

2º Trabalho Laboratorial



Mestrado Integrado em Engenharia Informática e
Computação

Redes de Computadores

Nuno Ramos - up201405498
Maria João Mira Paulo - up201403820
Pedro Duarte Costa - up201403291

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

21 de Dezembro de 2016

Conteúdo

1	Sumário	3
2	Introdução	3
3	Aplicação Download	3
3.1	Arquitetura	3
3.2	Resultados	6
4	Configuração da rede	7
4.1	Configurar um Ip de Rede	7
4.2	Implementar duas LAN's virtuais no switch	8
4.3	Configurar um router em Linux	9
4.4	Configurar um Router comercial e implementar NAT	9
4.5	DNS	11
4.6	Ligações TCP	12
5	Conclusões	12
6	Anexos	13

1 Sumário

2 Introdução

Este relatório está dividido em duas grandes partes. A primeira é implementar um cliente FTP (*File Transfer Protocol*) para fazer transferência de ficheiros, a segunda é configurar e analisar uma rede.

Este relatório, inicialmente irá explicar a arquitetura seguida na implementação do cliente e a sua estruturação, e em seguida, serão abordadas as experiências feitas nas aulas práticas e explicado o que é suposto alcançar em cada experiência.

3 Aplicação Download

3.1 Arquitetura

A aplicação está dividida em duas partes, numa primeira parte iremos processar a string de input do utilizador e em seguida, iremos focar-nos na ligação com o servidor, login e download do ficheiro indicado pelo utilizador.

Utilizamos uma estrutura chamada *url* onde guardamos toda a informação disponibilizada pelo o utilizador na string de input, ou seja, o username, a palavra-passe, o host e o host-path do ficheiro que o utilizador quer fazer download.

```
1 typedef struct url {
2     char * type;
3     char * user;
4     char * password;
5     char * host;
6     char * url_path;
7 } url;
```

Listing 1: Estrutura *url*

Ao correr o programa, primeiro é validado o input do utilizador. Após, a validação da string de input, é chamada a função *getInfo* que retira da string de input os vários dados necessário para inicializar a estrutura *url*.

```
1 int getInfo(char * str,url * url_info);
```

Listing 2: Argumentos da função *getInfo*

Após todas as variáveis estarem instanciadas, é chamada a função *connect_to_server* responsável por ligar o cliente FTP ao servidor através de um socket.

```
1 int connect_to_server(int socket, struct addrinfo *res)
```

Listing 3: Argumentos da função *connect_to_server*

Depois de conectados, o cliente irá fazer login no servidor com o username e a palavra passe disponibilizados pelo o utilizador. Para isso, utiliza a função *login_to_server*. Função que envia dois comandos ao servidor, o comando "USER 'username'\r\n" e "PASS 'password'\r\n" aguardando pela respetiva resposta do servidor.

```
1 int login_to_server(int sockfd, url *url_info)
```

Listing 4: Argumentos da função *login_to_server*

Após efetuado o login, iremos colocar o servidor em modo passivo, para o fazer chamamos a função *set_PASV_mode* que envia o comando "PASV\r\n". Desta forma o cliente inicia ambas as conexões ao servidor, resolvendo o problema de as firewalls filtrarem os dados recebidos pelo cliente do servidor. O servidor responde, enviando uma string, com o endereço ip e a porta onde iremos, mais tarde fazer o download do ficheiro. Após, a receção da mensagem chamamos a função *get_ip_adress* que desmonta a string e retorna o ip e a porta. Com este endereço e porta chamamos a função *initTCP* (função fornecida pelo professor), que retorna o socket que mais tarde iremos utilizar para fazer download do ficheiro.

```

1  int get_ip_address(char *answer3, char ip[100], char port1[100],
2  char port2[100]);
3  int initTCP(char *address, int port);

```

Listing 5: Argumentos das funções *get_ip_address* e *initTCP*

Após, estabelecida a nova conexão, iremos agora pedir ao servidor o ficheiro desejado, para isso utilizamos a função *asking_file_to_server*, que envia o comando "RETR 'path_to_file'\r\n". Após enviar o comando, lemos a resposta do servidor e verificamos se o ficheiro existe, se não existir acabamos a execução do programa. Após o pedido do ficheiro, chamamos a função *read_file_from_server* que lê o ficheiro pedido pelo o utilizador e o guarda em disco.

```

1  int asking_file_to_server(int sockfd, url *url_info);
2  int read_file_from_server(int datafd, char filename[MAXDATASIZE]);

