port**;**

**if(** get\_pasv\_from\_host**(**connectionA**->**fd**,** ip**,** **&**port**,** debug\_mode**)** **<** 0 **)** **{**

printf**(**"\t->Error interpreting passive message.\n"**);**

**return** ERROR**;**

**}**

char **\*** ip\_info **=** malloc**(**1024 **\*** **sizeof(**char**));**

strcpy**(**ip\_info**,** "Interpreted IP: \0"**);**

strcat**(**ip\_info**,** ip**);**

debug\_sub\_msg**(**debug\_mode**,** ip\_info**);**

char **\*** port\_info **=** malloc**(**1024 **\*** **sizeof(**char**));**

**if** **(**sprintf**(**port\_info**,** "Interpreted Port: %d"**,** port**)** **<** 0**)** **{**

printf**(**"\t->Error printing port to string.\n"**);**

**return** ERROR**;**

**}**

connectionB**->**ip **=** ip**;**

connectionB**->**port **=** port**;**

debug\_sub\_msg**(**debug\_mode**,** port\_info**);**

debug\_sub\_msg**(**debug\_mode**,** "Completed!"**);**

**return** OK**;**

**}**

int def\_path**(**connection **\*** connectionA**,** char **\*** path**,** int debug\_mode**)** **{**

char **\*** retr **=** malloc**(**1024 **\*** **sizeof(**char**));**

sprintf**(**retr**,** "retr %s\r\n"**,** path**);**

char **\*** path\_info **=** malloc**(**1024 **\*** **sizeof(**char**));**

sprintf**(**path\_info**,** "File: %s"**,** path**);**

debug\_sub\_msg**(**debug\_mode**,** path\_info**);**

debug\_sub\_msg**(**debug\_mode**,** "Sending 'retr' command to host..."**);**

**if** **(**send\_to\_host**(**connectionA**->**fd**,** retr**)** **==** ERROR**)** **{**

printf**(**"\t->Error sending a message to host."**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Command sent!"**);**

debug\_sub\_msg**(**debug\_mode**,** "Receiving message from host..."**);**

**if** **(**read\_from\_host**(**connectionA**->**fd**,** retr**,** debug\_mode**,** "150"**)** **==** ERROR**)** **{**

printf**(**"\nPath is not valid!\n\n"**);**

**return** ERROR**;**

**}**

free**(**retr**);**

free**(**path\_info**);**

debug\_sub\_msg**(**debug\_mode**,** "Message received!"**);**

**return** OK**;**

**}**

int disconnect\_host**(**connection **\*** connectionA**,** url **\*** url**,** int debug\_mode**)** **{**

char **\*** quitA **=** malloc**(**6 **\*** **sizeof(**char**));**

sprintf**(**quitA**,** "quit\r\n"**);**

debug\_sub\_msg**(**debug\_mode**,** "Sending 'quit' command to host..."**);**

**if** **(**send\_to\_host**(**connectionA**->**fd**,** quitA**)** **==** ERROR**)** **{**

printf**(**"\t->Error sending a message to host A."**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Closing socket..."**);**

**if** **(**connectionA**->**fd**)** **{**

close**(**connectionA**->**fd**);**

free**(**connectionA**);**

**}**

free**(**url**);**

**return** OK**;**

**}**

int send\_to\_host**(**int connection\_fd**,** const char**\*** msg**)** **{**

int written\_bytes **=** 0**;**

written\_bytes **=** write**(**connection\_fd**,** msg**,** strlen**(**msg**));**

int return\_value **=** **(**written\_bytes **==** strlen**(**msg**))** **?** OK **:** ERROR**;**

**return** return\_value**;**

**}**

int read\_from\_host**(**int connection\_fd**,** char**\*** msg**,** int debug\_mode**,** char **\*** code**)** **{**

FILE**\*** fp **=** fdopen**(**connection\_fd**,** "r"**);**

int size **=** 4**;**

**if(**debug\_mode**)**

printf**(**"\t->Message Code: "**);**

**do** **{**

memset**(**msg**,** 0**,** size**);**

msg **=** fgets**(**msg**,** size**,** fp**);**

**if(**debug\_mode**)**

printf**(**"%s"**,** msg**);**

**}** **while** **(!(**'1' **<=** msg**[**0**]** **&&** msg**[**0**]** **<=** '5'**));**

**if(**debug\_mode**)**

printf**(**"\n"**);**

**if** **(**strcmp**(**msg**,** code**)** **!=** 0**)** **{**

char **\*** error\_msg **=** malloc**(**1024 **\*** **sizeof(**char**));**

strcpy**(**error\_msg**,** "\n\nError!! It was suposed to receive a message with the '"**);**

strcat**(**error\_msg**,** code**);**

strcat**(**error\_msg**,** "' code, and a it was received a message with the '"**);**

strcat**(**error\_msg**,** msg**);**

strcat**(**error\_msg**,** "' code...\n\n"**);**

**if(**debug\_mode**)**

printf**(**error\_msg**,** "\n\nError! You received a wrong code message\n"**);**

