Faculdade de Engenharia da Universidade do Porto

RCOM - 2º Trabalho Laboratorial

Relatório final

Licenciatura em Engenharia Informática e Computação

Turma 2

Professor: Orangel Contreras

Estudantes:

Francisco Pimentel Serra - up202007723

João Paulo Moreira Araújo - up202004293

Índice

Sumário

O presente relatório incide no trabalho desenvolvido no 2º trabalho da cadeira Redes de Computadores (RCOM) do 3º ano de L.EIC na FEUP, que consistiu no desenvolvimento de uma aplicação que efetua download de ficheiros da internet e na configuração de uma rede local. As experiências descritas neste relatório foram desenvolvidas nos laboratórios da faculdade com posterior análise dos dados obtidos.

Introdução

Foi-nos proposto o desenvolvimento de um projeto em duas partes: a primeira incide sobre o desenvolvimento de uma aplicação de download de ficheiros por FTP. Esta aplicação foi desenvolvida em C e cumpre com os requisitos de transferência de ficheiros; a segunda, incide sobre um conjunto de experiências relativas à configuração de uma rede, tendo estas sido realizadas com sucesso.

Aplicação download

A primeira parte deste trabalho foi o desenvolvimento de uma aplicação para download de ficheiros de acordo com o protocolo FTP (file transfer protocol) descrito no RFC959 e com ligações TCP. Conforme descrito no RFC1738 a nossa aplicação aceita como link a seguinte combinação: ***ftp://[<user>:<password>@]<host>/<url-path>***. No <host> foram testados alguns servidores FTP incluindo o ftp.fe.up.pt com variados paths diferentes para garantir a total funcionalidade desta aplicação. Tivemos que ler alguma documentação dos RFC para entendermos totalmente as informações que nos seriam transmitidas pelos servidores FTP.

Arquitetura

```
typedef struct FTP{
    int control_socket_fd; // file descriptor to control socket
    int data_socket_fd; // file descriptor to data socket
} ftp;
```

```
typedef struct URL {
        char user[MAX_LENGTH]; // string to user
        char password[MAX_LENGTH]; // string to password
        char host[MAX_LENGTH]; // string to host
        char ip[MAX_LENGTH]; // string to IP
        char path[MAX_LENGTH]; // string to path
        char filename[MAX_LENGTH]; // string to filename
        int port; // integer to port
} url;
```

```c
//Parsing functions
void initURL(url* url);
int parseURL(url* url, const char* str); // Parse a string with the url to create the URL structure
int getIpByHost(url* url); // gets an IP by host name
char* processElementUntilChar(char* str, char chr);

//FTP functions
int ftpConnect(ftp* ftp, const char* ip, int port);
int ftpLogin(ftp* ftp, const char* user, const char* password);
int ftpCWD(ftp* ftp, const char* path);
int ftpPasv(ftp* ftp);
int ftpRetr(ftp* ftp, const char* filename);
int ftpDownload(ftp* ftp, const char* filename);
int ftpDisconnect(ftp* ftp);

int ftpSend(ftp* ftp, const char* str, size_t size);
int ftpRead(ftp* ftp, char* str, size_t size);
```

Começamos por fazer o parsing do url inserido ao correr a aplicação usando a função **parseURL** e **processElementUntilChar** que vão guardar os argumentos inseridos na *struct* URL. Depois usando a função **getIpByHost** (dada no moodle) passamos-lhe a struct criada anteriormente e esta retorna o IP. Sabendo este IP utilizamos a função **ftpConnect** para conectar com o socket TCP. Uma vez criada a ligação vamos transmitir os dados do utilizador pela seguinte ordem: ***USER*** (nome do utilizador), ***PASS*** (password do user), ***CWD*** (caminho no servidor para o recurso que pretendemos aceder), ***PASV*** (modo passivo permitindo comunicação bidirecional entre o servidor e o cliente FTP), ***RETR*** (pedido de envio do ficheiro).

Resultados

```
→  redes-de-computadores git:(trabalho2) make
mkdir -p tempfiles
gcc  -g -O3 -Wall -c src/ftp.c -o tempfiles/ftp.c.o
mkdir -p tempfiles
gcc  -g -O3 -Wall -c src/main.c -o tempfiles/main.c.o
→  redes-de-computadores git:(trabalho2) ✗ cd bin
→  bin git:(trabalho2) ✗ ./download ftp://ftp.up.pt/debian/README.html
Parsing command line arguments...

The IP received to ftp.up.pt was 193.137.29.15
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
You are now entering in anonymous mode.
Please insert your college email as password: up202007723@fe.up.pt
Bytes send: 16
Info: USER anonymous

331 Please specify the password.
Bytes send: 27
Info: PASS up202007723@fe.up.pt

230 Login successful.
Bytes send: 13
Info: CWD debian/

250 Directory successfully changed.
Bytes send: 6
Info: PASV
```

```
227 Entering Passive Mode (193,137,29,15,195,94).
IP: 193.137.29.15
PORT: 50014
Bytes send: 18
Info: RETR README.html

150 Opening BINARY mode data connection for README.html (3210 bytes).
226 Transfer complete.
Bytes send: 6
Info: QUIT
```

Como se pode confirmar a aplicação que desenvolvemos funciona como expectável nos dois modos de funcionamento(com ou sem credenciais de início de sessão). Foram também testados variados tipos de ficheros com múltiplos tamanhos.

Configuração e análise de dados

*Nota: Se num endereço de IP estiver presente a letra '**Y**' (por exemplo: 172.16.**Y**0.254), esta é referente ao número bancada de trabalho utilizada no laboratório. Caso esteja presente a letra '**S**' (por exemplo: 172.16.**S**.1), esta é referente ao laboratório em que se realizou a experiência, sendo **1** relativo ao laboratório I321 e **2** ao laboratório I320.*

As respostas às questões em cada experiência têm como base os resultados obtidos nas logs capturadas.

Experiência 1 - Configurar uma rede IP

O objetivo desta experiência incide na configuração de uma ligação entre 2 máquinas, sendo estas *tuxY3* e *tuxY4*. Para tal, foi necessário atribuir um endereço de IP a cada uma delas. No fim foi observado a troca de mensagens entre elas.

