

2º Projeto de RCOM

**Rede de Computadores**

**Engenharia Informática e Computação**

**Regente: Manuel Alberto Pereira Ricardo**

**Turma 6**

* **Eduardo Luís Pinheiro da Silva – up201603135 - up201603135@fe.up.pt**
* **João Pedro Viveiros Franco – up201605775 - up201605775@fe.up.pt**
* **Tomás Nuno Fernandes Novo – up201604503 - up201604503@fe.up.pt**

**Sumário**

O projeto “**Rede de Computadores**” consiste não só configuração de uma rede de computadores, usufruindo do ***Cisco Router*** e ***Switch***, como também no estudo do seu funcionamento*.* Para além do manipulamento da rede, foi também desenvolvida uma aplicação de *download* com recurso ao ***File Transfer Protocol(FTP)***.Com o fim de prestar apoio ao projeto, foi elaborado este relatório.

O relatório apresenta conceitos fundamentais para a compreensão do projeto, complementando-o e deixando claro o término com sucesso do mesmo, visto que os objetivos ambicionados foram cumpridos.

**1 - Introdução**

No âmbito desta unidade curricular, foi-nos proposta como 2ª parte do trabalho prático a realização de um projeto cuja meta residia em configurar com sucesso uma rede de computadores, bem como a criação de uma aplicação na qual é realizada o *download* de um ficheiro.

Neste projeto objetivámos então a configuração com êxito de uma rede de computadores, a qual recorre a duas **VLAN’s** residentes num ***Switch,*** com o objetivo de conceder a execução de uma aplicação.

Relativamente à aplicação desenvolvida, esta permite o download de um ficheiro consoante o ***File Transfer Protocol(FTP).*** Para tal, recorremos a ligações de acordo com o ***Transmission Control Protocol(TCP),*** nomeadamente a ***sockets***.

Como referido no sumário, o projeto consiste na implementação destas duas componentes.

A estrutura do relatório foi elaborada com o objetivo de auxiliar na interpretação do projeto concebido, sendo que esta se encontra dividida em:

1. **Introdução –** onde são referidos os objetivos do trabalho e do relatório bem como os tipos de informação que poderão ser encontrados na restante estrutura.
2. **Aplicação –** explicitação da implementação da aplicação de download
3. **Rede de Computadores –** síntese das ilações retiradas da realização de cada experiência.
4. **Conclusões –** resumo de toda a informação e conclusões retiradas da realização do projeto.

**2 - Aplicação**

A primeira parte do trabalho consistiu no desenvolvimento de um cliente ***File Transfer Protocol***(FTP) usando a linguagem C e o RFC 959 como referencia.

O programa é corrido da seguinte maneira: **./download ftp://[<user>:<password>@]<host>/<url-path>** sendo "<user>:<password>@" um argumento opcional. O programa começa por separar o URL em **username, password, hostname** e **filename**. De seguida, aloca espaço para dois objetos da estrutura *connection,* que contém um endereço IP, um porto e um apontador para *FILE*. Na primeira conexão é aberta a ligação na porta 21 ao servidor passado nos argumentos. A segunda conexão é reservada para a ligação ao servidor em modo passivo. De seguida, é feito o *login* com *username* e *password* passado nos argumentos, ou, no caso de estes dois parâmetros não serem especificados, o *login* será efetuado com *username* a tomar o valor "*anonymous*" e a password o valor "none”. Para isto é utilizado o comando **USER** e o comando **PASS**. Depois de um login com sucesso, é chamada a função **receiveFile**, que começa por separar o caminho do ficheiro em caminho das pastas e nome do ficheiro. Se ao remover o nome do ficheiro do caminho este não ficar vazio, então é enviado o comando **CWD**, com o resto do caminho a mudar o "*working directory*" para essa pasta. Posteriormente, é enviado ao servidor o comando **PASV** com o fim de abrir a ligação em modo passivo, e, de seguida, a mensagem do servidor é lida e dela é extraído o endereço IP e o porto proveniente da mensagem, abrindo a segunda ligação no servidor. Após estes acontecimentos, o programa muda o tipo de transferência para binário, utilizando o comando "**TYPE I**", e recebe o tamanho do ficheiro a transferir pelo comando "**SIZE <filename>**". Finalmente, o programa envia o comando "**RETR <filename>**" e entra num *loop* no qual vai ler da segunda conexão um certo número de bytes por ciclo, escrevendo-os num ficheiro com o mesmo nome.

Para auxiliar o programa, foram desenvolvidas funções como **sendCommand,** que envia uma *string* para o servidor acabada em "\r\n", como especificado pelo RFC 959. Outra função deste género é a função **receiveMessage,** que lê linha-a-linha a mensagem do servidor, guardando o código da mensagem e um número de três dígitos, no qual o primeiro dígito índica um erro se for 4 ou 5. Esta função acaba quando receber uma linha que contenha o código com um espaço de seguida, como especificado pelo RFC 959, ou então quando ocorrer um *timeout*. O programa também permite o *upload* de ficheiros usando o argumento -u com o comando **STOR**.

