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

2 Introdução

Este relatório esta dividido em duas grandes parte. 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 *conect_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.

```c
int get_ip_adress(char *answer3, char ip[100], char port1[100],
char port2[100]);
int initTCP(char *address, int port);
```

Listing 5: Argumentos das funções *get_ip_adress* 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.

```c
int asking_file_to_server(int sockfd,url *url_info);
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 rececão do ficheiro resta apenas fechar os sockets abertos e libertar a memória alocada para terminar o programa.

```c
#include "utilities.h"

int connect_to_server(int socket, struct  addrinfo *res);
int login_to_server(int sockfd, url *url_info);
int set_PASV_mode(int sockfd, char *answer);
int get_ip_adress(char *answer3, char ip[100], char port1[100], char port2[100]);
int get_string(char *info, int numberOfCommas, char ret[100]);
int write_to_server(int sockfd, const char *message);
int read_from_server(int sockfd, char * answer);
int asking_file_to_server(int socketfd, url *url_info);
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

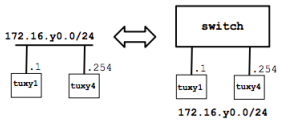

Figura 1: Experiência 1

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]¿, 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 connectividade 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-ProCom_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

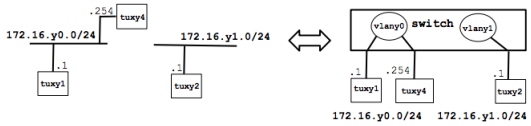

Figura 2: Experiência 2

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.

```
enable
configure terminal
vlan 50
exit
vlan 51
exit
interface fastethernet 0/1
switchport mode access
switchport access vlan 50
interface fastethernet 0/4
switchport mode access
switchport access vlan 50
interface fastethernet 0/2
switchport mode access
switchport access vlan 51
end
show vlan brief
show running-config interface fastethernet 0/1
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 brodcast envia um ping para todos os computadores estao ligados a essa rede. Ou seja se o **tux51** mandar um Ping brodcast, 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

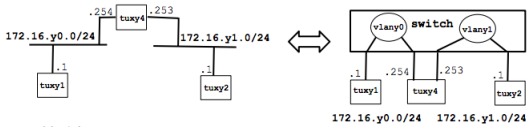

Figura 3: Experiência 3

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 endereçco 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.

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

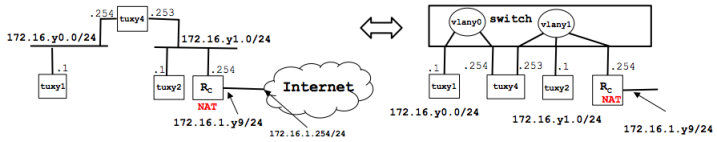

Figura 4: Experiência 4

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 atrvés 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.

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

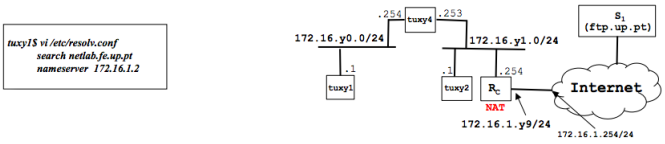

Figura 5: Experiência 5

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	6.014780	CiscoInc_d4:1c:04	Spanning-tree-(for…	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8004
7	8.014577	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	9.502908	CiscoInc_e3:df:10	HewlettP_d7:45:c4	ARP	60	172.16.41.254 is at 68:ef:bd:e3:df:10
11	9.503832	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	9.504285	172.16.41.1	172.16.1.1	DNS	86	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	14.029247	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:c0:df:25:1a:f4
22	14.541002	193.136.28.10	172.16.41.1	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	14.574388	172.16.1.1	172.16.41.1	DNS	302	Standard query response 0xabe7 PTR 163.214.58.216.in-addr.arpa PTR mad01s26-in-f3.1e100.ne…
27	14.574461	172.16.41.1	216.58.214.163	ICMP	98	Echo (ping) request id=0x47f0, seq=2/512, ttl=64 (reply in 30)
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 se 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

Configuração Computadores