```

Listing 6: Argumentos das funções *asking_file_to_server* e *read_file_from_server*

Terminada a receção do ficheiro resta apenas fechar os sockets abertos e libertar a memória alocada para terminar o programa.

```

1  #include "utilities.h"
2
3  int connect_to_server(int socket, struct addrinfo *res);
4  int login_to_server(int sockfd, url *url_info);
5  int set_PASV_mode(int sockfd, char *answer);
6  int get_ip_address(char *answer3, char ip[100], char port1[100], char port2[100]);
7  int get_string(char *info, int numberOfCommas, char ret[100]);
8  int write_to_server(int sockfd, const char *message);
9  int read_from_server(int sockfd, char * answer);
10 int asking_file_to_server(int sockfd, url *url_info);
11 int read_file_from_server(int datafd, char *filename);

```

Listing 7: connection.h

3.2 Resultados

Esta aplicação foi testada com diversos ficheiros, tanto em modo anónimo como em modo não anónimo. A transferência dos vários ficheiros foi verificada tendo sido o máximo ficheiro testado um ficheiro iso com cerca de 400MB.

Em caso de erro, para além da aplicação terminar é impresso na consola o erro em causa, de modo a que o utilizador tenha o máximo controlo possível sobre o sucedido.

4 Configuração da rede

4.1 Configurar um Ip de Rede

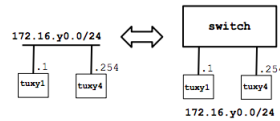


Figura 1: Experiência 1

A primeira experiência tem como objetivo configurar dois computadores numa só rede e estabelecer assim uma comunicação entre estes. Para isso foram configurados, com o comando `ifconfig [ip]i`, os dois computadores **tux51** e **tux54** para que estes assumissem os endereços de **IP** de **172.16.50.1** e **172.16.50.254**, respectivamente. Para isso, foram atribuídos estes mesmo valores e ativadas as portas `eth0` através do comando **`ifconfig eth0 up`**.

De seguida, depois de configuradas todas as rotas necessárias, foi chamado o comando `ping` para verificar a conectividade entre os dois computadores.

A chamada do comando Ping gera pacote ICMP. O comando Ping usa os pacotes ICMP para transferir mensagens de controlo entre endereços IP.

Analisando o log do wireshark, podemos verificar que, o pacote ARP envia uma mensagem broadcast a todos os computadores com o objetivo de saber qual o MAC address correspondente a um certo IP. A partir daí, todos os pacotes ICMP enviados obtêm uma resposta vinda deste último.

1	Destination	Protocol	Length	Info
2				
3	Broadcast	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
4	G-ProcCom_8b:e4:a7	ARP	60	172.16.50.254 is at 00:21:5a:c3:78:70

MAC é o *hardware address*, único, da placa de rede. ARP é o protocolo que atribui a um IP um MAC. Os ARP Packets possuem informação de qual é o MAC address para um determinado IP.

A interface *loopback* é responsável por enviar de 10 em 10 segundos um pacote LOOP que verifica se a ligação e o sistema se encontram ativos.

4.2 Implementar duas LAN's virtuais no switch

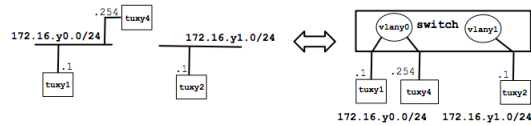


Figura 2: Experiência 2

Nesta experiência foram criadas duas LANs virtuais no switch: a primeira constituída pelos computadores **tux51** e **tux54**, e a segunda pelo computador **tux52**. Com esta configuração, o computador 2 deixaria de ter acesso aos computadores **tux51** e **tux54**, uma vez que se encontrariam em sub-redes diferentes.

```
1 enable
2 configure terminal
3 vlan 50
4 exit
5 vlan 51
6 exit
7 interface fastethernet 0/1
8 switchport mode access
9 switchport access vlan 50
10 interface fastethernet 0/4
11 switchport mode access
12 switchport access vlan 50
13 interface fastethernet 0/2
14 switchport mode access
15 switchport access vlan 51
16 end
17 show vlan brief
18 show running-config interface fastethernet 0/1
19 show interfaces fastethernet 0/1 switchport
```

De seguida, foi enviado o comando Ping do **tux51** para o **tux54** e depois para o **tux52**. De seguida foi também chamado o comando Ping Broadcast a partir do **tux52**.