Foram utilizados os seguintes comandos para realizar com sucesso esta experiência:

- Para atribuir um endereço de IP a uma máquina: ***ifconfig <interface de rede> <endereço de IP>/<máscara de sub-rede>*** (por exemplo: ifconfig eth0 172.16.60.1/24)
- Para configurar rotas: ***route add -net <endereço de IP da rede>/<máscara de sub-rede> gw <gateway>*** (por exemplo: route add -net 192.168.1.0/24 gw 172.16.4.254)
- Para fazer *ping* entre duas máquinas: ***ping <endereço de IP de destino>*** (por exemplo: ping 172.16.60.1)

Questões

- *O que são pacotes ARP e para que são usados?*

  ARP (Address Resolution Protocol) é um protocolo de rede usado para mapear o endereço de IP de uma máquina para o seu endereço físico (MAC). Quando um dispositivo deseja comunicar com outro dispositivo numa rede local, ele precisa saber o endereço MAC do dispositivo de destino (admitindo que a tabela ARP não tem entradas para o IP da máquina destino). Para tal, o dispositivo de origem envia um pacote ARP que é transmitido por broadcast, para todos os dispositivos da rede local, sendo que este pacote contém o endereço de IP da máquina de destino e pergunta qual dispositivo na rede possui este endereço. Se encontrar o destinatário, este irá enviar um pacote de resposta, que contém o endereço MAC. Desta forma, a

transferência de pacotes pode ser efetuada.

- *Quais são os endereços MAC e IP dos pacotes ARP e porquê?*

  Como inicialmente a tabela ARP foi apagada, quando o *tuxY3* tenta enviar um pacote ao *tuxY4*, o *tuxY3* não sabe qual o MAC address associado ao IP do *tuxY4*, então envia um pacote ARP em broadcast com o seu IP e MAC address como fonte e o IP address do *tuxY4*. Como não sabe o MAC address da máquina com a qual quer comunicar este campo vai vazio (00:00:00:00:00:00). O *tuxY4* ao receber este pacote vai reconhecer o seu IP address e então vai responder com um pacote ARP onde envia o seu IP address e o seu MAC address. Posto isto concluimos que um pacote ARP contém o IP e MAC address do destinatário e da fonte.

- *Que pacotes gera o comando ping?*

  O comando ping gera inicialmente pacotes ARP com o objetivo de identificar o endereço MAC do dispositivo destinatário. De seguida, gera pacotes ICMP (Internet Control Message Protocol) Echo Request para o dispositivo destinatário e espera por pacotes ICMP Echo Reply em retorno.

- *Quais são os endereços MAC e IP dos pacotes ping?*

  Se fizermos um comando ping do *tuxY3* para *tuxY4* são estes os pacotes enviados:

  - Request (*tuxY3* para *tuxY4*)
    - IP address fonte: 172.16.50.1
    - MAC address fonte: 00:21:5a:61:2c:54
    - IP address destinatário: 172.16.50.254
    - MAC address destinatário: 00:22:64:19:09:5c
  - Reply (*tuxY4* para *tuxY3*)
    - IP address fonte: 172.16.50.254
    - MAC address fonte: 00:22:64:19:09:5c
    - IP address destinatário: 172.16.50.1
    - MAC address destinatário: 00:21:5a:61:2c:54

  (Anexo-1)

- *Como se determina se uma frame Ethernet é ARP, IP, ICMP?*

  Para determinar o tipo da frame Ethernet, é necessário analisar o campo EtherType no header da frame : caso o valor seja 0x0806, é do tipo ARP. Caso seja 0x0800, indica que é uma frame do tipo IP, e caso o header de IP seja 0x0001, a frame é do tipo ICMP.

- *Como é que se determina o tamanho de uma frame recebida?*

O comprimento da frame pode ser visualizado no campo Frame Length, que inclui o tamanho total da do header e da payload

- *O que é a interface de loopback e qual é a sua importância?*

  A interface de loopback é uma interface virtual da rede utilizada para se comunicar com o host local. Permite ao computador comunicar consigo próprio, simulando o funcionamento do dispositivo/aplicação numa rede, sem necessidade de testar numa rede externa.

Experiência 2 - Implementação de duas *bridges* num *switch*

O objetivo desta experiência incide na implementação de duas *bridges* num *switch*, pelo que para tal foi necessário configurar primeiramente *tuxY2* de acordo com a experiência passada e de seguida criar duas bridges no switch, adicionando à *bridgeY0 tuxY3* e *tuxY4*, e à *bridgeY1 tuxY2*. Por fim, foi observado a troca de mensagens entre as máquinas.

Foram utilizados os seguintes comandos para realizar com sucesso esta experiência:

- Para criar uma *bridge* → ***/interface bridge add name=<nome da bridge>***
- Para configurar uma porta → ***/interface bridge port add interface=<nome da interface> bridge=<nome da bridge>***

Questões

- *Como configurar a bridgeY0?*

  Para configurar a *bridgeY0*, é necessário ligar previamente, na *rack*, a porta S0 de, por exemplo, *tuxY4*, a RS232 e de seguida ligar de RS232 ao *switch*, através da porta *Console*. Assim, é possível aceder ao switch via a aplicação GTKterm (com o *baudrate* a 115200), onde de seguida se efetua login com as credenciais dadas.

  Para configurar a bridgeY0, é necessário criá-la na switch (através do comando: */interface bridge add name=bridgeY0*) e em seguida remover as portas de tuxYX que estão conectadas à bridge default (através do comando: */interface bridge port remove [find interface=etherXX]*; onde XX representa a porta no switch onde tuxYX está ligado) e por fim adiciona-se essas mesmas portas à bridgeY0 (através do comando: */interface bridge port add interface=etherXX bridge=bridgeY0*).

- *Quantos domínios de broadcast existem? Como é que se pode concluir essa declaração através dos logs?*

  Existem dois domínios de *broadcast* nesta rede, correspondente a cada *bridge* implementada. Ao fazer broadcast em algum dos *tuxYX*, as mensagens de

broadcast apenas serão visíveis pelos dispositivos conectados na *bridge* à qual *tuxYX* pertence.

(Anexo-2)

Experiência 3 - Configuração de um router em Linux

Nesta experiência, pretende-se configurar *tuxY4* de modo a funcionar como *router*, estando ligado à *bridgeY0* e *bridgeY1*, de como a permitir fazer a conexão e transmissão de mensagens entre os dispositivos presentes em cada uma das bridges.

Foram utilizados os seguintes comandos para realizar com sucesso esta experiência:

- Para habilitar IP forwarding em *tuxY4*: ***echo 1 > /proc/sys/net/ipv4/ip-forward***
- Para adicionar como rota default *tuxY4* em *tuxY2* e *tuxY3*: ***route add default gw <endereço de IP de tuxY4>***

Questões

- *Que rotas existem nos tuxes? Quais são os seus significados?*

  Uma vez que *tuxY4* e *tuxY3* estão conectadas à *bridgeY0* (*tuxY4* e *tuxY2* à *bridgeY1*), estas terão rotas geradas automaticamente para ligar as máquinas entre si dentro da mesma bridge.