**3 – Rede de Computadores**

**Experiência 1 – Configuração de um IP de rede**

Esta experiência consiste na conexão entre o **tux1** e o **tux4** através do ***switch***.

1. **Pacotes ARP e sua finalidade**

Os pacotes ARP baseiam-se no protocolo de comunicação *Adress Resolution Protocol*, cuja finalidade é encontrar o endereço da camada de ligação do endereço IPv4. Estes pacotes têm como fim o mapeamento do endereço de rede a um endereço físico(MAC).

1. **Endereços MAC e IP dos pacotes ARP**

Introduzindo o comando *ping* para o tux4 no tux1, o tux4 envia um pacote ARP com o seu endereço IP e endereço MAC. O endereço IP do tux4 é 172.16.60.254 e o seu MAC é 00:21:5a:c5:61:bb. O pacote enviado funciona como uma pergunta ao tux1, visto que o tux4 inquere sobre qual o endereço MAC do tux que realizou o comando *ping*. Como o tux4 não sabe o MAC do tux1, esse endereço é regitado como 00:00:00:00:00:00(logs REF). O tux1 responde confirmando que foi ele que fez ping e envia o seu endereço MAC. Sendo assim:

* Tux1 – IP:172. 16.60.1 MAC: 00:0f:fe:8c:af:71
* Tux4- IP: 172. 16.60.254 MAC: 00:21:5a:c5:61:bb

1. **Pacotes gerados pelo comando *ping***

O comando *ping,* quando introduzido na consola, gera não só pacotes ARP como também pacotes ICMP. Enquanto que os primeiros têm o objetivo de obter os endereços MAC, os segundos auxiliam na ocorrência de erros. (REF)

1. **Endereços MAC e IP dos pacotes *ping***

Fazendo *ping* ao tux4 através do tux1, os endereços IP e MAC de origem e destino dos pacotes passam a ser os destes tux’s.

Pacote de pedido (figura X);

* Endereço MAC origem: 00:0f:fe:8c:af:71 (tux1)
* Endereço IP origem: 172. 16.60.1 (tux1)
* Endereço MAC destino: 00:21:5a:c5:61:bb (tux4)
* Endereço IP destino: 172. 16.60.254 (tux4)

Pacote de resposta (figura X);

* Endereço MAC origem: 00:21:5a:c5:61:bb (tux4)
* Endereço IP origem: 172.16.60.254 (tux4)
* Endereço MAC destino: 00:0f:fe:8c:af:71 (tux1)
* Endereço IP destino: 172. 16.60.1 (tux1)

(Ref)

1. **Determinação do tipo (ARP, IP, ICMP) da trama recetora Ethernet**

O tipo da trama recetora Ethernet é descoberto através da inspeção do Ethernet Header de um pacote. A trama será do tipo IP se o Ethernet Header possuir o valor 0x0800. Analisando o IP header, caso assuma valor 1, o protocolo é ICMP, caso assuma o valor 0x0806, o protocolo é ARP. (Ref)

1. **Determinação do comprimento de uma trama recetora**

A determinação do comprimento da trama recetora é determinada recorrendo ao Wireshark, um programa analisador dos protocolos de internet.

1. **Interface *loopback* e sua importância**

A interface *loopback* trata-se de uma interfave de rede virtual que permite que os tux’s recebam respostas sobre eles próprios. A sua importância é fulcral no teste da configuração da carta de rede. (REF)

**Experiência 2 – Implementação de duas LANs virtuais num switch**

A segunda experiência efetuada traduz-se na criação de VLANY0 e VLANY1, duas LAN’s virtuais. Os tux’s 1 e 4 foram conectados a VLANY0. Por sua vez, o tux2 foi associado à outra LAN concebida.

1. **Configuração de VLANY0**

Com o fim de configurar VLANY0, é necessário estabelecer ligações de cabos entre os tux’s e o *switch,* sendo que a porta T4 da primeira régua tem que estar conectada à porta do *switch* da segunda régua. Por sua vez, a porta T3 da primeira régua precisa de estar ligada à porta S0 do tux1 e do tux4.

