**Relatório:**

**2º Trabalho Laboratorial**

Licenciatura em Engenharia Informática e computação

Faculdade de Engenharia da Universidade do Porto

No âmbito da unidade curricular: **Redes de computadores**

Trabalho realizado por:

* José Leandro Rodrigues Silva – up202008061
* José Pedro Teixeira Ramos - up202005460

**Índice**

[**Sumário**](#_heading=h.lkd4zg9tjf3p) **2**

[**Introdução**](#_heading=h.k1xf651jg9nu) **2**

[**1st Part – Download Application**](#_heading=h.scyvlpx4o4xb) **3**

[Description](#_heading=h.vnwz3ggmym90) 3

[Architecture](#_heading=h.l38zafkinvlv) 4

[**2st Part – Configuration and Study of a Computer network**](#_heading=h.b9hwvsa55kmw) **6**

[Experiência 1 - Configure an IP network](#_heading=h.elji3v85yfd5) 6

[Principais comandos usados:](#_heading=h.r31l2371vzmp) 6

# Sumário

# Introdução

No âmbito da unidade curricular, Redes de Computadores, do 3º ano do curso de Licenciatura em Engenharia Informática e Computação (LEIC) no ano letivo 2022/23, foi realizado um 2º trabalho prático. Este relatório servirá então para descrever, discutir e partilhar o trabalho desenvolvido durante e fora das aulas práticas para este trabalho prático. Este trabalho consiste em 2 partes lógicas, numa primeira parte desenvolveu-se uma aplicação para download de ficheiros usando FTP e na segunda parte realizaram-se várias experiências em laboratório para configurar e analisar uma rede de computadores.

# 1st Part – Download Application

## 

## Description

Nesta parte do trabalho prático foi desenvolvida uma aplicação para download de ficheiros usando FTP (File Transfer Protocol). Para desenvolver esta aplicação utilizou-se a linguagem C e foi necessário um trabalho prévio de preparação para primeiro compreender a fundo como funciona o protocolo FTP e depois como conseguir implementar este protocolo usando conexões a sockets TCP.

Ao desenvolver este trabalho foi definido que este deveria dar alguma liberdade ao utilizador no que toca à sintaxe dos comandos (login); deveria ser resistente a erros e no caso destes acontecerem deveria-se fornecer explicações para a sua ocorrência na linha de comandos; deveria, também, fazer algumas verificações tanto de dados recebidos e dados enviados (IP e número de bytes); por fim as funções construídas deveriam ser o mais gerais possíveis para o futuro reaproveitamento e reutilização do código desenvolvido.

Para utilizar a aplicação, dá-se como argumento um comando com a seguinte formatação:

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

Como se pode verificar, o login é opcional, e a sua presença é identificada pela presença do caráter ‘@’ e na sua ausência assume-se as credenciais default:

User: “anonymous” | Pass: “”

Adicionalmente assumiu-se que na omissão que qualquer uma das credenciais é utilizada a credencial default para a credencial em falta.

Para testar a aplicação utilizaram-se os seguintes comandos:

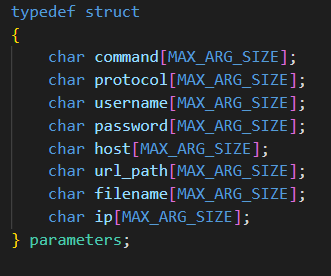
* download ftp://ftp.up.pt/pub/kodi/timestamp.txt
* download ftp://anonymous:@ftp.up.pt/pub/parrot/index.db
* download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4

## Architecture

As funções desenvolvidas/utilizadas para esta aplicação foram as seguintes:

* int parse\_arguments(char \*\*args, parameters\* params)
* int getIP(char \*host, char \*host\_ip)
* int open\_connect\_TCP\_socket(int \*sockfd, char \*server\_ip, int port)
* int parse\_pasv\_response(char \*response, int \*port, char\* ip)
* int parse\_retr\_response(char \*response, int \*bytes)
* void print\_socket\_response(int sockfd)
* int get\_cmd\_response(int sockfd, char\* response)
* int send\_cmd\_to\_socket(int sockfd, char \*cmd, char \*arg, char\* response)
* int download\_file(int sockfd, char\* filename)

Foi também utilizada a struct:

****

Esta estrutura contém todos os dados necessários para completar o download de um ficheiro e grande parte destes argumentos são dados no comando de argumento

Passo agora a explicar cada um destas funções tendo em conta o workflow da aplicação e a utilidade de cada uma.