Através do log do wireshark podemos concluir que existem duas sub-

redes diferentes, logo, dois broadcasts diferentes. O comando Ping broadcast envia um ping para todos os computadores estao ligados a essa rede. Ou seja se o **tux51** mandar um Ping broadcast, apenas o **tux54** o vai receber pois estão ambos ligados a **vlan50**. Contudo o **tux52** não o recebe pois esta ligado a uma rede diferente, neste caso a **vlan51**.

Desta forma, podemos concluir que existem dois domínios de Broadcast, **vlan50** e **vlan51**.

4.3 Configurar um router em Linux

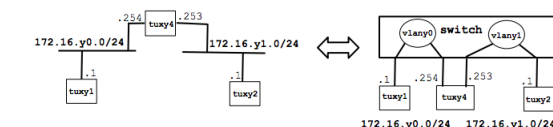


Figura 3: Experiência 3

Esta experiência consiste na configuração do computador **tux54** como router por forma a ligar as duas vlan's existentes:

- **Vlan 50:** 172.16.50.0/24
- **Vlan 51:** 172.16.51.0/24

De seguida, foi ativada a porta eth1 do tux54. Além disso, foi ligada ao switch e, de seguida, à vlan51. Configura-se esta mesma porta com o endereço IP 172.16.51.253/24.

Por fim, foram adicionadas as rotas necessárias no tux51 e no tux52 por forma a que estes possam enviar informação entre eles através do tux54. O primeiro endereço da rota identifica a gama de endereços para a qual se quer adicionar a rota, ou seja os possíveis endereços de destino, e o segundo endereço identifica o IP para o qual se deve encaminhar o pacote. Estas rotas foram adicionadas através do comando:

Finalmente, foi possível a comunicação entre qualquer um dos três computadores.

4.4 Configurar um Router comercial e implementar NAT

Nesta experiência pretendia-se a configuração de um router comercial com **NAT** devidamente implementado.

```

1 //Rotas adicionadas ao tux51
2 //Neste caso o pacote deve ser reencaminhado através do IP do tux54.
3 route add - net 172.16.51.0/24 gw 172.16.50.254
4 //Rotas adicionadas ao tux52
5 //Desta vez o IP 172.16.y1.253 é o IP do tux54 nesta sub-rede.
6 route add - net 172.16.50.0/24 gw 172.16.51.253

```

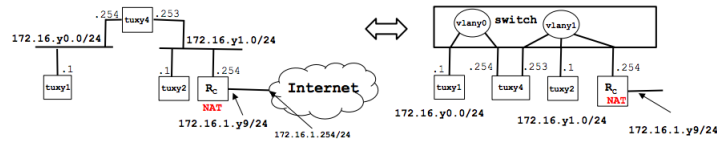


Figura 4: Experiência 4

Na medida em que os IP's públicos são um recurso limitado e atualmente escasso, o NAT tem como objetivo poupar o espaço de endereçamento público, recorrendo a IP's privados. Os endereços públicos são pagos e permitem identificar univocamente uma máquina (PC, routers, etc) na Internet. Por outro lado os endereços privados apenas fazem sentido num domínio local e não são conhecidos (encaminháveis) na Internet, sendo que uma máquina configurada com um IP privado terá de sair para a Internet através de um IP público.

Essa tradução de um endereço privado num endereço público é conseguida através do **NAT**.

Para configurar o router foi necessário fazer login na linha de comandos e correr o script do Anexo. Configurou-se o router definindo as rotas internas e externas com o comando `ip route` na consola de configuração do router.

```

1 ip route 0.0.0.0 0.0.0.0 172.16.1.254
2 //este comando cria uma rota, quando o IP de destino for
3 //172.16.50.0/24 deve enviar os pacotes pela gateway 172.16.51.253.
4 ip route 172.16.50.0 255.255.255.0 172.16.51.253

```

Desta forma, definiu-se o computador tux54 como default gateway do computador tux51 e o router como default gateway dos computadores tux52 e tux54. Assim, os pacotes enviados pelo computador tux51 seguem para o computador tux54 e depois para o router ou para o computador tux52.