  Salientando que foi implementado *IP forwarding* em *tuxY4*, adicionou-se em *tuxY3* a rota → *route add default gw 172.16.Y0.254*, que permite enviar mensagens para a *bridgeY1* utilizando como *gateway* o *router tuxY4*, e em *tuxY2* a rota → *route add default gw 172.16.Y1.253*, que permite enviar mensagens para a *bridgeY0* utilizando como *gateway* o *router* tuxY4.

- *Que informações contém uma entrada da forwarding table?*

  Uma entrada na forwarding table contém a seguinte informação: ***Destination Network***, ***Gateway***, ***Interface***. ***Destination Network*** refere-se ao endereço de IP da rede/dispositivo de destino, ***Gateway*** ao endereço de IP do próximo dispositivo que irá dar routing à mensagem para chegar à rede de destino e ***Interface*** à interface local que irá enviar a mensagem para o próximo gateway.

- *Quais mensagens ARP, e endereços MAC associados, são observados e porquê?*

  Na tentativa do tuxY3 dar ping ao tuxY2, é possível verificar inicialmente a mensagem broadcast ARP em que tuxY3 pede por tuxY2, esta contém o endereço

IP e MAC de tuxY3 e o IP de tuxY2. A mensagem é recebida por tuxY4, que reenvia a mensagem por broadcast uma vez que o endereço MAC de tuxY2 ainda não é conhecido. Após este receber a mensagem, envia um pacote ARP de resposta diretamente para tuxY3 (uma vez que as rotas já estão definidas).

Estes são os endereços MAC associados:

- Request (*tuxY3* para *tuxY4*)
    - IP address fonte: 172.16.50.1
    - MAC address fonte: 00:21:5a:61:2c:54
    - IP address destinatário: 172.16.51.1
    - MAC address destinatário: 00:00:00:00:00:00
- Request (*tuxY4* para *tuxY2*)
    - IP address fonte: 172.16.50.1
    - MAC address fonte: 00:21:5a:61:2c:54
    - IP address destinatário: 172.16.51.1
    - MAC address destinatário: 00:00:00:00:00:00
- Reply (*tuxY2* para *tuxY3*)
    - IP address fonte: 172.16.51.1
    - MAC address fonte: 00:22:64:19:09:5c
    - IP address destinatário: 172.16.50.1
    - MAC address destinatário: 00:21:5a:61:2c:54

(Anexo-3)

- *Quais pacotes ICMP são observados e porquê?*

  Após definir as rotas (através do envio de pacotes ARP), são observado pacotes ICMP do tipo *request* e *reply*, indicando que a conexão entre *tuxY3* e *tuxY2* e o envio de informação por estes está bem definida. Caso a rota não fosse bem sucedida, seriam enviados pacotes do tipo Host Unreachable.

  (Anexo-3)

- *Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?*

  Os endereços IP e MAC associados aos pacotes observados são os das respectivas máquinas/interfaces de destino e origem entre *tuxY3* e *tuxY2*. Uma vez que os dispositivos encontram-se em redes diferentes (*bridgeY0* e *bridgeY1*), o pacote ICMP é modificado em *tuxY4*, dado que este faz fowarding dos pacotes entre redes.

Desta forma, os endereços IP e MAC associados aos pacotes ICMP são os seguintes: quando tuxY3 envia o pacote para tuxY2, este vai conter o endereço de IP e MAC de tuxY3, o IP de tuxY2 e MAC da interface eth0 de tuxY4. De seguida, após o pacote chegar a tuxY4, onde é feito forwarding entre redes, os endereços MAC do pacote são modificados para, na origem, a interface eth1 de tuxY4 e, no destino, o endereço MAC de tuxY2.

Quanto ao envio do pacote de resposta por tuxY2, este tem o endereço de IP e MAC, de origem, de tuxY2, IP de destino de tuxY3 e endereço MAC da interface eth1 de tuxY4. Quando este chega a tuxY4, o endereço MAC de origem muda para a interface eth0 de tuxY4 e de destino de tuxY3.

Experiência 4 - Configurar um router comercial e implementar NAT

Esta experiência incide na configuração de um router comercial, ligando-o a bridgeY1, rede local previamente configurada. De seguida, implementou-se o protocolo NAT para comunicar-se com as redes exteriores. O objetivo desta experiência trata-se em compreender a influência do protocolo NAT entre dispositivos locais e a internet.

Foram utilizados os seguintes comandos para realizar com sucesso esta experiência:

- Configuração do router:
  - Configurar os endereços de IP → ***/ip address add address=<endereço de IP>/<máscara> interface=etherXX***
- Configuração do NAT:
  - Desligar o NAT → ***/ip firewall nat disable 0***
  - Configurar o NAT → ***/ip firewall nat add chain=srcnat action=masquerade out-interface=ether1***

Questões

- *Como configurar uma rota estática num router comercial?*

  Inicialmente, é necessário desconectar o cabo que liga as portas RS232 e Console e conectar de RS232 à porta de configuração do router. De seguida, inicia-se a sessão no GTKterm (com o *baudrate* a 115200).

  Para configurar as rotas no router, executa-se o comando → ***ip route add dst-address=<endereço de IP>/<máscara> gateway=<gateway>***.

- *Quais são os caminhos que os pacotes seguem nas experiências e porquê?*

  Caso a rota já esteja definida, os pacotes seguem essa rota, caso contrário, os pacotes serão enviados para o router (uma vez que se definiu que este é a rota

default).

No passo 4, desativou-se os redirecionamentos para tuxY2, ou seja, em caso de necessidade de redirect dentro da mesma rede, não é guardado na forwarding table essa entrada. Em tuxY2, quando se faz ping para tuxY3, uma vez que estes não estão na mesma rede local, a mensagem é enviada ao router Rc. Já este sabe onde está tuxY3, pelo que vai reenviar a mensagem para o tuxY4. Agora em tuxY4, este conhece a rota para tuxY3, pelo que o caminho está concluído. Nos pings seguintes, a rota será a mesma.

Ativando os redirecionamentos, o pacote seguirá, inicialmente, o mesmo caminho dado previamente, contudo, após concluir, irá se adicionar na forwarding table o caminho para se chegar de tuxY2 a tuxY3, pelo que todos os pings seguintes não decorrerão do router Rc, e irão ser direcionados diretamente de tuxY2 para tuxY4.