```
#!/bin/bash
hostname=$(hostname | tr -d 'tux')
stand=$(echo $hostname | head -c 1)
if [ "`echo $HOSTNAME|grep tux${stand}1 -c`" = "1" ]; then
        /etc/init.d/networking restart
        arp -d ipaddress
        ifconfig eth0 down
        ifconfig eth0 up
        ifconfig eth0 172.16.${stand}0.1/24
        route add -net 172.16.${stand}1.0/24 gw 172.16.${stand}0.254
        route add default gw 172.16.${stand}0.254
        echo -e "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
        ifconfig
fi
if [ "`echo $HOSTNAME|grep tux${stand}4 -c`" = "1" ]; then
        arp -d ipaddress
        ifconfig eth0 down
        ifconfig eth1 down
        ifconfig eth1 up
        ifconfig eth0 up
        ifconfig eth0 172.16.${stand}0.254/24
        ifconfig eth1 172.16.${stand}1.253/24
        route add default gw 172.16.${stand}1.254
        echo 1 > /proc/sys/net/ipv4/ip_forward
        echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
        echo -e "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
        ifconfig
fi
if [ "`echo $HOSTNAME|grep tux${stand}2 -c`" = "1" ]; then
        arp -d ipaddress
        ifconfig eth0 down
        ifconfig eth0 up
        ifconfig eth0 172.16.${stand}1.1/24
        route add -net 172.16.${stand}0.0/24 gw 172.16.${stand}1.253
        route add default gw 172.16.${stand}1.254
        echo -e "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" > /etc/resolv.conf
        ifconfig
fi
route -n
```

```
Configuração Switch

#!/bin/bash

command_switch () {
        echo $1 > /dev/ttyS0
}

config_port () {
        command_switch "interface fastethernet 0/$1"
        command_switch "switchport mode access"
        command_switch "switchport access vlan $2"
}

hostname=$(hostname | tr -d 'tux')
stand=$(echo $hostname | head -c 1)

#command_switch "enable"
#sleep 1
#command_switch "8nortel"

command_switch "configure terminal"

command_switch "vlan ${stand}0"
command_switch "exit"

command_switch "vlan ${stand}1"
command_switch "exit"

config_port "1" "${stand}0"
config_port "4" "${stand}0"
config_port "3" "${stand}1"
config_port "2" "${stand}1"
config_port "5" "${stand}1"

command_switch "end"
command_switch "show vlan brief"
```

Configuração Router

```
#!/bin/bash

command_router () {
        echo $1 > /dev/ttyS0
        #cat < /dev/ttyS0
}
getopts n nat
hostname=$(hostname | tr -d 'tux')
stand=$(echo $hostname | head -c 1)
command_router "configure terminal"
command_router "interface gigabitethernet 0/0"
command_router "ip address 172.16.${stand}1.254 255.255.255.0"
command_router "no shutdown"
if [ "$nat" != "?" ]
then
        command_router "ip nat inside"
fi
command_router "exit"
command_router "interface gigabitethernet 0/1"
command_router "ip address 172.16.1.${stand}9 255.255.255.0"
command_router "no shutdown"
if [ "$nat" != "?" ]
then
        command_router "ip nat outside"
fi
command_router "exit"
if [ "$nat" != "?" ]
then
command_router "ip nat pool ovrld 172.16.1.${stand}9 172.16.1.${stand}9 prefix 24"
command_router "ip nat inside source list 1 pool ovrld overload"
#access-list - ultimo numero (exemplo: 7) indicar quantas portas podem aceder ao rooter
#tux4 nao pode porque é a port 253, que é maior que 7
command_router "access-list 1 permit 172.16.${stand}0.0 0.0.0.7"
command_router "access-list 1 permit 172.16.${stand}1.0 0.0.0.7"
fi
command_router "ip route 0.0.0.0 0.0.0.0 172.16.1.254"
command_router "ip route 172.16.${stand}0.0 255.255.255.0 172.16.${stand}1.253"
command_router "end"
command_router "show running-config"
```

```
clientTPC.c

#include "utilities.h"
#include "connection.h"

// login anonymous
// password anything

int initTCP(char *address, 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;
    /*32 bit Internet address network byte ordered*/
        server_addr.sin_addr.s_addr = inet_addr(address);
    /*server TCP port must be network byte ordered */
        server_addr.sin_port = htons(port);