Para testar, foi executado no tux51 um ping ao router da sala e verificou-se que os pacotes enviados pelo tux51, passavam pelo tux54, onde eram reencaminhados para o router no IP 172.16.51.254.

4.5 DNS

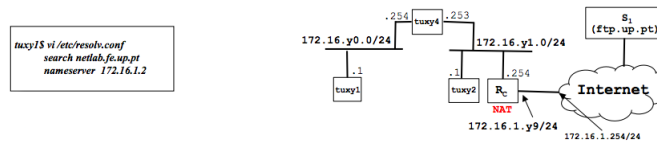


Figura 5: Experiência 5

Esta experiência consiste em adicionar um DNS.

O **DNS** é um sistema de gerenciamento de nomes hierárquico e distribuído para computadores, serviços ou qualquer recurso conectado à Internet ou numa rede privada. Funciona como um sistema de tradução de endereços IP em nomes de domínios.

Para isso, editou-se o ficheiro resolv.conf situado no diretório /etc nos computadores com o sistema operativo em Linux, de modo a que fosse semelhante ao seguinte:

```
1 search netlab.fe.up.pt
2 nameserver 172.16.1.1
```

Por fim, testou-se esta funcionalidade, pingando www.google.pt, obtendo se o seguinte log no wireshark:

4	4.498050	172.16.41.1	172.16.1.1	DNS	73	Standard query 0x9cc1 A www.google.pt
5	5.889228	CiscoInc_d4:1c:04	CiscoInc_d4:1c:04	LOOP	60	Reply
6	6.814760	CiscoInc_d4:1c:04	Spanning-tree-for-	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
7	7.814577	CiscoInc_d4:1c:04	Spanning-tree-for-	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
8	9.502371	HewlettP_d7:45:c4	CiscoInc_e3:df:10	ARP	42	Who has 172.16.41.254? Tell 172.16.41.1
9	9.502432	172.16.41.1	193.136.28.10	DNS	73	Standard query 0x9cc1 A www.google.pt
10	9.502900	CiscoInc_e3:df:10	HewlettP_d7:45:c4	ARP	60	172.16.41.254 is at 68:ef:b6:e3:df:10
11	9.503032	193.136.28.10	172.16.41.1	DNS	235	Standard query response 0x9cc1 A www.google.pt A 216.58.214.163 NS ns2.google.com NS ns1.g...
12	9.504034	172.16.41.1	216.58.214.163	ICMP	98	Echo (ping) request id=0x47f0, seq=1/256, ttl=64 (reply in 15)
13	9.504167	172.16.41.253	172.16.41.1	ICMP	126	Redirect (Redirect for host)
14	9.504205	172.16.41.1	172.16.1.1	DNS	60	Standard query 0x3464 PTR 253.41.16.172.in-addr.arpa
15	9.521261	216.58.214.163	172.16.41.1	ICMP	98	Echo (ping) reply id=0x47f0, seq=1/256, ttl=53 (request in 12)
16	10.019452	CiscoInc_d4:1c:04	Spanning-tree-for-	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
17	12.029351	CiscoInc_d4:1c:04	Spanning-tree-for-	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
18	14.028047	CiscoInc_d4:1c:04	Spanning-tree-for-	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
19	14.509349	172.16.41.1	193.136.28.10	DNS	86	Standard query 0x3464 PTR 253.41.16.172.in-addr.arpa
20	14.510368	HewlettP_d7:45:c4	Kye_25:1a:f4	ARP	42	Who has 172.16.41.253? Tell 172.16.41.1
21	14.510462	Kye_25:1a:f4	HewlettP_d7:45:c4	ARP	60	172.16.41.253 is at 00:c8:df:25:1a:f4
22	14.541002	193.136.28.10	172.16.1.1	DNS	163	Standard query response 0x3464 No such name PTR 253.41.16.172.in-addr.arpa SOA prisoner.ia...
23	14.541134	172.16.41.1	172.16.1.1	DNS	86	Standard query 0x3582 PTR 254.41.16.172.in-addr.arpa
24	14.573343	172.16.1.1	172.16.41.1	DNS	163	Standard query response 0x3582 No such name PTR 254.41.16.172.in-addr.arpa SOA prisoner.ia...
25	14.573585	172.16.41.1	172.16.1.1	DNS	87	Standard query 0xabe7 PTR 163.214.58.216.in-addr.arpa
26	14.574308	172.16.1.1	172.16.41.1	DNS	382	Standard query response 0xabe7 PTR 163.214.58.216.in-addr.arpa PTR mad@1s26-in-f3.1e100.ne...
27	14.574461	172.16.41.1	216.58.214.163	ICMP	98	Echo (ping) request id=0x47f0, seq=2/252, ttl=64 (reply in 38)
28	14.574592	172.16.41.253	172.16.41.1	ICMP	126	Redirect (Redirect for host)