- *Como configurar o NAT num router comercial?*

  Para configurar o NAT no router utilizado nas experiências, primeiramente desativamos o NAT pré-feito com o comando → ***/ip firewall nat disable 0***.

  De seguida, para a configurar utilizou-se este comando → ***/ip firewall nat add chain=srcnat action=masquerade out-interface=ether1***

- *O que faz o NAT?*

  O NAT (Network Address Translation) é um protocolo que permite que dispositivos de uma rede privada se comuniquem com uma rede externa usando um único endereço de IP público compartilhado.

  Quando um dispositivo dentro da rede privada tenta comunicar com uma rede externa (por exemplo: internet), o pacote é enviado para o router, operado por NAT, onde este modifica o endereço de origem com um endereço público compartilhado, mascarando o remetente originário do pacote, fornecendo uma camada de segurança e privacidade. Quando o pacote retorna ao router, este novamente traduz o endereço de IP para o da rede local, assegurando a correta transmissão de pacotes.

Experiência 5 - DNS

O objetivo desta experiência incide na configuração de um serviço DNS nos dispositivos

locais, de modo a ser possível conectar à internet através da tradução de *hostnames*.

Foram utilizados os seguintes comandos para realizar com sucesso esta experiência:

- Aceder ao ficheiro ***resolv.conf,*** que contém uma lista de nameservers que são usados para traduzir *hostnames*, através do comando → ***sudo nano /etc/resolv.conf***

Questões

- *Como configurar o serviço DNS num host?*

  O serviço DNS é configurado, num *host*, dentro do ficheiro ***resolv.conf,*** que encontra-se na pasta ***/etc/*** do *host*. Para a sua configuração, primeiro adiciona-se o comando com o servidor DNS → ***search netlab.fe.up.pt***; e de seguida com o endereço de IP que o host utiliza para aceder, no caso, à internet → ***nameserver 172.16.1.1***

- *Que pacotes são trocados por DNS, e que informação é transportada?*

  O host envia um pacote com o hostname para o servidor DNS e aguarda por receber um endereço de IP. O servidor envia um pacote de resposta que contém o endereço de IP do hostname (se o encontrar).
  (Anexo-4)

Experiência 6 - Ligações TCP

Questões

- *Quantas conexões TCP são abertas pela aplicação FTP?*
  A aplicação abriu 2 ligações TCP, uma recebia e mandava os comandos FTP para o servidor e a outra recebia os dados enviados pelo servidor.
- *Em qual conexão é transportado o controlo de informação?*

  O controlo de informação é transportado na primeira conexão TCP (na que trata do envio e recepção de comandos).
  (Anexo-5)

- *Quais são as fases de uma conexão TCP?*

Numa conexão TCP, primeiro estabelece-se a conexão, depois ocorre a troca de dados, e depois a conexão é encerrada, havendo assim três fases.

- *Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos logs?*

  O mecanismo ARQ (Automatic Repeat Request) TCP é uma variação do Go-Back-N com a diferença de que o receptor não deixa de processar os frames recebidos quando detecta um erro. O receptor continua a receber as frames seguintes enviando no ACK o número da frame que falhou até a conseguir receber.

Em anexo logs (Anexo-6)

- *Como funciona o mecanismo de congestionamento de controlo TCP? Quais são os campos relevantes? Como é que o throughput da conexão de dados evolui ao longo do tempo? Está de acordo com o mecanismo de congestionamento de controlo TCP?*

  O mecanismo de congestionamento de controlo TCP é um algoritmo que é utilizado para controlar a taxa de transmissão de dados em uma conexão TCP (Transmission Control Protocol). É utilizado para garantir que a conexão não fica sobrecarregada e para evitar a perda de pacotes de dados. Os campos relevantes para o mecanismo de congestionamento de controlo TCP incluem:

  - Janela de congestão (CWND): é um valor que determina a quantidade de dados que um host pode enviar sem receber um ACK (acknowledgment). Quando o host recebe um ACK, a janela de congestão é atualizada e mais dados podem ser enviados.
  - RTT (Round-Trip Time): é o tempo que leva para que um pacote de dados seja enviado para o destino e o ACK seja recebido de volta. Este valor é utilizado para calcular o tamanho da janela de congestão e para determinar o tempo de timeout para a conexão.
  - Slow start: é um algoritmo que é utilizado para aumentar gradualmente a taxa de transmissão de dados ao longo do tempo. Durante o slow start, a janela de congestão é aumentada exponencialmente a cada vez que um ACK é recebido.
  - Aceleração congestionada: é um algoritmo que é utilizado quando a conexão

está sobrecarregada e a janela de congestão precisa ser reduzida. Durante a aceleração congestionada, a janela de congestão é reduzida linearmente a cada vez que um pacote é perdido.

Ao longo do tempo, o throughput da conexão de dados evolui de acordo com o mecanismo de congestionamento de controlo TCP. Quando a conexão está ociosa, o throughput é baixo e aumenta gradualmente à medida que mais dados são enviados. Se a conexão ficar sobrecarregada, o throughput diminui até que a congestão seja controlada e a taxa de transmissão de dados possa ser aumentada novamente.

- *O throughput de uma conexão de dados TCP é perturbado pelo aparecimento de uma segunda conexão TCP? Como?*

  Ao iniciar uma segunda conexão TCP o *throughput* é afetado. Apesar da média de transferência de pacotes se ter mantido ao iniciar uma segunda conexão verificou-se um aumento de decréscimos, levando, assim, a que o download do ficheiro do servidor TCP demora-se mais tempo.