A criação da *vlan* resulta da invocação no **GTKTerm** dos seguintes comandos:

* + configure terminal
  + vlan y0
  + end

A adição das portas do tux1 e do tux4 é o resultado da introdução dos seguintes comandos:

* + configure terminal
  + interface fasethernet 0/nº da porta do tux(172.16.60..1 no tux1 e .254 no tux4)
  + switchport mode acess
  + switchport acess vlan y0
  + end

1. **Domínios de transmissão e conclusões retiradas dos registos**

Existem dois domínios de transmissão: um que contém tux1 e tux4 e outro que contém apenas tux2. Conclui-se a partir dos registos que tux1 consegue receber resposta do tux4 mas não do tux2 quando é feito *ping broadcast,* visto quetux2 não recebe respostas de nenhum dos outros tux na realização deste tipo de *ping*.

**Experiência 3 – Configuração de um router em LINUX**

Por sua vez, a execução desta experiência residiu na configuração de tux4 como um *router*, com a finalidade de estabelecer uma ligação entre VLANY0 e VLANY1

1. **Rotas dos tuxes e seu significado**

As rotas para as VLAN’s associadas a cada tux são as seguintes:

* Tux1 – rota VLAN0: 172.16.60.0 gateway: 172.16.60.1
* Tux2 – rota VLAN1: 172.16.61.0 gateway: 172.16.61.1
* Tux4 – rota VLAN0: 172.16.60.0 gateway: 172.16.60.254

rota VLAN1: 172.16.61.0 gateway: 172.16.61.253

No decorrer da experiência, foram criadas as seguintes rotas para cada tux:

* Tux1 – rota VLAN1: 172.16.61.0 gateway:172.16.60.254
* Tux2 - rota VLAN0: 172.16.60.0 gateway:172.16.61.254

1. **Informações fornecidas por uma entrada da tabela de *forwarding***

A tabela de *forwarding* contém a seguinte informação:

* Destination – destino da rota
* Gateway – ip do próximo ponto da rota
* Natemask – determina o ID da rede a partir do IP do destino
* Flags – informações sobre as idiossincrasias da rota
* Metric – custo das rotas
* Ref – total de referências para a rota
* Use – contador de pesquisas pela rota. Conta número de sucessos ou falhas da cache
* Interface – indicação da placa de rede gateway(eth0 ou eth1)

1. **Causa e observação de mensagens ARP e endereços MAC**

No caso de um tux dar *ping* a outro tux que tenho recebido o *ping* mas não conheça o endereço MAC do que realizou o comando, o tux que recebeu o *ping* pergunta qual o MAC do tux com aquele IP através do envio de uma mensagem ARP que contém o MAC do tux de origem associado e 00:00:00:00:00:00, visto que desconhece o tux de destino. Se isto acontecer, o tux de destino também envia uma mensagem ARP como respota, a qual contém o seu endereço MAC do tux de origem e do tux de destino. (REF)

1. **Causa e observação de pacotes ICMP**

Os pacotes ICMP observados são de *request* e *reply,* visto que depois da adição das rotas os tux’s têm a possibilidade de se visualizarem. No caso de esta visualização ser impossível, os pacotes ICMP enviados seriam de *Host Unreachable*.

1. **Endereços IP e MAC associados a um pacote ICMP**

Os endereços IP e MAC associados aos pacotes ICMP são os endereços IP e MAC dos tux’s de origem e de destino.

**Experiência 4 – Configuração de um router comercial e Implementação NAT**

Esta experiência consistiu na configuração de um router comercial. Primeiro foi implementado com uma conexão com a rede do laboratório. Posteriormente, o router foi configurado com NAT, estabelecendo uma ligação entre os computadores da rede e a internet.

1. **Configuração de um router estático num router comercial**

Com o fim de configurar um router comercial, conetou-se a porta T4 da primeira régua à porta do router da segunda régua e a porta T3 também da primeira régua à porta S0 do tux4. Relativamente à VLAN, a sua criação resulta da aplicação no **GTKTerm** dos seguintes comandos:

* configure terminal
* ip route (ip rota destino) 0.255.255.255 (ip gateway)
* exit

1. **Rotas seguidas pelos pacotes no decorrer da experiência**

Durante a experiência, os pacotes seguem a rota especificada. No caso de não haver especificação, os pacotes seguem a rota *default.*

1. **Configuração NAT num router comercial**

A configuração do NAT num router comercial implica a configuração da interface interna através do **GTKTerm** com a inserção dos comandos presente na figura 6.

1. **Funcionalidades NAT**

O **Network Adress Translation(**NAT) opera num router, no qual são conectadas duas redes e traduzidos endereços privados para endereços legais, objetivando a conservação de endereços IP, permitindo a conexão a uma rede pública ou à internet por parte de redes IP privadas que usem endereços IP não registados.

**Experiência 5 – DNS**

A configuração do *Domain Name System*(DNS) em todos os tux’s foi o pilar desta experiência. Um servidor de DNS possui uma base de dados que contém os endereços IP públicos, bem como os respetivos *hostnams.* O servidor DNS utilizado nesta experiência foi **services.netlab.fe.up.pt**(172.16.2.1).