Figura 6: DNS

4.6 Ligações TCP

Por fim, na experiência 6, compilou-se e executou-se a aplicação desenvolvida e descrita na primeira parte do relatório. O teste foi feito com recurso à transferência de um ficheiro através de um servidor FTP. A transferência foi bem-sucedida, mostrando que a configuração da rede foi feita sem erros.

O **Transmission Control Protocol (TCP)** utiliza o mecanismo Automatic Repeat Request (ARQ). Este método consiste no controlo de erros na transmissão de dados. Para isso utiliza acknowledgments (mensagens enviadas pelo recetor indicando que a trama de dados foi recebida corretamente) e timeouts (tempo permitido para esperar por um acknowledgment), de forma a garantir uma transmissão confiável através do serviço não confiável. Se não for recebido um acknowledgment antes do timeout, a trama é retransmitida até ser recebido um acknowledgment.

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

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

5 Conclusões

Após a conclusão do segundo projeto de Redes e Computadores, o grupo interiorizou grande parte dos conceitos necessários para uma implementação estável e coerente do que era pedido no guião.

A implementação do cliente FTP (*File Transfer Protocol*) foi concluída com sucesso, o cliente era capaz de fazer download de diferentes tipos de ficheiros e tamanhos diferentes. Com esta implementação, o grupo entendeu melhor este protocolo e percebeu, também, como funciona um protocolo oficial comparando ao protocolo desenvolvido no primeiro projeto.

Relativamente à configuração de rede, também foi concluída com sucesso. Permitindo, aos elementos do grupo perceber, a um nível aprofundado, como funciona a configuração de uma rede, algo usado no dia-a-dia.

Portanto, podemos concluir que os objetivos delineados pelo guião do segundo projeto, foram atingidos com sucesso. Deste projeto, o grupo fica com uma perspetiva diferente de como funciona uma rede de computadores.