**return** ERROR**;**

**}**

**return** OK**;**

**}**

int get\_pasv\_from\_host**(**int connection\_fd**,** char**\*** ip\_str**,** int **\*** port**,**int debug\_mode**)** **{**

FILE**\*** fp **=** fdopen**(**connection\_fd**,** "r"**);**

int size **=** 1024**;**

char **\*** msg **=** malloc**(**size **\*** **sizeof(**char**));**

**if(**debug\_mode**)**

printf**(**"\t->Passive Message: "**);**

**do** **{**

memset**(**msg**,** 0**,** size**);**

msg **=** fgets**(**msg**,** size**,** fp**);**

**if(**debug\_mode**)**

printf**(**"%s"**,** msg**);**

**}** **while** **(!(**'1' **<=** msg**[**0**]** **&&** msg**[**0**]** **<=** '5'**));**

int ip**[**4**];**

int port\_arr**[**2**];**

**if** **((**sscanf**(**msg**,** "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)"**,**

**&**ip**[**0**],** **&**ip**[**1**],** **&**ip**[**2**],** **&**ip**[**3**],** **&**port\_arr**[**0**],** **&**port\_arr**[**1**]))** **<** 0**)**

**return** ERROR**;**

**if** **(**sprintf**(**ip\_str**,** "%d.%d.%d.%d"**,** ip**[**0**],** ip**[**1**],** ip**[**2**],** ip**[**3**])** **<** 0**)**

**return** ERROR**;**

**\***port **=** 256 **\*** port\_arr**[**0**]** **+** port\_arr**[**1**];**

**return** OK**;**

**}**

int download\_from\_host**(**connection **\*** connectionB**,** char**\*** path**,** int debug\_mode**)** **{**

FILE**\*** file**;**

int bytes**;**

char **\*** filename **=** basename**(**path**);**

debug\_sub\_msg**(**1**,** "Creating file with defined path..."**);**

**if** **(!(**file **=** fopen**(**filename**,** "w"**)))** **{**

printf**(**"ERROR: Cannot open file.\n"**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**1**,** "Created!"**);**

debug\_sub\_msg**(**1**,** "Downloading..."**);**

char buf**[**1024**];**

**while** **((**bytes **=** read**(**connectionB**->**fd**,** buf**,** **sizeof(**buf**))))** **{**

**if** **(**bytes **<** 0**)** **{**

printf**(**"ERROR: Nothing was received from data socket fd.\n"**);**

**return** ERROR**;**

**}**

**if** **((**bytes **=** fwrite**(**buf**,** bytes**,** 1**,** file**))** **<** 0**)** **{**

printf**(**"ERROR: Cannot write data in file.\n"**);**

**return** ERROR**;**

**}**

**}**

debug\_sub\_msg**(**1**,** "Completed!"**);**

**if(**connectionB**->**fd**)** **{**

close**(**connectionB**->**fd**);**

free**(**connectionB**);**

**}**

**if(**file**)**

fclose**(**file**);**

**return** 0**;**

**}**

url.h

#pragma once

**typedef** struct Url **{**

char **\*** user**;**

char **\*** password**;**

char **\*** host**;**

char **\*** path**;**

char **\*** filename**;**

**}** url**;**

int init\_url**(**url **\*** url**,** char **\*** url\_str**,** int debug\_mode**);**

url.c

#include "utilities.h"

#include "url.h"

int init\_url**(**url **\*** url**,** char **\*** url\_str**,** int debug\_mode**)** **{**

debug\_sub\_msg**(**debug\_mode**,** "Getting url strings..."**);**

char **\*** str\_beg **=** malloc**(**6 **\*** **sizeof(**char**));**

memcpy**(**str\_beg**,** url\_str**,** 6**);**

**if** **(**strcmp**(**str\_beg**,** "ftp://\0"**)** **!=** 0**)** **{**

printf**(**"\nError! Please start your url by 'ftp://'...\n"**);**

**return** ERROR**;**

**}**

char **\*\*** sub\_str **=** malloc**(**5 **\*** **sizeof(**char**\*));**

int size **=** strlen**(**url\_str**)** **-** 6**;**

sub\_str**[**0**]** **=** malloc**(**size**);**

memcpy**(**sub\_str**[**0**],** url\_str **+** 6**,** size**);**

sub\_str**[**0**][**size**]** **=** '\0'**;**

url**->**user **=** malloc**(**strlen**(**sub\_str**[**0**]));**

memcpy**(**url**->**user**,** sub\_str**[**0**],** strlen**(**sub\_str**[**0**]));**