1. **Configuração do serviço DNS num *host***

Para a configuração do serviço DNS, é necessária a alteração em cada um dos tux’s do ficheiro **resolv.conf**, utilizando para tal o comando *vi.* O ficheiro alterado deve conter:

1. search netlab.fe.up.pt
2. nameserver 172.16.2.1 (o 2 advém da sala I320)
3. **Pacotes trocados pelo DNS e informação transportada**

Através do DNS, é enviado um pacote que contém o *hostname* desejado do *host* para o *server,* pedindo o seu endereço IP. A resposta do servidor consiste no envio de um pacote que contém o IP do *hostname* (REF)

**Experiência 6 – Conexões TCP**

1. **Conexões TCP abertas pela aplicação FTP**

Foram abertas duas conexões TCP pela aplicação TCP. A primeira envia os comandos FTP ao *server* e recebe respostas. A segunda recebe dados enviados pelo servidor e recebe respostas do *client.*

1. **Conexão de transporte do controlo de informação**

É na conexão TCP responsável pela troca de comandos que é transportado o controlo de informação.

1. **Fases da conexão TCP**

A conexão TCP possui três fases: estabelecimento da ligação, troca de dados e encerramento da ligação.

1. **Mecanismo AQR TCP, campos TCP relevantes, informação relevante visualizada nos logs**

O Transmission Control Protocol(TCP) usa o mecanismo Automatic Repeat Request(ARP) com o método da janela deslizante, que consiste na transmissão de dados e controlo de erros.

1. **Mecanismo de controlo de congestão TCP, evolução do fluxo de dados da conexão ao longo do tempo e comparação com mecanismo de controlo de congestão TCP**

O mecanismo de controlo de congestão é realizado sempre que o TCP mantém uma janela de congestão que se baseia numa estimativa da quantidade de octetos que a rede encaminha, não enviando mais octetos do que o mínimo definido pelo recetor e janela de congestão. O fluxo de dados de conexão está de acordo com o mecanismo de controlo de congestão.

1. **Alteração da conexão de dados TCP devido ao aparecimento de uma segunda conexão TCP**

O aparecimento de uma segunda conexão TCP no decorrer de uma transferência de dados em simultâneo pode levar a um decréscimo da taxa de transmissão, visto que a taxa de transferência é distribuída igualmente para ambas as ligações.

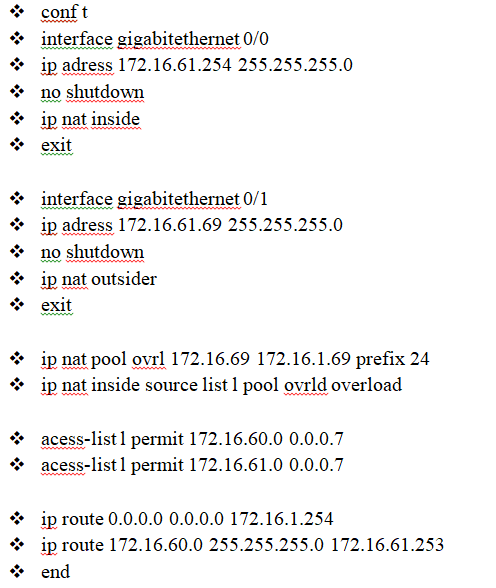
**4 – Conclusões**

Em suma, a implementação do cliente de *download* e a configuração de uma rede de computadores foram benéficas para os elementos constituintes do grupo devido à aquisição de conhecimentos.

Enquanto que as experiências realizadas contribuíram para uma aprendizagem valiosa sobre conceitos importantes no estabelecimento de uma rede de computadores, a implementação do cliente de *download* trouxe-nos novas ilações sobre o funcionamento do *File Transfer Protocol.*

Os objetivos a atingir foram alcançados, visto que implementámos com êxito tanto a aplicação de *download* como a rede em si, conforme nos foi proposto.

**Anexos**



**Figura 6 – Configuração NAT num router comercial**

**Código**

* **makefile**

CC = gcc

CFLAGS = -g

LDFLAGS =

DEPS = main.c connection.c connection.h utilities.c utilities.h makefile

OBJFILES = main.o connection.o utilities.o

TARGET = download

all: $(TARGET)

$(TARGET): $(OBJFILES) $(DEPS)

$(CC) $(CFLAGS) -o $(TARGET) $(OBJFILES) $(LDFLAGS)

clean:

rm -f $(OBJFILES) $(TARGET) \*~

* **README.txt**

Usage:

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

Examples:

./download “ftp://ftp.fe.up.pt/welcome.msg”

./download “ftp://speedtest.tele2.net/50MB.zip”

./download “ftp://dlpuser@dlptest.com:e73jzTRTNqCN9PYAAjjn@ftp.dlptest.com/curl.txt”