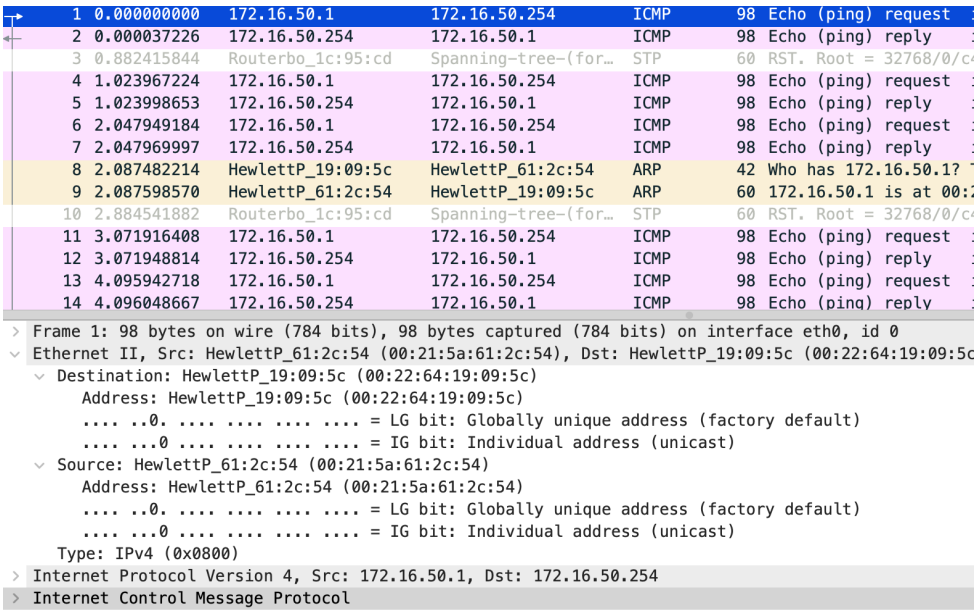
Conclusões

No final, podemos dizer que a aplicação implementada é robusta e transfere ficheiros completamente sem erros. Acreditamos que a implementação abrangente e a compreensão das experiências levaram posteriormente à facilidade de desenvolvimento da aplicação e à compreensão de todos os conceitos relacionados ao projeto. Destaca-se, portanto, que as experiências mostraram-se uma grande fonte de conhecimento, pois envolveu a parte mais complexa e trabalhosa do projeto: Compreender o conceito de "cliente-servidor"; compreender o protocolo de comunicação TCP/IP; compreender o protocolo de comunicação FTP; compreender o serviço DNS.

Referências

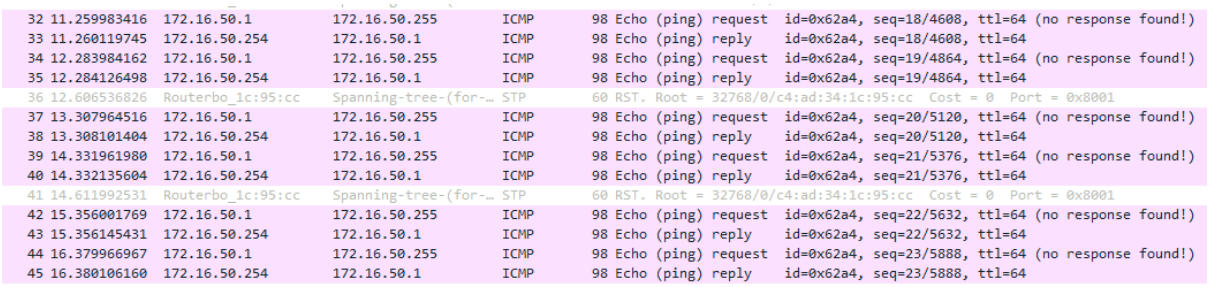
Este trabalho foi desenvolvido com recurso à consulta do material fornecido pelos docentes no moodle. Usamos também a documentação dos RFC 5681 - TCP Congestion Control, RFC959 - File Transfer Protocol, RFC1738 - Uniform Resource Locators encontrada na internet.

Anexos



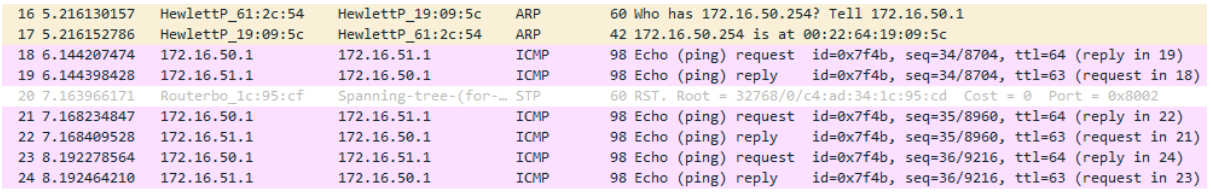
```
    1 0.000000000  172.16.50.1      172.16.50.254     ICMP   98 Echo (ping) request
    2 0.000037226  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply
    3 0.882415844  Routerbo_1c:95:cd Spanning-tree-(for… STP  60 RST. Root = 32768/0/c
    4 1.023967224  172.16.50.1      172.16.50.254     ICMP   98 Echo (ping) request
    5 1.023998653  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply
    6 2.047949184  172.16.50.1      172.16.50.254     ICMP   98 Echo (ping) request
    7 2.047969997  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply
    8 2.087482214  HewlettP_19:09:5c HewlettP_61:2c:54 ARP    42 Who has 172.16.50.1?
    9 2.087598570  HewlettP_61:2c:54 HewlettP_19:09:5c ARP    60 172.16.50.1 is at 00:
   10 2.884541882  Routerbo_1c:95:cd Spanning-tree-(for… STP  60 RST. Root = 32768/0/c
   11 3.071916408  172.16.50.1      172.16.50.254     ICMP   98 Echo (ping) request
   12 3.071948814  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply
   13 4.095942718  172.16.50.1      172.16.50.254     ICMP   98 Echo (ping) request
   14 4.096048667  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
v Ethernet II, Src: HewlettP_61:2c:54 (00:21:5a:61:2c:54), Dst: HewlettP_19:09:5c (00:22:64:19:09:5c
  v Destination: HewlettP_19:09:5c (00:22:64:19:09:5c)
      Address: HewlettP_19:09:5c (00:22:64:19:09:5c)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: HewlettP_61:2c:54 (00:21:5a:61:2c:54)
      Address: HewlettP_61:2c:54 (00:21:5a:61:2c:54)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.16.50.1, Dst: 172.16.50.254
> Internet Control Message Protocol
```

Anexo 1 - Endereço IP e MAC de tuxY3 e tuxY4



```
32 11.259983416  172.16.50.1      172.16.50.255     ICMP   98 Echo (ping) request  id=0x62a4, seq=18/4608, ttl=64 (no response found!)
33 11.260119745  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply    id=0x62a4, seq=18/4608, ttl=64
34 12.283984162  172.16.50.1      172.16.50.255     ICMP   98 Echo (ping) request  id=0x62a4, seq=19/4864, ttl=64 (no response found!)
35 12.284126498  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply    id=0x62a4, seq=19/4864, ttl=64
36 12.606536826  Routerbo_1c:95:cc Spanning-tree-(for-… STP  60 RST. Root = 32768/0/c4:ad:34:1c:95:cc  Cost = 0  Port = 0x8001
37 13.307964516  172.16.50.1      172.16.50.255     ICMP   98 Echo (ping) request  id=0x62a4, seq=20/5120, ttl=64 (no response found!)
38 13.308101404  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply    id=0x62a4, seq=20/5120, ttl=64
39 14.331961980  172.16.50.1      172.16.50.255     ICMP   98 Echo (ping) request  id=0x62a4, seq=21/5376, ttl=64 (no response found!)
40 14.332135604  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply    id=0x62a4, seq=21/5376, ttl=64
41 14.611992531  Routerbo_1c:95:cc Spanning-tree-(for-… STP  60 RST. Root = 32768/0/c4:ad:34:1c:95:cc  Cost = 0  Port = 0x8001
42 15.356001769  172.16.50.1      172.16.50.255     ICMP   98 Echo (ping) request  id=0x62a4, seq=22/5632, ttl=64 (no response found!)
43 15.356145431  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply    id=0x62a4, seq=22/5632, ttl=64
44 16.379966967  172.16.50.1      172.16.50.255     ICMP   98 Echo (ping) request  id=0x62a4, seq=23/5888, ttl=64 (no response found!)
45 16.380106160  172.16.50.254    172.16.50.1       ICMP   98 Echo (ping) reply    id=0x62a4, seq=23/5888, ttl=64
```

