

Redes de Computadores

2.º Trabalho Prático - Rede de Computadores

Gonçalo Teixeira e Gonçalo Alves

Mestrado Integrado em Engenharia
Informática e Computação



FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

20 de dezembro de 2020

Porto

Sumário

Sumário aqui -> dizer para que é que serve este relatório

Conteúdo

| | |
|--|----|
| Sumário | 1 |
| Introdução | 3 |
| Parte I - Aplicação Download | 4 |
| Arquitetura | 4 |
| Resultados | 4 |
| Parte II - Configuração e Análise de Rede | 5 |
| Experiência 1 - Configuração IP de Rede | 5 |
| Experiência 2 - Implementar duas LANs Virtuais no Switch | 6 |
| Experiência 3 - Configurar um Router em Linux | 6 |
| Experiência 4 - Configurar um Router comercial e implementar o NAT | 7 |
| Conclusão | 8 |
| Referências | 9 |
| Anexos | 10 |
| Imagens | 10 |
| Aplicação Download | 13 |

Introdução

Sumário aqui -> introduzir o tema

Dizer do que se trata a aplicação download

Dizer do que se trata a configurar uma rede de computadores

Parte I - Aplicação Download

Pequeno texto acerca da aplicação download

Arquitetura

Arquitetura da Aplicação Download

Resultados

Pequeno texto acerca de como foi testada a aplicação e um log de resultados

Parte II - Configuração e Análise de Rede

Pequeno texto acerca da rede a configurar

Experiência 1 - Configuração IP de Rede

Explicar sucintamente o objetivo desta experiência.

1. O que são os pacotes ARP e para que são usados?

O ARP (*Address Resolution Protocol*) é um protocolo de comunicação que serve para descobrir o endereço da camada de ligação associado ao endereço IP numa LAN (*Local Area Network*). O endereço da camada de ligação é também conhecido por Endereço MAC (*Media Access Control*).

2. Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Executando o comando *ping* do tux3 para o tux4, o tux3 envia uma pergunta para saber qual é o endereço MAC associado ao IP do tux4. A "pergunta" é feita através de um pacote ARP que contém o endereço IP e MAC do tux3 (172.16.30.1 e 00:21:5a:5a:7d:74) e o endereço IP do tux4 (172.16.1.254), uma vez que se quer descobrir o endereço MAC do tux4, o campo dedicado a esse efeito está a 00:00:00:00:00:00. De seguida é enviada uma resposta, também sob a forma de um pacote ARP, do tux4 para o tux3, indicando o seu endereço MAC (00:21:5a:5a:7d:74). Figura 1

3. Quais os pacotes gerados pelo comando ping?

Primeiro o comando *ping* gera pacotes ARP para fazer a relação entre endereços IP e MAC, de seguida gera pacotes ICMP (*Internet Control Message Protocol*).

4. Quais são os endereços MAC e IP dos pacotes ping?

Quando se executa o comando *ping* no tux3 para o tux4, os endereços (IP e MAC) vão ser os endereços dos tux. Podemos ver de seguida os endereços registados nos pacotes de pedido e resposta, respetivamente.

| | tux | MAC | IP |
|---------|-----|-------------------|---------------|
| Origem | 3 | 00:21:5a:61:24:92 | 172.16.30.1 |
| Destino | 4 | 00:21:5a:5a:7d:74 | 172.16.30.254 |

Tabela 1: Pacote de Pedido

| | tux | MAC | IP |
|---------|-----|-------------------|---------------|
| Origem | 4 | 00:21:5a:5a:7d:74 | 172.16.30.254 |
| Destino | 3 | 00:21:5a:61:24:92 | 172.16.30.1 |

Tabela 2: Pacote de Resposta

Devem-se consultar as figuras 2 e 3 para referência.

5. Como determinar a trama recetora Ethernet é ARP, IP, ICMP?

O *Ethernet Header* de um pacote contém a informação acerca do tipo da trama. Para as tramas IP, o valor do tipo será 0x0800, se o *IP Header* tiver o valor 1 então o tipo de protocolo é ICMP. Para as tramas ARP o valor do tipo será 0x0806.

Para referência devem-se consultar as figuras 4 e 5.

6. Como determinar o comprimento de uma trama recetora?

O comprimento de uma trama recetora pode ser determinado inspecionando a entrada no registo do *Wireshark*, tal como se pode observar na figura 6.

Experiência 2 - Implementar duas LANs Virtuais no Switch

Explicar sucintamente o objetivo desta experiência.