* **constants.h**

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

#define ANONYMOUS "anonymous"

#define SERVER\_PORT 21

#define SERVER\_HOSTNAME "ftp.fe.up.pt"

#define MAXCONNECTIONS 10

#define MAXATTEMPTS 3

#define TIMEOUT 3

#endif

* **connection.h**

#ifndef CONNECTION\_H

#define CONNECTION\_H

#include <stdio.h>

typedef struct

{

char\* IPAddress;

int port;

FILE\* fp;

}

**connection;**

**int initializeConnection(connection\*\* conn);**

**int freeConnection(connection\*\* conn);**

#endif

* **connection.c**

#include "connection.h"

#include <stdlib.h>

**int initializeConnection(connection\*\* conn)**

{

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

(\*conn)->IPAddress = malloc(16);

return 0;

}

**int freeConnection(connection\*\* conn)**

{

free((\*conn)->IPAddress);

free(\*conn);

return 0;

}

* **utilities.h**

#ifndef UTILITIES\_H

#define UTILITIES\_H

#include <stdio.h>

#include <fcntl.h>

**int getFreeFilePointer(FILE\*\* fpArray);**

**int closeSockets(FILE\*\* fileArray);**

**void sigalrm\_handler(int signal);**

**int setHandler();**

**int isNumber(char input);**

**void printUsage();**

**int findFirst(char\* str, char target);**

**int findLast(char\* str, char target);**

**int extractFromArgument(char\* input, char\* username, char\* password, char\* hostname, char\* filename);**

**void printPercentage(double percentage);**

**void printTransferRate(double rate);**

**void clearScreen();**

**int splitFilename(char\* fullPath, char\* path, char\* filename);**

#endif

* **utilities.c**

#include "utilities.h"

#include <string.h>

#include <stdlib.h>

#include <signal.h>

#include "constants.h"

**int getFreeFilePointer(FILE\*\* fpArray)**

{

int i;

for (i = 0; i < MAXCONNECTIONS; i++)

{

if (fpArray[i] == NULL)

return i;

}

return -1;

}

**int closeSockets(FILE\*\* fileArray)**

{

int i;

for (i = 0; fileArray[i] != NULL; ++i)

{

fclose(fileArray[i]);

}

free(fileArray);

return 0;

}

**void sigalrm\_handler(int signal)**

{

printf("Server response timed out!\n");

}

**int setHandler()**

{

struct sigaction sigalrm\_action;

sigalrm\_action.sa\_handler = sigalrm\_handler;

sigemptyset(&sigalrm\_action.sa\_mask);

sigalrm\_action.sa\_flags = 0;

if (sigaction(SIGALRM, &sigalrm\_action, NULL) < 0)

{

fprintf(stderr,"Unable to install SIGINT handler\n");

return 1;

}

return 0;

}

**int isNumber(char input)**

{

return (input >= '0' && input <= '9');

}

**void printUsage()**

{

printf("Usage: download [OPTIONS] [HOSTNAME] [FILE]\n");

}

**int findFirst(char\* str, char target)**

{

int i;

for (i = 0; str[i] != 0; i++)

{

if (str[i] == target)

return i;

}

return -1;

}

**int findLast(char\* str, char target)**

{

int i, ret = -1;

for (i = 0; str[i] != 0; i++)

{

if (str[i] == target)

ret = i;

}

return ret;

}

**int extractFromArgument(char\* input, char\* username, char\* password, char\* hostname, char\* filename)**

{

char ftp[7];

memcpy(ftp, input, 6);

ftp[6] = 0;

username[0] = 0;

password[0] = 0;

hostname[0] = 0;

filename[0] = 0;

if (strcmp(ftp, "ftp://") == 0)

{

input += 6;

int index = findLast(input, '@');

if (index != -1) // Login present

{

int index2 = findFirst(input, ':');

if (index2 != -1)

{

memcpy(username, input, index2+1);

username[index2] = 0;

memcpy(password, input+index2+1, index-index2-1);

password[index-index2+1] = 0;

input += index+1;

}

else

return 1;

}

else // No login present

{

strcpy(username, ANONYMOUS);

strcpy(password, "none");

}

int index2 = findFirst(input, '/');

if (index2 != -1)

{

memcpy(hostname, input, index2);

hostname[index2+1] = 0;

strcpy(filename, input+index2+1);

}

else

return 1;

return 0;

}

return 1;

}

**void printPercentage(double percentage)**

{

if (percentage >= 0 && percentage <= 1)

{

printf("<");

int i, length = 15 /\* length of the percentage bar \*/;

for (i = 0; i < length; i++)

{

if ((double)i/length < percentage)

printf("|");

else

printf(" ");

}

printf(">%.1f%%\n", percentage\*100);

}

}

**void printTransferRate(double rate)**

{

if (rate > 0)