Anexo 2 - Ping Broadcast (bridge50)



```
16 5.216130157  HewlettP_61:2c:54  HewlettP_19:09:5c  ARP    60 Who has 172.16.50.254? Tell 172.16.50.1
17 5.216152786  HewlettP_19:09:5c  HewlettP_61:2c:54  ARP    42 172.16.50.254 is at 00:22:64:19:09:5c
18 6.144207474  172.16.50.1        172.16.51.1        ICMP   98 Echo (ping) request  id=0x7f4b, seq=34/8704, ttl=64 (reply in 19)
19 6.144398428  172.16.51.1        172.16.50.1        ICMP   98 Echo (ping) reply    id=0x7f4b, seq=34/8704, ttl=63 (request in 18)
20 7.163966171  Routerbo_1c:95:cf  Spanning-tree-(for-… STP  60 RST. Root = 32768/0/c4:ad:34:1c:95:cd  Cost = 0  Port = 0x8002
21 7.168234847  172.16.50.1        172.16.51.1        ICMP   98 Echo (ping) request  id=0x7f4b, seq=35/8960, ttl=64 (reply in 22)
22 7.168409528  172.16.51.1        172.16.50.1        ICMP   98 Echo (ping) reply    id=0x7f4b, seq=35/8960, ttl=63 (request in 21)
23 8.192278564  172.16.50.1        172.16.51.1        ICMP   98 Echo (ping) request  id=0x7f4b, seq=36/9216, ttl=64 (reply in 24)
24 8.192464210  172.16.51.1        172.16.50.1        ICMP   98 Echo (ping) reply    id=0x7f4b, seq=36/9216, ttl=63 (request in 23)
```

Anexo 3 - Pacote ARP enviado por tuxY3, perguntando por tuxY2. Redirecionado para tuxY4.

1	0.000000000	Routerbo_1c:8b:e4	Spanning-tree-(for…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:e4 Cost = 0 Port = 0
2	0.344976688	172.16.60.1	172.16.1.1	DNS	74	Standard query 0xd203 A www.google.com
3	0.344987234	172.16.60.1	172.16.1.1	DNS	74	Standard query 0x600c AAAA www.google.com
4	0.347563943	172.16.1.1	172.16.60.1	DNS	90	Standard query response 0xd203 A www.google.com A 172.217
5	0.347588946	172.16.1.1	172.16.60.1	DNS	102	Standard query response 0x600c AAAA www.google.com AAAA 2
6	0.347992417	172.16.60.1	172.217.168.164	ICMP	98	Echo (ping) request id=0x0e84, seq=1/256, ttl=64 (reply
7	0.365918470	172.217.168.164	172.16.60.1	ICMP	98	Echo (ping) reply id=0x0e84, seq=1/256, ttl=112 (reque
8	0.366055079	172.16.60.1	172.16.1.1	DNS	88	Standard query 0x69d4 PTR 164.168.217.172.in-addr.arpa
9	0.366545292	172.16.1.1	172.16.60.1	DNS	126	Standard query response 0x69d4 PTR 164.168.217.172.in-add
10	1.349644288	172.16.60.1	172.217.168.164	ICMP	98	Echo (ping) request id=0x0e84, seq=2/512, ttl=64 (reply
11	1.366680779	172.217.168.164	172.16.60.1	ICMP	98	Echo (ping) reply id=0x0e84, seq=2/512, ttl=112 (reque
12	2.002148226	Routerbo_1c:8b:e4	Spanning-tree-(for…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:e4 Cost = 0 Port = 0
13	2.350745254	172.16.60.1	172.217.168.164	ICMP	98	Echo (ping) request id=0x0e84, seq=3/768, ttl=64 (reply
14	2.367790823	172.217.168.164	172.16.60.1	ICMP	98	Echo (ping) reply id=0x0e84, seq=3/768, ttl=112 (reque

Anexo 4 - Registo dos pacotes de um ping feito ao www.google.com

```
TCP        66 52100 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=18598
FTP       100 Response: 220 Welcome to netlab-FTP server
TCP        66 52100 → 21 [ACK] Seq=1 Ack=35 Win=64256 Len=0 TSval=1859
FTP        75 Request: user rcom
TCP        66 21 → 52100 [ACK] Seq=35 Ack=10 Win=65280 Len=0 TSval=269
FTP        67 Request:
TCP        66 21 → 52100 [ACK] Seq=35 Ack=11 Win=65280 Len=0 TSval=269
FTP       100 Response: 331 Please specify the password.
FTP        75 Request: pass rcom
```

Anexo 5 - Pacotes capturados no tuxY1

73	8.414717364	172.16.60.1	192.168.109.136	TCP	68	[TCP Dup ACK 72#1] 53344 → 40102 [ACK] Seq=1 Ack=1449
74	8.414786509	192.168.109.136	172.16.60.1	FTP-D…	1516	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
75	8.414792166	192.168.109.136	172.16.60.1	TCP	1516	[TCP Retransmission] 40102 → 53344 [ACK] Seq=2897 Ack
76	8.414830021	172.16.60.1	192.168.109.136	TCP	68	53344 → 40102 [ACK] Seq=1 Ack=2897 Win=64128 Len=0 TS
77	8.414833304	172.16.60.1	192.168.109.136	TCP	68	[TCP Dup ACK 76#1] 53344 → 40102 [ACK] Seq=1 Ack=2897
78	8.414909712	192.168.109.136	172.16.60.1	FTP-D…	1516	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
79	8.414915649	192.168.109.136	172.16.60.1	TCP	1516	[TCP Retransmission] 40102 → 53344 [ACK] Seq=4345 Ack
80	8.414953923	172.16.60.1	192.168.109.136	TCP	68	53344 → 40102 [ACK] Seq=1 Ack=4345 Win=64128 Len=0 TS
81	8.414957275	172.16.60.1	192.168.109.136	TCP	68	[TCP Dup ACK 80#1] 53344 → 40102 [ACK] Seq=1 Ack=4345
82	8.415032916	192.168.109.136	172.16.60.1	FTP-D…	1516	FTP Data: 1448 bytes (PASV) (retr files/crab.mp4)
83	8.415038782	192.168.109.136	172.16.60.1	TCP	1516	[TCP Retransmission] 40102 → 53344 [PSH, ACK] Seq=579
84	8.415080618	172.16.60.1	192.168.109.136	TCP	68	53344 → 40102 [ACK] Seq=1 Ack=5793 Win=64128 Len=0 TS
85	8.415083831	172.16.60.1	192.168.109.136	TCP	68	[TCP Dup ACK 84#1] 53344 → 40102 [ACK] Seq=1 Ack=5793