6 Anexos

```
1  Configuração Computadores
2
3  #!/bin/bash
4  hostname=$(hostname | tr -d 'tux')
5  stand=$(echo $hostname | head -c 1)
6  if [ "`echo $HOSTNAME|grep tux${stand}1 -c`" = "1" ]; then
7      /etc/init.d/networking restart
8      arp -d ipaddress
9      ifconfig eth0 down
10     ifconfig eth0 up
11     ifconfig eth0 172.16.${stand}0.1/24
12     route add -net 172.16.${stand}1.0/24 gw 172.16.${stand}0.254
13     route add default gw 172.16.${stand}0.254
14     echo -e "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
15     ifconfig
16 fi
17 if [ "`echo $HOSTNAME|grep tux${stand}4 -c`" = "1" ]; then
18     arp -d ipaddress
19     ifconfig eth0 down
20     ifconfig eth1 down
21     ifconfig eth1 up
22     ifconfig eth0 up
23     ifconfig eth0 172.16.${stand}0.254/24
24     ifconfig eth1 172.16.${stand}1.253/24
25     route add default gw 172.16.${stand}1.254
26     echo 1 > /proc/sys/net/ipv4/ip_forward
27     echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
28     echo -e "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
29     ifconfig
30 fi
31 if [ "`echo $HOSTNAME|grep tux${stand}2 -c`" = "1" ]; then
32     arp -d ipaddress
33     ifconfig eth0 down
34     ifconfig eth0 up
35     ifconfig eth0 172.16.${stand}1.1/24
36     route add -net 172.16.${stand}0.0/24 gw 172.16.${stand}1.253
37     route add default gw 172.16.${stand}1.254
38     echo -e "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
39     ifconfig
40 fi
41 route -n
```

```
1  Configuração Switch
2
3  #!/bin/bash
4
5  command_switch () {
6      echo $1 > /dev/ttyS0
7  }
8
9  config_port () {
10     command_switch "interface fastethernet 0/$1"
11     command_switch "switchport mode access"
12     command_switch "switchport access vlan $2"
13 }
14
15 hostname=$(hostname | tr -d 'tux')
16 stand=$(echo $hostname | head -c 1)
17
18 #command_switch "enable"
19 #sleep 1
20 #command_switch "8nortel"
21
22 command_switch "configure terminal"
23
24 command_switch "vlan ${stand}0"
25 command_switch "exit"
26
27 command_switch "vlan ${stand}1"
28 command_switch "exit"
29
30 config_port "1" "${stand}0"
31 config_port "4" "${stand}0"
32 config_port "3" "${stand}1"
33 config_port "2" "${stand}1"
34 config_port "5" "${stand}1"
35
36 command_switch "end"
37 command_switch "show vlan brief"
```

```

1  Configuração Router
2
3  #!/bin/bash
4
5  command_router () {
6      echo $1 > /dev/ttyS0
7      #cat < /dev/ttyS0
8  }
9
10 getopts n nat
11 hostname=$(hostname | tr -d 'tux')
12 stand=$(echo $hostname | head -c 1)
13 command_router "configure terminal"
14 command_router "interface gigabitethernet 0/0"
15 command_router "ip address 172.16.${stand}1.254 255.255.255.0"
16 command_router "no shutdown"
17 if [ "$nat" != "?" ]
18 then
19     command_router "ip nat inside"
20 fi
21 command_router "exit"
22 command_router "interface gigabitethernet 0/1"
23 command_router "ip address 172.16.1.${stand}9 255.255.255.0"
24 command_router "no shutdown"
25 if [ "$nat" != "?" ]
26 then
27     command_router "ip nat outside"
28 fi
29 command_router "exit"
30 if [ "$nat" != "?" ]
31 then
32     command_router "ip nat pool ovrlld 172.16.1.${stand}9 172.16.1.${stand}9 prefix 24"
33     command_router "ip nat inside source list 1 pool ovrlld overload"
34     #access-list - ultimo numero (exemplo: 7) indicar quantas portas podem aceder ao router
35     #tux4 nao pode porque é a port 253, que é maior que 7
36     command_router "access-list 1 permit 172.16.${stand}0.0 0.0.0.7"
37     command_router "access-list 1 permit 172.16.${stand}1.0 0.0.0.7"
38 fi
39 command_router "ip route 0.0.0.0 0.0.0.0 172.16.1.254"
40 command_router "ip route 172.16.${stand}0.0 255.255.255.0 172.16.${stand}1.253"
41 command_router "end"
42 command_router "show running-config"

```

```

1  clientTPC.c
2
3  #include "utilities.h"
4  #include "connection.h"
5
6  // login anonymous
7  // password anything
8
9  int initTCP(char *address, int port) {
10     int      sockfd;
11     struct sockaddr_in server_addr;
12
13     /*server address handling*/
14     bzero((char*)&server_addr,sizeof(server_addr));
15     server_addr.sin_family = AF_INET;
16     /*32 bit Internet address network byte ordered*/
17     server_addr.sin_addr.s_addr = inet_addr(address);
18     /*server TCP port must be network byte ordered */
19     server_addr.sin_port = htons(port);
20
21     /*open an TCP socket*/
22     if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
23         perror("socket()");
24         exit(-1);
25     }
26
27     /*connect to the server*/
28     if(connect(sockfd,(struct sockaddr *)&server_addr,sizeof(server_addr)) < 0){
29         perror("connect()");
30         exit(-1);
31     }
32
33     return sockfd;
34 }

```

```

1  int main(int argc, char **argv) {
2
3      char answer[MAXDATASIZE];
4      char answer3[MAXDATASIZE];
5
6      if (argc != 2) {
7          printf("\n\nInvalid arguments, expected:\n");
8          printf(" ./app ftp://[<user>:<password>@]<host>/<url-path>\n\n");
9          return -1;
10     }
11
12     url *url_info = malloc(sizeof(url));
13     getInfo(argv[1], url_info);
14
15     int ret = strcmp(url_info->user, "anonymous");
16
17     if (ret != 0) {
18         printf("Error! User must be anonymous\n");
19         exit(-1);
20     }
21
22     int sockfd;
23     struct addrinfo hints, *res;
24
25     memset(&hints, 0, sizeof(hints));
26     hints.ai_family = AF_INET;
27     hints.ai_socktype = SOCK_STREAM;
28
29     if (getaddrinfo(url_info->host, "21", &hints, &res) != 0) {
30         fprintf(stderr, "getaddrinfo: %s\n", "ERROR");
31         exit(0);
32     }
33     if ((sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) <
34         0) {
35         perror("socket()");
36         exit(0);
37     }
38     connect_to_server(sockfd, res);
39     read_from_server(sockfd, answer);
40 }