{ printf("Transfer rate : %.1f KB/s\n", rate/1024); }

}

**void clearScreen()**

{

printf("\033[2J\033[1;1H");

printf("\033[2J\033[1;1H");

}

**int splitFilename(char\* fullPath, char\* path, char\* filename)**

{

int i, index = -1;

for (i = 0; fullPath[i] != 0; i++)

{

if (fullPath[i] == '/')

index = i;

}

if (index != -1)

{

memcpy(path, fullPath, index+1);

path[index+1] = 0;

strcpy(filename, fullPath+index+1);

return 0;

}

else

{

path[0] = 0;

strcpy(filename, fullPath);

return -1;

}

}

* **main.c**

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <unistd.h>

#include <netdb.h>

#include <string.h>

#include <fcntl.h>

#include <ctype.h>

#include <sys/time.h>

#include <sys/stat.h>

#include "constants.h"

#include "connection.h"

#include "utilities.h"

connection\* currentFp = NULL;

**int attemptConnect(connection\* conn, char\* message)**

{

int sockfd, i;

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(conn->IPAddress); /\*32 bit Internet address network byte ordered\*/

server\_addr.sin\_port = htons(conn->port); /\*server TCP port must be network byte ordered \*/

for (i = 0; i < MAXATTEMPTS; i++)

{

sleep(1);

/\*open an TCP socket\*/

if ((sockfd = socket(AF\_INET,SOCK\_STREAM,0)) < 0)

{

// fprintf(stderr, "socket() failed!\n");

continue;

}

/\*connect to the server\*/

if (connect(sockfd,

(struct sockaddr \*)&server\_addr,

sizeof(server\_addr)) < 0)

{

// fprintf(stderr, "connect() failed!\n");

continue;

}

break;

}

if (i == MAXATTEMPTS)

{

fprintf(stderr, "Connection timed out!\n");

return 1;

}

conn->fp = fdopen(sockfd, "r+");

return 0;

}

**int receiveMessage(connection\* conn, char\* message)**

{

size\_t\* bufferSize = malloc(sizeof(size\_t));

\*bufferSize = 1024;

int bytes, i, flag = 1;

char\* buffer = malloc(\*bufferSize);

char code[4];

message[0] = 0;

buffer[0] = 0;

for (i = 0; flag; i++)

{

alarm(TIMEOUT);

bytes = getline(&buffer, bufferSize, conn->fp);

alarm(0); // Cancel alarm

if (bytes < 0)

{

fprintf(stderr, "Error on reading from server!\n");

free(bufferSize);

free(buffer);

return -1;

}

buffer[bytes] = 0;

if (i == 0)

{

memcpy(code, buffer, 3);

code[3] = 0;

}

if (isNumber(buffer[0]) && isNumber(buffer[1]) && isNumber(buffer[2]))

{

if (memcmp(code, buffer, 3) == 0)

{

if (buffer[3] == ' ')

flag = 0;

}

}

strcat(message, buffer);

}

free(bufferSize);

free(buffer);

return (code[0] == '4' || code[0] == '5');

}

**int sendCommand(connection\* conn, char\* command)**

{

int length = strlen(command);

char\* buffer = malloc(length+2+1);

buffer[0] = 0;

strcpy(buffer, command);

strcat(buffer, "\r\n");

int bytes = fwrite(buffer, 1, length+2, conn->fp);

if (bytes != length+2)

{

fprintf(stderr, "Error seding command!\n");

return 1;

}

return 0;

}

**int closeConnection(connection\* conn, char\* message)**

{

message[0] = 0;

strcpy(message, "QUIT");

if (sendCommand(conn, message) != 0)

return 1;

if (receiveMessage(conn, message) != 0)

return 1;

printf("%s\n", message);

return 0;

}

**int login(connection\* conn, char\* message, char\* username, char\* password)**

{

char buffer[100+6+1];

buffer[0] = 0;

strcat(buffer, "user ");

strcat(buffer, username);

sendCommand(conn, buffer);

if (receiveMessage(conn, message) != 0)

return 1;

// printf("%s\n", message);

buffer[0] = 0;

strcat(buffer, "pass ");

strcat(buffer, password);

sendCommand(conn, buffer);

if (receiveMessage(conn, message) != 0)

return 1;

// printf("%s\n", message);

return 0;

}

**int enterPassiveMode(connection\*\* connections, char\* message)**

{

int sockfd, i, j, k, start, IPAddressCounter, messageLength;

for (k = 0; k < MAXCONNECTIONS; k++)

{

connections[1]->IPAddress[0] = 0;

start = -1;

IPAddressCounter = 0;

connections[1]->port = 0;

sendCommand(connections[0], "PASV");

receiveMessage(connections[0], message);

messageLength = strlen(message);

printf("Changing server for passive mode\n");

for (i = 0; i < messageLength; ++i)