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

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

    return sockfd;
}
```

```c
int main(int argc, char **argv) {

        char answer[MAXDATASIZE];
        char answer3[MAXDATASIZE];

  if (argc != 2) {
    printf("\n\nInvalid arguments, expected:\n");
    printf(" ./app ftp://[<user>:<password>@]<host>/<url-path>\n\n");
                return -1;
  }

        url *url_info = malloc(sizeof(url));
        getInfo(argv[1], url_info);

        int ret = strcmp(url_info->user,"anonymous");

        if(ret != 0){
                printf("Error! User must be anonymous\n");
                exit(-1);
        }

  int sockfd;
  struct addrinfo hints, *res;

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_STREAM;

  if (getaddrinfo(url_info->host, "21", &hints, &res) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", "ERROR");
    exit(0);
  }
  if ((sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) <
      0) {
    perror("socket()");
    exit(0);
  }
  connect_to_server(sockfd, res);
  read_from_server(sockfd, answer);
}
```

```
login_to_server(sockfd, url_info);

      printf("%s\n\n", "Logged in");

      set_PASV_mode(sockfd, answer3);
      printf("%s\n", "Passive mode set !");

      char ip[100];
      char port1[100];
      char port2[100];

      get_ip_adress(answer3, ip, port1, port2);

      printf("Ip : %s\n", ip);

      int port = atoi(port1) * 256 + atoi(port2);

      printf("Port : %d\n\n", port);

      int datafd = initTCP(ip, port);

      asking_file_to_server(sockfd,url_info);

      char *filename = calloc(MAXDATASIZE, sizeof(char));

      get_filename(url_info->url_path, filename);

      printf("filename: %s\n", filename);

      read_file_from_server(datafd,filename);

      close(datafd);
close(sockfd);
exit(0);
```

connect.c

```c
#include "connection.h"
#define BUFFER_SIZE 512

int connect_to_server(int socket, struct addrinfo *res) {
  if (connect(socket, res->ai_addr, res->ai_addrlen) < 0) {
    perror("connect()");
    exit(0);
  }
  return 1;
}

int login_to_server(int sockfd, url *url_info) {

  char answer[MAXDATASIZE];
  char answer2[MAXDATASIZE];
  char user_password_command[MAXDATASIZE];
  char user_login_command[MAXDATASIZE];
  memset(user_login_command,0,sizeof(user_login_command));
  memset(user_password_command,0,sizeof(user_password_command));
  strcat(user_login_command, "USER ");
  strcat(user_login_command, url_info->user);
  strcat(user_login_command, "\r\n");
  printf("%s\n", user_login_command);
  write_to_server(sockfd, user_login_command);
  read_from_server(sockfd, answer);
  printf("%s\n",answer );
  strcat(user_password_command, "PASS ");
  strcat(user_password_command, url_info->password);
  strcat(user_password_command, "\r\n");
  printf("PASS: %s\n",user_password_command );

    write_to_server(sockfd, user_password_command);
    read_from_server(sockfd, answer2);
    printf("%s\n",answer2);
  return 1;
}
```

```c
int set_PASV_mode(int sockfd, char answer[MAXDATASIZE]){
  char pasv_command[9]="PASV \r\n";

  write_to_server(sockfd, pasv_command);

  read_from_server(sockfd, answer);

  return 1;
}

int get_ip_adress(char answer3[MAXDATASIZE], char ip[100], char port1[100], char port2[100]){

  char *ret  = calloc(MAXDATASIZE, sizeof(char));;
  ret=strchr(answer3, '(');

  ret++;

  get_string(ret, 4, ip);
printf("11111111\n" );
  ret += strlen(ip);

  get_string(ret, 2, port1);

  ret += strlen(port1);

  get_string(ret, 2, port2);

  memmove(port1, port1+1, strlen(port1));
  memmove(port2, port2+1, strlen(port2));

  return 1;