Num primeiro momento é necessário dar parse do comando enviado como argumento e para isso é utilizada a função **parse\_arguments** que recebe esse comando, guarda os dados relevantes numa struct params e, no caso de erro devolve o valor 1.

Tendo o nome do host, é necessário obter o seu IP address e para o efeito utiliza-se a função **getIP**, que calcula o endereço IP para o host name dado no argumento. Esta função foi reaproveitada do código de exemplo disponibilizado pelo corpo docente.

Agora pode-se estabelecer uma conexão com um socket TCP utilizando o IP calculado e a porta 21 (porta default para FTP). Este socket (control socket) servirá para controlo, ou seja, será o meio para envio de comandos e receção de resposta a esses comandos. Para abrir esta conexão utiliza-se a função **open\_connect\_TCP\_socket** que também foi reaproveitada do código de exemplo disponibilizado pelo corpo docente. Esta devolve um file descriptor.

A função **print\_socket\_response** é utilizada para imprimir para a linha de comandos a resposta de várias linhas dada inicialmente pelo control socket recentemente aberto.

Agora é necessário enviar comandos para o control socket utilizando a função **send\_cmd\_to\_socket** que recebe duas strings uma com o comando a ser enviado e outra (opcional) com um possível argumento do comando, se não for necessário um argumento, pode ser deixada em branco. Esta função escreve para o control socket o comando e utiliza a função **get\_cmd\_response** para obter a resposta do socket ao comando enviado e imprimi-la para a linha de comandos. O valor retornado corresponde ao response status code que pode ser 5xx, 4xx, 3xx, 2xx, 1xx dependendo do efeito do comando. A resposta do server é também devolvida através dos parâmetros.

Assim para fazer o login bastam duas chamadas à função **send\_cmd\_to\_socket**:

* send\_cmd\_to\_socket(control\_socketfd, "user", params.username, response)
* send\_cmd\_to\_socket(control\_socketfd, "pass", params.password, response)

E para entrar em modo passivo utiliza-se:

* send\_cmd\_to\_socket(control\_socketfd, "pasv", params.password, response)

A resposta ao comando “pasv” deve ser analisada para obter a porta, que será usada para abrir um novo socket que permita a transferência de dados (data socket). Para o efeito é usada a função **parse\_pasv\_response** que constrói um array de bytes a partir da resposta onde os primeiros 4 bytes correspondem ao IP do host e os últimos 2 bytes são utilizados para calcular o nº da porta ( bytes[4]\*256 + bytes[5] ). O IP do host é enviado através dos parâmetros para comparar com o IP obtido, dentro da função, e confirmar a ausência de erros. A porta calculada é devolvida através dos parâmetros.

De seguida a porta calculada é utilizada para abrir um nova conexão com o TCP data socket usando uma chamada à função já referida:

* open\_connect\_TCP\_socket(&data\_socketfd, params.ip, port)

Tendo agora o data socket aberto falta enviar o comando **retr** com o caminho para o ficheiro como argumento (url\_path):

* send\_cmd\_to\_socket(control\_socketfd, "retr", params.url\_path, response)

A resposta do socket a este comando deve também ser analisada para obter o nº de bytes enviados para posteriormente confirmar a integridade do ficheiro e ausência de erros. Para tal é utilizada a função **parse\_retr\_response,** esta função analisa a resposta do socket e devolve o nº de bytes enviados.

Por fim, basta utilizar a função **download\_file** que lê byte a byte os dados enviados para o data socket e armazena-os num ficheiro no mesmo diretório do executável com o mesmo nome do ficheiro a transferir. Essencialmente, realiza o download do ficheiro desejado. O nº de bytes lidos é contabilizado e devolvido pela função. Se o nº de bytes enviados for diferente dos recebidos, então ocorreu algum erro e por isso esta verificação é realizada.

O workflow de chamadas a funções da aplicação (e algumas verificações de erros) é realizado na função main da aplicação.

# 2st Part – Configuration and Study of a Computer network

## Experiência 1 - Configure an IP network

O objetivo principal desta experiência é configurar as máquinas tux3 e tux4 com IPs e máscaras, de maneira a que possam se comunicar entre elas. Também usamos o wireshark para ver como as duas máquinas se comunicam entre si.

### Principais comandos usados:

Linux: ifconfig <carta\_de\_rede> <endIP>/<mascara>;

Linux: route add -net <destino>/<mask> gw <endDestino>;

**Questões:**

**What are the ARP packets and what are they used for?**

ARP ou Address Resolution Protocol é usado para mapear IPs e MAC addresses. Quando um computador tenta enviar um pacote para outro que esteja na mesma rede local, ele vai usar ARP para descobrir qual das outras máquinas tem o IP que ele quer. E quando o ARP chegar ao computador que quer, o destinatário vai enviar outro ARP para a rede que enviou o pedido inicialmente, mas com o seu endereço MAC, para que seja possível fazer a transferência de dados.