{

if (message[i] == '(')

start = i+1;

if (start != -1 && (message[i] == ',' || message[i] == ')'))

{

if (IPAddressCounter < 4)

{

int offset = strlen(connections[1]->IPAddress);

memcpy(&connections[1]->IPAddress[offset], &message[start], i-start);

if (IPAddressCounter < 3)

{

connections[1]->IPAddress[offset + i-start] = '.';

connections[1]->IPAddress[offset + i-start + 1] = 0;

}

else

connections[1]->IPAddress[offset + i-start] = 0;

}

else

{

char portString[4];

memcpy(portString, &message[start], i-start);

portString[i-start] = 0;

long int part = strtol(portString, NULL, 10);

if (IPAddressCounter == 4)

{

connections[1]->port = 256\*part;

}

else if (IPAddressCounter == 5)

{

connections[1]->port += part;

break;

}

}

start = i+1;

IPAddressCounter++;

}

}

// printf("IPAddress = %s\n", connections[1]->IPAddress);

// printf("port = %i\n", connections[1]->port);

if (attemptConnect(connections[1], message) == 0)

break;

}

if (i == MAXCONNECTIONS)

return 1;

return 0;

}

**int receiveFile(connection\*\* connections, char\* message, char\* serverFilename)**

{

char path[200];

char filename[100];

splitFilename(serverFilename, path, filename);

if (path[0] != 0) // Not empty string

{

message[0] = 0;

strcat(message, "CWD ");

strcat(message, path);

sendCommand(connections[0], message);

receiveMessage(connections[0], message);

printf("%s\n", message);

}

if (enterPassiveMode(connections, message) != 0)

return 1;

// Sets transfer type to binary

message[0] = 0;

strcat(message, "TYPE I");

sendCommand(connections[0], message);

if (receiveMessage(connections[0], message) != 0)

{

printf("%s\n", message);

return 1;

}

// Gets size of file

message[0] = 0;

strcat(message, "SIZE ");

strcat(message, filename);

sendCommand(connections[0], message);

if (receiveMessage(connections[0], message) != 0)

{

printf("%s\n", message);

return 1;

}

long long size = strtoll(&message[4], NULL, 10);

// printf("size = %lli\n", size);

message[0] = 0;

strcat(message, "RETR ");

strcat(message, filename);

sendCommand(connections[0], message);

receiveMessage(connections[0], message);

// printf("%s\n", message);

size\_t bufferSize = 512;

int i, sumBytes = 0, bytes = bufferSize, fd = open(filename, O\_WRONLY | O\_TRUNC | O\_CREAT, 0777);

char\* buffer = malloc(bufferSize);

struct timeval startTime, finishTime;

double sumTime = 0;

if (gettimeofday(&startTime, NULL) != 0)

printf("Error getting time!\n");

if (gettimeofday(&finishTime, NULL) != 0)

printf("Error getting time!\n");

int sumBytesAverage = 0;

double sumTimeAverage = 0, repeatTime = 0.5, rate = 0;

for (i = 0; sumBytes < size; i++)

{

// usleep(100 \* 65); // Sleeps for 10 millisecond

bytes = fread(buffer, 1, bufferSize, connections[1]->fp);

if (bytes < 0)

{

printf("Error getting file from!\n");

return 1;

}

bytes = write(fd, buffer, bytes);

if (bytes < 0)

{

printf("Error writing to file!\n");

return 1;

}

if (gettimeofday(&finishTime, NULL) != 0)

printf("Error getting time!\n");

double deltaTime = (double)(finishTime.tv\_sec - startTime.tv\_sec) + (double)(finishTime.tv\_usec - startTime.tv\_usec)/1000/1000; // In seconds

if (gettimeofday(&startTime, NULL) != 0)

printf("Error getting time!\n");

if (sumTimeAverage > repeatTime)

{

clearScreen();

rate = (double)sumBytesAverage / sumTimeAverage;

sumBytesAverage = 0;

sumTimeAverage = 0;

printPercentage((double)sumBytes / size);

printTransferRate(rate);

}

sumBytes += bytes;

sumTime += deltaTime;

sumBytesAverage += bytes;

sumTimeAverage += deltaTime;

}

clearScreen();

printPercentage((double)sumBytes / size);

printTransferRate(rate);

free(buffer);

close(fd);

fclose(connections[1]->fp);

receiveMessage(connections[0], message);

printf("%s\n", message);

printf("File downloaded with average %.1f KB/s\n", (double)sumBytes/sumTime/1024);

return 0;

}

**int sendFile(connection\*\* connections, char\* message, char\* filepath)**

{

char path[200];

char filename[100];

splitFilename(filepath, path, filename);

if (enterPassiveMode(connections, message) != 0)

return 1;