}
```

```c
int get_string(char *info, int numberOfCommas, char ret[100]){

  int i=0;
  int atualNumberOfCommas=0;
  while(1){
    if(*info == ',' || *info == ')'){
      atualNumberOfCommas++;
      if (atualNumberOfCommas == numberOfCommas){
        break;
      }
      else ret[i] = '.';
    }
    else {
      ret[i] = *info;
    }

    i++;
    info++;
  }

  ret[i] = '\0';


  return 1;
}

int write_to_server(int sockfd, const char *message) {
  int bytes = 0;

  bytes = write(sockfd, message, strlen(message));

  if (bytes == strlen(message))
    return 1;
  perror("Writting to server.");

  return 1;
}
```

```c
int read_from_server(int sockfd, char answer[MAXDATASIZE]) {

  int numbytes = 0;
  if ((numbytes = recv(sockfd, answer, MAXDATASIZE - 1, 0)) == -1) {
    perror("Receiving from server");
    exit(1);
  }
  answer[numbytes] = '\0';

  return 1;
}

int asking_file_to_server(int socketfd, url *url_info){
  char file_path_to_download_command[MAXDATASIZE]; memset(file_path_to_download_command,0,sizeof(file_pa
  char answer[MAXDATASIZE];
  strcat(file_path_to_download_command, "RETR ");
  strcat(file_path_to_download_command, url_info->url_path);
  strcat(file_path_to_download_command, "\r\n");
  if(write(socketfd, file_path_to_download_command, strlen(file_path_to_download_command)) < 0){
    perror("error on write retr command to the server");
    exit(-1);
  }
  read_from_server(socketfd, answer);
  printf("%s\n",answer);
  char ret[5];

  ret[0] = answer[0];
  ret[1] = answer[1];
  ret[2] = answer[2];
  ret[3] = '\0';

  int ret_int = atoi(ret);
  if(ret_int == 550){
    printf("File does not exit!Ending program!\n");
    exit(-1);
  }
  return 1;
}
```

```c
int read_file_from_server(int datafd,char filename[MAXDATASIZE]){
  int new_file_fd = open(filename, O_CREAT|O_WRONLY, 0777);
  int ret;
  char buffer[MAXDATASIZE];

  if(new_file_fd < 0){
    perror("Error open file");
    exit(-1);
  }
  printf("Start reading file\n\n");

  while((ret=read(datafd,buffer, BUFFER_SIZE)) > 0){
    if(ret == -1){
      perror("Error reading file from server");
      exit(-1);
    }

    if(write(new_file_fd,buffer,ret) < 0){
      perror("Error writing to the destination file");
      exit(-1);
    }
  }
  printf("File read, ending program\n");
  return 1;
}
```

utilities.c

```c
#include "utilities.h"

int getInfo(char * str,url * url_info){

  char * token = malloc(strlen(str));
  strcpy(token, str);

  url_info->type = calloc(strlen(token), sizeof(char));
  url_info->user = calloc(strlen(token), sizeof(char));
  url_info->password = calloc(strlen(token), sizeof(char));
  url_info->host = calloc(strlen(token), sizeof(char));
  url_info->url_path = calloc(strlen(token), sizeof(char));

  strcpy(url_info->type, token);
  token = strtok(token+6, ":");
  strcpy(url_info->user, token);
  token = strtok(NULL, "@");
  strcpy(url_info->password, token);
  token = strtok(NULL, "/");
  strcpy(url_info->host, token);
  token = strtok(NULL, "");
  strcpy(url_info->url_path, token);

  return 1;
}

int get_filename(char * path, char * filename){

  char * token = malloc(strlen(path));
  memcpy(token, path, strlen(path));
  token = strtok(path, "/");

  while(token != NULL){
          memset(filename,0,strlen(filename));
    memcpy(filename, token, strlen(token));
    token = strtok(NULL, "/");
  }
  return 1;
}
```

utilities.h

```c
#pragma once

#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 <signal.h>
#include <netdb.h>
#include <string.h>
#include <strings.h>
#include <termios.h>
#include <fcntl.h>

#define MAXDATASIZE 512

typedef struct Url {
  char * type;
  char * user;
  char * password;
  char * host;
  char * url_path;
} url;

int getInfo(char * str, url * url_info);
int get_filename(char path[100], char * filename);
```