strtok**(**url**->**user**,** ":"**);**

**if** **(**strlen**(**url**->**user**)** **==** strlen**(**sub\_str**[**0**]))** **{**

printf**(**"\nError!!! Please declare an user...\n"**);**

**return** ERROR**;**

**}**

size **=** strlen**(**sub\_str**[**0**])** **-** strlen**(**url**->**user**)** **-** 1**;**

sub\_str**[**1**]** **=** malloc**(**size**);**

memcpy**(**sub\_str**[**1**],** sub\_str**[**0**]** **+** strlen**(**url**->**user**)** **+** 1**,** size**);**

sub\_str**[**1**][**size**]** **=** '\0'**;**

url**->**password **=** malloc**(**strlen**(**sub\_str**[**1**]));**

memcpy**(**url**->**password**,** sub\_str**[**1**],** strlen**(**sub\_str**[**1**]));**

strtok**(**url**->**password**,** "@"**);**

**if** **(**strlen**(**url**->**password**)** **==** strlen**(**sub\_str**[**1**]))** **{**

printf**(**"\nError!!! Please declare a password...\n"**);**

**return** ERROR**;**

**}**

size **=** strlen**(**sub\_str**[**1**])** **-** strlen**(**url**->**password**)** **-** 1**;**

sub\_str**[**2**]** **=** malloc**(**size**);**

memcpy**(**sub\_str**[**2**],** sub\_str**[**1**]** **+** strlen**(**url**->**password**)** **+** 1**,** size**);**

sub\_str**[**2**][**size**]** **=** '\0'**;**

url**->**host **=** malloc**(**strlen**(**sub\_str**[**2**]));**

memcpy**(**url**->**host**,** sub\_str**[**2**],** strlen**(**sub\_str**[**2**]));**

strtok**(**url**->**host**,** "/"**);**

**if** **(**strlen**(**url**->**host**)** **==** strlen**(**sub\_str**[**2**]))** **{**

printf**(**"\nError! Please declare a host...\n"**);**

**return** ERROR**;**

**}**

size **=** strlen**(**sub\_str**[**2**])** **-** strlen**(**url**->**host**)** **-** 1**;**

sub\_str**[**3**]** **=** malloc**(**size**);**

memcpy**(**sub\_str**[**3**],** sub\_str**[**2**]** **+** strlen**(**url**->**host**)** **+** 1**,** size**);**