```

```
1 login_to_server(sockfd, url_info);
2
3     printf("%s\n\n", "Logged in");
4
5     set_PASV_mode(sockfd, answer3);
6     printf("%s\n", "Passive mode set !");
7
8     char ip[100];
9     char port1[100];
10    char port2[100];
11
12    get_ip_adress(answer3, ip, port1, port2);
13
14    printf("Ip : %s\n", ip);
15
16    int port = atoi(port1) * 256 + atoi(port2);
17
18    printf("Port : %d\n\n", port);
19
20    int datafd = initTCP(ip, port);
21
22    asking_file_to_server(sockfd,url_info);
23
24    char *filename = calloc(MAXDATASIZE, sizeof(char));
25
26    get_filename(url_info->url_path, filename);
27
28    printf("filename: %s\n", filename);
29
30    read_file_from_server(datafd,filename);
31
32    close(datafd);
33    close(sockfd);
34    exit(0);
```

```

1 connect.c
2
3 #include "connection.h"
4 #define BUFFER_SIZE 512
5
6 int connect_to_server(int socket, struct addrinfo *res) {
7     if (connect(socket, res->ai_addr, res->ai_addrlen) < 0) {
8         perror("connect()");
9         exit(0);
10    }
11    return 1;
12 }
13
14 int login_to_server(int sockfd, url *url_info) {
15
16     char answer[MAXDATASIZE];
17     char answer2[MAXDATASIZE];
18     char user_password_command[MAXDATASIZE];
19     char user_login_command[MAXDATASIZE];
20     memset(user_login_command,0,sizeof(user_login_command));
21     memset(user_password_command,0,sizeof(user_password_command));
22     strcat(user_login_command, "USER ");
23     strcat(user_login_command, url_info->user);
24     strcat(user_login_command, "\r\n");
25     printf("%s\n", user_login_command);
26     write_to_server(sockfd, user_login_command);
27     read_from_server(sockfd, answer);
28     printf("%s\n",answer );
29     strcat(user_password_command, "PASS ");
30     strcat(user_password_command, url_info->password);
31     strcat(user_password_command, "\r\n");
32     printf("PASS: %s\n",user_password_command );
33
34     write_to_server(sockfd, user_password_command);
35     read_from_server(sockfd, answer2);
36     printf("%s\n",answer2);
37     return 1;
38 }

```

```

1  int set_PASV_mode(int sockfd, char answer[MAXDATASIZE]){
2      char pasv_command[9]="PASV \r\n";
3
4      write_to_server(sockfd, pasv_command);
5
6      read_from_server(sockfd, answer);
7
8      return 1;
9  }
10
11 int get_ip_adress(char answer3[MAXDATASIZE], char ip[100], char port1[100], char port2[100]){
12
13     char *ret = calloc(MAXDATASIZE, sizeof(char));
14     ret=strchr(answer3, '(');
15
16     ret++;
17
18     get_string(ret, 4, ip);
19     printf("11111111\n" );
20     ret += strlen(ip);
21
22     get_string(ret, 2, port1);
23
24     ret += strlen(port1);
25
26     get_string(ret, 2, port2);
27
28     memmove(port1, port1+1, strlen(port1));
29     memmove(port2, port2+1, strlen(port2));
30
31     return 1;
32
33 }