// Sets transfer type to binary

message[0] = 0;

strcat(message, "TYPE I");

sendCommand(connections[0], message);

receiveMessage(connections[0], message);

printf("%s\n", message);

message[0] = 0;

strcat(message, "STOR ");

strcat(message, filename);

sendCommand(connections[0], message);

receiveMessage(connections[0], message);

printf("%s\n", message);

size\_t bufferSize = 512;

int i, sumBytes = 0, bytes = bufferSize, fd = open(filepath, O\_RDONLY);

char\* buffer = malloc(bufferSize);

struct stat st;

if (stat(filename, &st) != 0)

{

printf("Failed to get file size!\n");

return 1;

}

int size = st.st\_size;

struct timeval startTime, finishTime;

double sumTime = 0;

if (gettimeofday(&startTime, NULL) != 0)

printf("Error getting time!\n");

if (gettimeofday(&finishTime, NULL) != 0)

printf("Error getting time!\n");

int sumBytesAverage = 0;

double sumTimeAverage = 0, repeatTime = 0.5, rate = 0;

for (i = 0; sumBytes < size; i++)

{

bytes = read(fd, buffer, bytes);

if (bytes < 0)

{

printf("Error reading from file!\n");

return 1;

}

bytes = fwrite(buffer, 1, bufferSize, connections[1]->fp);

if (bytes < 0)

{

printf("Error sending file to server!\n");

return 1;

}

if (gettimeofday(&finishTime, NULL) != 0)

printf("Error getting time!\n");

double deltaTime = (double)(finishTime.tv\_sec - startTime.tv\_sec) + (double)(finishTime.tv\_usec - startTime.tv\_usec)/1000/1000; // In seconds

if (gettimeofday(&startTime, NULL) != 0)

printf("Error getting time!\n");

if (sumTimeAverage > repeatTime)

{

clearScreen();

rate = (double)sumBytesAverage / sumTimeAverage;

sumBytesAverage = 0;

sumTimeAverage = 0;

printPercentage((double)sumBytes / size);

printTransferRate(rate);

}

sumBytes += bytes;

sumTime += deltaTime;

sumBytesAverage += bytes;

sumTimeAverage += deltaTime;

}

clearScreen();

printPercentage((double)sumBytes / size);

printTransferRate(rate);

free(buffer);

close(fd);

fclose(connections[1]->fp);

receiveMessage(connections[0], message);

printf("%s\n", message);

printf("File uploaded with average %.1f KB/s\n", (double)sumBytes/sumTime/1024);

return 0;

}

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

{

if (argc < 2 || argc > 3)

{

printUsage();

return 1;

}

char \*username = malloc(100), \*password = malloc(100), \*hostname = malloc(100), \*filename = malloc(100);

if (extractFromArgument(argv[argc-1], username, password, hostname, filename) != 0)

{

printUsage();

return 1;

}

char\* options = "hu";

int i, opterr = 0, uflag = 0;

char c;

while ((c = getopt (argc-1, argv, options)) != -1)

{

if (c == 'h')

{

printUsage();

return 0;

}

if (c == 'u')

{

uflag = 1;

}

else if (c == '?')

{

int flag = 1;

for (i = 0; options[i] != 0; i++)

{

if (options[i] == ':' && optopt == options[i-1])

{

fprintf (stderr, "Option -%c requires an argument.\n", optopt);

flag = 0;

}

}

if (flag)

{

if (isprint (optopt))

fprintf (stderr, "Unknown option `-%c'.\n", optopt);

else

fprintf (stderr, "Unknown option character `\\x%x'.\n", optopt);

}

}

else

return 1;

}

for (i = optind; i < argc-1; i++)

printf ("Non-option argument %s\n", argv[i]);

char message[4096];

setHandler();

struct hostent \* ent = gethostbyname(hostname);

char IPAddress[16];

inet\_ntop(AF\_INET, ent->h\_addr\_list[0], IPAddress, INET\_ADDRSTRLEN);

connection\* connections[2];

initializeConnection(&connections[0]);

initializeConnection(&connections[1]);

strcpy(connections[0]->IPAddress, IPAddress);

connections[0]->port = SERVER\_PORT;

if (attemptConnect(connections[0], message) != 0)

return 1;

receiveMessage(connections[0], message);

printf("%s\n", message);

if (login(connections[0], message, username, password) != 0)

{

fprintf(stderr, "Failed to login!\n");

return 1;

}

printf("Login sucessful!\n");

if (uflag)

{

if (sendFile(connections, message, filename))

return 1;

}

else

{

if (receiveFile(connections, message, filename))

return 1;

}

if (closeConnection(connections[0], message) != 0)

return 1;

freeConnection(&connections[0]);

freeConnection(&connections[1]);

return 0;

}