sub\_str**[**3**][**size**]** **=** '\0'**;**

url**->**path **=** malloc**(**strlen**(**sub\_str**[**3**]));**

size **=** strlen**(**sub\_str**[**3**]);**

memcpy**(**url**->**path**,** sub\_str**[**3**],** size**);**

url**->**path**[**size**]** **=** '\0'**;**

**if** **(!**strlen**(**url**->**path**))** **{**

printf**(**"\nError! Please declare a path...\n"**);**

**return** ERROR**;**

**}**

debug\_sub\_msg**(**debug\_mode**,** "Completed!"**);**

**return** OK**;**

**}**

utilities.h

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <unistd.h>

#include <signal.h>

#include <libgen.h>

#define ERROR -1

#define OK 0

void debug\_msg **(**int debug\_mode**,** char **\*** msg**);**

void debug\_sub\_msg **(**int debug\_mode**,** char **\*** msg**);**

utilities.c

#include <stdio.h>

void debug\_msg **(**int debug\_mode**,** char **\*** msg**)** **{**

**if** **(**debug\_mode**)** **{**

printf**(**msg**,** "Error printing msg, missing argument!"**);**

printf**(**"\n"**);**

**}**

**}**

void debug\_sub\_msg **(**int debug\_mode**,** char **\*** msg**)** **{**

**if** **(**debug\_mode**)** **{**

printf**(**"\t->"**);**

printf**(**msg**,** "\t->Error printing msg, missing argument!"**);**

printf**(**"\n"**);**

**}**

**}**

## Comandos de configuração

Computadores

#!/bin/bash

**ifconfig** eth0 up 172**.**16**.**10**.**1**/**24

route add **-**net 172**.**16**.**11**.**0**/**24 gw 172**.**16**.**10**.**254

route add default gw 172**.**16**.**10**.**254

**printf** "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" **>** **/**etc**/**resolv.conf

**echo** "tux1 configured"

#!/bin/bash

**ifconfig** eth0 up 172**.**16**.**11**.**1**/**24

route add **-**net 172**.**16**.**10**.**0**/**24 gw 172**.**16**.**11**.**253

route add default gw 172**.**16**.**11**.**254

**printf** "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" **>** **/**etc**/**resolv.conf

**echo** "tux2 configured"

#!/bin/bash

**ifconfig** eth0 up 172**.**16**.**10**.**254**/**24

**ifconfig** eth1 up 172**.**16**.**11**.**253**/**24

route add default gw 172**.**16**.**11**.**254

**echo** 1 **>** **/**proc**/**sys**/**net**/**ipv4**/**ip\_forward

**echo** 0 **>** **/**proc**/**sys**/**net**/**ipv4**/**icmp\_echo\_ignore\_broadcasts

**printf** "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" **>** **/**etc**/**resolv.conf

**echo** "tux4 configured"

*Router*

conf t

interface gigabitethernet 0/0

ip address 172.16.11.254 255.255.255.0

no shutdown

ip nat inside

exit

interface gigabitethernet 0/1

ip address 172.16.1.19 255.255.255.0

no shutdown

ip nat outside

exit

ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix 24

ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.10.0 0.0.0.7

access-list 1 permit 172.16.11.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254

ip route 172.16.10.0 255.255.255.0 172.16.11.253

end

*Switch*

conf t

vlan 10

end

conf t

vlan 11

end

conf t

interface fastethernet 0/1

switchport mode access

switchport access vlan 10

end

conf t

interface fastethernet 0/3

switchport mode access

switchport access vlan 10

end

conf t

interface fastethernet 0/2

switchport mode access

switchport access vlan 11

end

conf t

interface fastethernet 0/4

switchport mode access

switchport access vlan 11

end

conf t

interface gigabitethernet 0/1

switchport mode access

switchport access vlan 20

end

## Logs

Os logs encontram-se na pasta *logs* enviada em conjunto com este relatório, à exceção do *log* da experiência 6, devido ao seu tamanho. Por esse motivo, apresenta-se, de seguida, um excerto desse *log*.

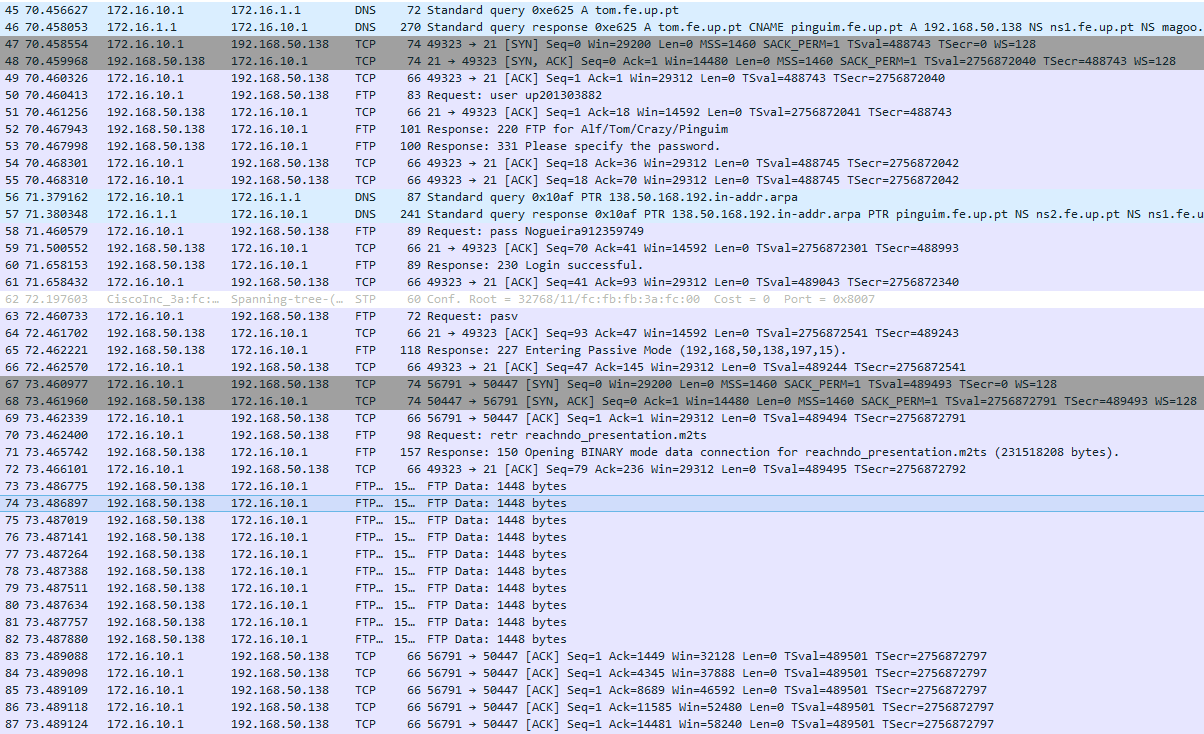


Figura 9 - Excerto do log da experiência 6 (captura no computador 1)