```

```
1  int get_string(char *info, int numberOfCommas, char ret[100]){
2
3      int i=0;
4      int atualNumberOfCommas=0;
5      while(1){
6          if(*info == ',' || *info == ' '){
7              atualNumberOfCommas++;
8              if (atualNumberOfCommas == numberOfCommas){
9                  break;
10             }
11             else ret[i] = '.';
12         }
13         else {
14             ret[i] = *info;
15         }
16
17         i++;
18         info++;
19     }
20
21     ret[i] = '\0';
22
23
24     return 1;
25 }
26
27 int write_to_server(int sockfd, const char *message) {
28     int bytes = 0;
29
30     bytes = write(sockfd, message, strlen(message));
31
32     if (bytes == strlen(message))
33         return 1;
34     perror("Writting to server.");
35
36     return 1;
37 }
```

```

1  int read_from_server(int sockfd, char answer[MAXDATASIZE]) {
2
3      int numbytes = 0;
4      if ((numbytes = recv(sockfd, answer, MAXDATASIZE - 1, 0)) == -1) {
5          perror("Receiving from server");
6          exit(1);
7      }
8      answer[numbytes] = '\0';
9
10     return 1;
11 }
12
13 int asking_file_to_server(int sockfd, url *url_info){
14     char file_path_to_download_command[MAXDATASIZE]; memset(file_path_to_download_command,0,sizeof(file_p
15     char answer[MAXDATASIZE];
16     strcat(file_path_to_download_command, "RETR ");
17     strcat(file_path_to_download_command, url_info->url_path);
18     strcat(file_path_to_download_command, "\r\n");
19     if(write(sockfd, file_path_to_download_command, strlen(file_path_to_download_command)) < 0){
20         perror("error on write retr command to the server");
21         exit(-1);
22     }
23     read_from_server(sockfd, answer);
24     printf("%s\n",answer);
25     char ret[5];
26
27     ret[0] = answer[0];
28     ret[1] = answer[1];
29     ret[2] = answer[2];
30     ret[3] = '\0';
31
32     int ret_int = atoi(ret);
33     if(ret_int == 550){
34         printf("File does not exist!Ending program!\n");
35         exit(-1);
36     }
37     return 1;
38 }

```

```
1 int read_file_from_server(int datafd, char filename[MAXDATASIZE]){
2     int new_file_fd = open(filename, O_CREAT|O_WRONLY, 0777);
3     int ret;
4     char buffer[MAXDATASIZE];
5
6     if(new_file_fd < 0){
7         perror("Error open file");
8         exit(-1);
9     }
10    printf("Start reading file\n\n");
11
12    while((ret=read(datafd,buffer, BUFFER_SIZE)) > 0){
13        if(ret == -1){
14            perror("Error reading file from server");
15            exit(-1);
16        }
17
18        if(write(new_file_fd,buffer,ret) < 0){
19            perror("Error writing to the destination file");
20            exit(-1);
21        }
22    }
23    printf("File read, ending program\n");
24    return 1;
25 }
```

```

1  utilities.c
2
3  #include "utilities.h"
4
5  int getInfo(char * str,url * url_info){
6
7      char * token = malloc(strlen(str));
8      strcpy(token, str);
9
10     url_info->type = calloc(strlen(token), sizeof(char));
11     url_info->user = calloc(strlen(token), sizeof(char));
12     url_info->password = calloc(strlen(token), sizeof(char));
13     url_info->host = calloc(strlen(token), sizeof(char));
14     url_info->url_path = calloc(strlen(token), sizeof(char));
15
16     strcpy(url_info->type, token);
17     token = strtok(token+6, ":");
18     strcpy(url_info->user, token);
19     token = strtok(NULL, "@");
20     strcpy(url_info->password, token);
21     token = strtok(NULL, "/");
22     strcpy(url_info->host, token);
23     token = strtok(NULL, "");
24     strcpy(url_info->url_path, token);
25
26     return 1;
27 }
28
29 int get_filename(char * path, char * filename){
30
31     char * token = malloc(strlen(path));
32     memcpy(token, path, strlen(path));
33     token = strtok(path, "/");
34
35     while(token != NULL){
36         memset(filename,0,strlen(filename));
37         memcpy(filename, token, strlen(token));
38         token = strtok(NULL, "/");
39     }
40     return 1;
41 }

```

```
1  utilities.h
2
3  #pragma once
4
5  #include <stdio.h>
6  #include <sys/types.h>
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <arpa/inet.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <signal.h>
13 #include <netdb.h>
14 #include <string.h>
15 #include <strings.h>
16 #include <termios.h>
17 #include <fcntl.h>
18
19 #define MAXDATASIZE 512
20
21 typedef struct Url {
22     char * type;
23     char * user;
24     char * password;
25     char * host;
26     char * url_path;
27 } url;
28
29 int getInfo(char * str, url * url_info);
30 int get_filename(char path[100], char * filename);
```