Anexo 6 - Pacotes ACK

Código

```c
#include "ftp.h"

static void printUsage(char* argv0);

int main(int argc, char** argv) {
    if (argc != 2) {
        printf("WARNING: Wrong number of arguments.\n");
        printUsage(argv[0]);
        return 1;
    }

    // PARSING PROCESS //
    url url;
    initURL(&url);

    // start parsing argv[1] to URL components
    if (parseURL(&url, argv[1]))
        return -1;

    // edit url ip by hostname
    if (getIpByHost(&url)) {
        printf("ERROR: Cannot find ip to hostname %s.\n", url.host);
        return -1;
    }

    printf("\nThe IP received to %s was %s\n", url.host, url.ip);

    // FTP CLIENT PROCESS //

    ftp ftp;
    ftpConnect(&ftp, url.ip, url.port);

    // Verifying username
    const char* user = strlen(url.user) ? url.user : "anonymous";

    // Verifying password
    char* password;
    if (strlen(url.password)) {
        password = url.password;
    } else {
        char buf[100];
        printf("You are now entering in anonymous mode.\n");
        printf("Please insert your college email as password: ");
        while (strlen(fgets(buf, 100, stdin)) < 14)
            printf("\nIncorrect input, please try again: ");
        password = (char*) malloc(strlen(buf));
        strncat(password, buf, strlen(buf) - 1);
    }
```

```
    // Sending credentials to server
    if (ftpLogin(&ftp, user, password)) {
        printf("ERROR: Cannot login user %s\n", user);
        return -1;
    }

    // Changing directory
    if (ftpCWD(&ftp, url.path)) {
        printf("ERROR: Cannot change directory to the folder of %s\n",url.filename);
        return -1;
    }

    // Entry in passive mode
    if (ftpPasv(&ftp)) {
        printf("ERROR: Cannot entry in passive mode\n");
        return -1;
    }

    // Begins transmission of a file from the remote host
    if(ftpRetr(&ftp, url.filename)) {
        printf("ERROR: Cannot retrieve file %s\n", url.filename);
        return -1;
    }

    // Starting file transfer
    ftpDownload(&ftp, url.filename);

    // Disconnecting from server
    ftpDisconnect(&ftp);

    return 0;
}

void printUsage(char* argv0) {
    printf("\nUsage1 Normal: %s ftp://[<user>:<password>@]<host>/<url-path>\n",argv0);
    printf("Usage2 Anonymous: %s ftp://<host>/<url-path>\n\n", argv0);
}
```

```c
#include "ftp.h"

void initURL(url* url) {
    memset(url->user, 0, MAX_LENGTH);
    memset(url->password, 0, MAX_LENGTH);
    memset(url->host, 0, MAX_LENGTH);
    memset(url->path, 0, MAX_LENGTH);
    memset(url->filename, 0, MAX_LENGTH);
    url->port = 21;
}

const char* regExpression = "ftp://([[A-Za-z0-9]*:([A-Za-z0-9]*@])*([A-Za-z0-9.~-])+/([[A-Za-z0-9/~._-])+";

const char* regExprAnony = "ftp://([A-Za-z0-9.~-])+/([[A-Za-z0-9/~._-])+";

int parseURL(url* url, const char* urlStr) {
    printf("Parsing command line arguments...\n");

    char* tempURL, *element, *activeExpression;
    regex_t* regex;
    size_t nmatch = strlen(urlStr);
    regmatch_t pmatch[nmatch];
    int userPassMode;

    element = (char*) malloc(sizeof(char)*(strlen(urlStr) + 1));
    tempURL = (char*) malloc(sizeof(char)*(strlen(urlStr) + 1));

    memcpy(tempURL, urlStr, strlen(urlStr));

    if (tempURL[6] == '[') {
        userPassMode = 1;
        activeExpression = (char*) regExpression;
    } else {
        userPassMode = 0;
        activeExpression = (char*) regExprAnony;
    }

    regex = (regex_t*) malloc(sizeof(regex_t));

    int reti;
    if ((reti = regcomp(regex, activeExpression, REG_EXTENDED)) != 0) {
        printf("URL format is wrong.");
        return 1;
    }
```

```c
    if ((reti = regexec(regex, tempURL, nmatch, pmatch, REG_EXTENDED)) != 0) {
        printf("URL could not execute.");
        return 1;
    }

    free(regex);

    // removing ftp:// from string
    strcpy(tempURL, tempURL + 6);

    if (userPassMode) {
        //removing [ from string
        strcpy(tempURL, tempURL + 1);

        // saving username
        strcpy(element, processElementUntilChar(tempURL, ':'));
        memcpy(url->user, element, strlen(element));

        //saving password
        strcpy(element, processElementUntilChar(tempURL, '@'));
        memcpy(url->password, element, strlen(element));
        strcpy(tempURL, tempURL + 1);
    }

    //saving host
    strcpy(element, processElementUntilChar(tempURL, '/'));
    memcpy(url->host, element, strlen(element));

    //saving url path
    char* path = (char*) malloc(strlen(tempURL));
    int startPath = 1;
    while (strchr(tempURL, '/')) {
        element = processElementUntilChar(tempURL, '/');

        if (startPath) {
            startPath = 0;
            strcpy(path, element);
        } else {
            strcat(path, element);
        }

        strcat(path, "/");
    }
    strcpy(url->path, path);

    // saving filename
    strcpy(url->filename, tempURL);
```

```
    free(tempURL);
    free(element);

    /*printf("\n%s\n%s\n%s\n%s\n%s\n", url->user, url->password, url->host,
     url->path, url->filename);*/

    return 0;
}

int getIpByHost(url* url) {
    struct hostent* h;

    if ((h = gethostbyname(url->host)) == NULL) {
        herror("gethostbyname");
        return 1;
    }

//  printf("Host name  : %s\n", h->h_name);
//  printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *) h->h_addr)));

    char* ip = inet_ntoa(*((struct in_addr *) h->h_addr));
    strcpy(url->ip, ip);

    return 0;
}

char* processElementUntilChar(char* str, char chr) {
    // using temporary string to process substrings