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

Quando o tux1(IP - 172.16.40.1; MAC - 00:21:5a:61:2f:d4) tenta enviar pacotes para o tux4(IP - 172.16.40.254; MAC - 00:21:5a:5a:7b:ea), ele vai mandar ARPs em broadcast, ou seja para toda a rede local. O ARP vai ter o seguinte, o IP do tux3 IP - 172.16.40.1, o MAC - 00:21:5a:61:2f:d4, o ip do tux4 IP - 172.16.40.254 e o MAC - 00:00:00:00:00:00, o MAC é assim porque ele não sabe o MAC do tux4 ainda. Quando o tux4 receber esse ARP ele vai enviar o ARP, mas com o seu MAC e não vai ser em broadcast Portanto o ARP tem o IP e MAC da máquina que envia e o IP e MAC da máquina que recebe.

**What packets does the ping command generate?**

O comando ping gera pacotes ARP para saber o enderesso MAC e depois gera comandos ICMP echo request para o endereço especificado, no qual depois vai receber uma resposta de echo reply packet.

**What are the MAC and IP addresses of the ping packets?**

Se fizermos ping do tux3 para o tux4:

Pacote **request**:

* Enderesso IP do emissor: 172.16.40.1
* MAC do emissor: 00:21:5a:61:2f:d4
* Enderesso IP do receptor: 172.16.40.254
* MAC do receptor: 00:21:5a:5a:7b:ea

Pacote **reply**:

* Enderesso IP do emissor: 172.16.40.254
* MAC do emissor: 00:21:5a:5a:7b:ea
* Enderesso IP do receptor: 172.16.40.1
* MAC do receptor: 00:21:5a:61:2f:d4

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

Se a ethernet header for 0x0806 então a frame é um ARP e se for 0x0800 é um IP. Se no IP header estiver 1, então é um ICMP.

**How to determine the length of a receiving frame?**

O tamanho da frame pode ser visto no wireshark.

**What is the loopback interface and why is it important?**

Uma interface de loopback é uma interface virtual que está sempre ativa e acessível, no qual deixa o computador receber respostas de si mesmo. Como resultado, uma interface de loopback é útil para tarefas de debug, pois seu endereço IP sempre pode receber ping.

**Experiência 2**

O objetivo desta experiência foi criar duas bridges//Virtual LAN, uma no tux3 e tux4 e outra no tux2. Os principais pontos de destaque nesta experiência foi a criação e configuração das bridges, e como afeta a ligação entre as outras máquinas.

**Principais comandos usados:**

Linux: ifconfig <carta\_de\_rede> <endIP>/<mascara>;

GTK: /interface bridge add name=bridgeY0

GTK: /interface bridge port remove [find interface =ether1]

GTK: /interface bridge port add bridge=bridgeY0 interface=ether1

**Questões:**

**How to configure bridgeY0?**

Para configurar a brigeY0 primeiro nós temos de criar a bridge, para fazer isto acedemos ao GTK e escrevemos o seguinte comando /interface bridge add name=bridgeY0, de seguida fazemos /interface bridge port remove [find interface =ether1], neste passo é preciso ver a onde o tux3 e tux4 estão ligados no MicroTick, o que este comando faz é eliminar a porta no ether1 para que depois podermos adicionar essa porta à bridge, fazemos isto para a porta do tux3 e tux4. No final adicionamos as portas que eliminamos à bridge com o comando /interface bridge port add bridge=bridgeY0 interface=ether1.

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

Existem 2 domínios de broadcast, quando fazemos broadcast apenas as portas que pertencem à bridge são abrangidas.

**Experiência 3**

**Principais comandos usados:**

**Questões:**

**What routes are there in the tuxes? What are their meaning?**

**What information does an entry of the forwarding table contain?**

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

**What ICMP packets are observed and why?**

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

**Experiência 4**

**Principais comandos usados:**

**Questões:**

**How to configure a static route in a commercial router?**

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

**How to configure NAT in a commercial router?**

**What does NAT do?**

**Experiência 5**

**Principais comandos usados:**

**Questões:**

**How to configure the DNS service at an host?**

**What packets are exchanged by DNS and what information is transported**

**Experiência 6**

**Principais comandos usados:**

**Questões:**

**How many TCP connections are opened by your ftp application?**

**In what connection is transported the FTP control information?**

**What are the phases of a TCP connection?**

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

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

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

**Conclusão**