    char* tempStr = (char*) malloc(strlen(str));

    // calculating length to copy element
    int index = strlen(str) - strlen(strcpy(tempStr, strchr(str, chr)));

    tempStr[index] = '\0'; // termination char in the end of string
    strncpy(tempStr, str, index);
    strcpy(str, str + strlen(tempStr) + 1);

    return tempStr;
}

static int connectSocket(const char* ip, int port) {
    int sockfd;
    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(ip); /*32 bit Internet address network byte ordered*/
```

```
    server_addr.sin_port = htons(port); /*server TCP port must be network byte ordered */

    // open an TCP socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        return -1;
    }

    // connect to the server
    if (connect(sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr))< 0) {
        perror("connect()");
        return -1;
    }

    return sockfd;
}

int ftpConnect(ftp* ftp, const char* ip, int port) {
    int socketfd;
    char rd[1024];

    if ((socketfd = connectSocket(ip, port)) < 0) {
        printf("ERROR: Cannot connect socket.\n");
        return 1;
    }

    ftp->control_socket_fd = socketfd;
    ftp->data_socket_fd = 0;

    if (ftpRead(ftp, rd, sizeof(rd))) {
        printf("ERROR: ftpRead failure.\n");
        return 1;
    }

    return 0;
}

int ftpLogin(ftp* ftp, const char* user, const char* password) {
    char sd[1024];

    // username
    sprintf(sd, "USER %s\r\n", user);
    if (ftpSend(ftp, sd, strlen(sd))) {
        printf("ERROR: ftpSend failure.\n");
        return 1;
    }
```

```
    if (ftpRead(ftp, sd, sizeof(sd))) {
        printf("ERROR: Access denied reading password response.\nftpRead failure.\n");
        return 1;
    }

    return 0;
}

int ftpCWD(ftp* ftp, const char* path) {
    char cwd[1024];

    sprintf(cwd, "CWD %s\r\n", path);
    if (ftpSend(ftp, cwd, strlen(cwd))) {
        printf("ERROR: Cannot send path to CWD.\n");
        return 1;
    }

    if (ftpRead(ftp, cwd, sizeof(cwd))) {
        printf("ERROR: Cannot send path to change directory.\n");
        return 1;
    }

    return 0;
}

int ftpPasv(ftp* ftp) {
    char pasv[1024] = "PASV\r\n";
    if (ftpSend(ftp, pasv, strlen(pasv))) {
        printf("ERROR: Cannot enter in passive mode.\n");
        return 1;
    }

    if (ftpRead(ftp, pasv, sizeof(pasv))) {
        printf("ERROR: None information received to enter in passive mode.\n");
        return 1;
    }

    // starting process information
    int ipPart1, ipPart2, ipPart3, ipPart4;
    int port1, port2;
    if ((sscanf(pasv, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)", &ipPart1,&ipPart2, &ipPart3, &ipPart4, &port1, &port2)) < 0) {
        printf("ERROR: Cannot process information to calculating port.\n");
        return 1;
    }

    // cleaning buffer
    memset(pasv, 0, sizeof(pasv));
```

```
    // forming ip
    if ((sprintf(pasv, "%d.%d.%d.%d", ipPart1, ipPart2, ipPart3, ipPart4))< 0) {
        perror("ERROR: Cannot form ip address.\n");
        return 1;
    }

    // calculating new port
    int portResult = port1 * 256 + port2;

    printf("IP: %s\n", pasv);
    printf("PORT: %d\n", portResult);

    if ((ftp->data_socket_fd = connectSocket(pasv, portResult)) < 0) {
        printf("ERROR: Incorrect file descriptor associated to ftp data socket fd.\n");
        return 1;
    }

    return 0;
}

int ftpRetr(ftp* ftp, const char* filename) {
    char retr[1024];

    sprintf(retr, "RETR %s\r\n", filename);
    if (ftpSend(ftp, retr, strlen(retr))) {
        printf("ERROR: Cannot send filename.\n");
        return 1;
    }

    if (ftpRead(ftp, retr, sizeof(retr))) {
        printf("ERROR: None information received.\n");
        return 1;
    }

    return 0;
}

int ftpDownload(ftp* ftp, const char* filename) {
    FILE* file;
    int bytes;

    if (!(file = fopen(filename, "w"))) {
        printf("ERROR: Cannot open file.\n");
        return 1;
    }
```

```
    char buf[1024];
    while ((bytes = read(ftp->data_socket_fd, buf, sizeof(buf)))) {
        if (bytes < 0) {
            printf("ERROR: Nothing was received from data socket fd.\n");
            return 1;
        }

        if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
            printf("ERROR: Cannot write data in file.\n");
            return 1;
        }
    }

    fclose(file);
    close(ftp->data_socket_fd);

    return 0;
}

int ftpDisconnect(ftp* ftp) {
    char disc[1024];

    if (ftpRead(ftp, disc, sizeof(disc))) {
        printf("ERROR: Cannot disconnect account.\n");
        return 1;
    }

    sprintf(disc, "QUIT\r\n");
    if (ftpSend(ftp, disc, strlen(disc))) {
        printf("ERROR: Cannot send QUIT command.\n");
        return 1;
    }

    if (ftp->control_socket_fd)
        close(ftp->control_socket_fd);

    return 0;
}

int ftpSend(ftp* ftp, const char* str, size_t size) {
    int bytes;

    if ((bytes = write(ftp->control_socket_fd, str, size)) <= 0) {
        printf("WARNING: Nothing was sent.\n");
        return 1;
    }
```

```
    printf("Bytes send: %d\nInfo: %s\n", bytes, str);

    return 0;
}

int ftpRead(ftp* ftp, char* str, size_t size) {
    FILE* fp = fdopen(ftp->control_socket_fd, "r");

    do {
        memset(str, 0, size);
        str = fgets(str, size, fp);
        int aux = atoi (str);
        if(aux==550){
            printf("ERROR: File not found.\n");
            return 1;
        }
        printf("%s", str);
    } while (!('1' <= str[0] && str[0] <= '5') || str[3] != ' ');

    return 0;
}
```