1. Como configurar a VLANy0?

Primeiro é necessário ligar um cabo série do tux3 ao switch para aceder ao terminal de configuração (*configure terminal*) do switch. De seguida cria-se uma vlan, de ID y0, no caso, 31. Por fim resta atribuir as portas em questão a essa vlan que acabou de ser criada.

```
1 configure terminal
2 > vlan y0
3 > end
4 > configure terminal
5 > interface fastethernet 0/[n da porta]
6 > switchport mode access
7 > switchport access vlan y0
8 > end
```

2. Quantos domínios de broadcast existem? O que se pode concluir a partir dos registos?

O tux3 recebe resposta do tux4 quando faz *ping broadcast*, mas não recebe do tux2. O tux2 não recebe nenhuma resposta quando executa a instrução de *ping broadcast*. Desta forma pode-se concluir que existem dois domínios de broadcast, um que contem o tux3 e tux4, e outro que contém o tux2.

Experiência 3 - Configurar um Router em Linux

Explicar sucintamente o objetivo desta experiência.

1. Que rotas existem nos tux? Qual o seu significado?

| tux | vlan | gateway |
|-----|-------------------|---------------|
| 2 | vlan0 172.16.y0.0 | 172.16.y1.253 |
| 2 | vlan1 172.16.y1.0 | 0.0.0.0 |
| 3 | vlan0 172.16.y0.0 | 0.0.0.0 |
| 3 | vlan1 172.16.y1.0 | 172.16.y1.254 |
| 4 | vlan0 172.16.y0.0 | 0.0.0.0 |
| 4 | vlan1 172.16.y1.0 | 0.0.0.0 |

Tabela 3: Rotas Existentes nos tux

O destino das rotas é até onde o tux que está na origem da rota consegue chegar.

2. Qual é a informação que uma entrada da tabela de *forwarding* contém?

Destination: o destino da rota;

Gateway: o IP do próximo ponto por onde passará a rota;

Netmask: usado para determinar o ID da rede a partir do endereço IP do destino;

Flags: informações sobre a rota;

Metric: o custo de cada rota;

Ref: número de referências para esta rota (não usado no kernel do Linux);

Use: contador de pesquisas pela rota, dependendo do uso de -F ou -C isto vai ser o número de falhas da cache (-F) ou o número de sucessos (-C);

Interface: qual a placa de rede responsável pela gateway (*eth0/eth1*).

3. Que mensagens ARP e endereços MAC associados são observados e porquê?

Tal como referido nos pontos 1 e 2 da experiência 1, quando um tux faz *ping* para outro tux é preciso relacionar o IP do destino com um endereço MAC. Pode ser consultadas a figuras 7 (*eth1 tux4*) e 8 (*eth0 tux4*).

4. Que pacotes ICMP são observados e porquê?

São observados pacotes de pedido e resposta ICMP, uma vez que as rotas estão configuradas, caso contrário seriam enviados pacotes ICMP de *Host Unreachable*. Pode ser consultada a figura 9 para referência.

5. Quais são os endereços IP e MAC associados a um pacote ICMP e porquê?

Os endereços IP e MAC associados com os pacotes ICMP são os endereços IP e MAC dos tux de origem e destino. Quando é feito *ping* do tux3 para o tux4 os endereços de origem vão ser 172.16.y0.1 (IP) e 00:21:5a:61:24:92 (MAC) e o de destino 172.16.y1.253 (IP) e 00:21:5a:5a:7d:74 (MAC).

Experiência 4 - Configurar um Router comercial e implementar o NAT

Explicar sucintamente o objetivo desta experiência.

1. Como se configura um Router estático num Router comercial?

Para configurar o Router é necessário ligar um cabo série do tux ao router, depois executam-se os seguintes comandos no *GTKTerm* (router):

```
1  configure terminal
2  > ip route [destino] [mascara] [gateway]
3  > exit
```

2. Quais são as rotas seguidas pelos pacotes durante a experiência? Explique.

No caso de a rota existir, os pacotes utilizam essa rota, caso contrário, os pacotes passam pela rota default (router), são informados que o tux4 existe, e deverão ser enviados pelo mesmo.

3. Como se configura o NAT num Router comercial?

Para configurar o router, foi necessário configurar a interface interna no processo de NAT, o que foi feito recorrendo ao guião fornecido. Ligando ao router através da porta série, utilizamos os seguintes comandos que podem ser consultados em anexo, figura 10.

4. O que faz o NAT?

O *Network Address Translation* NAT foi concebido para a preservação de endereços IP. Permite que as redes IP privadas que usem endereços IP não registados a possibilidade de se ligarem à Internet. O NAT opera num router, normalmente ligando duas redes, e traduz os endereços da rede privada (que não são únicos à escala global) em endereços válidos, antes que os pacotes sejam transmitidos para outra rede. Adicionalmente, o NAT pode ser configurado para mostrar apenas um endereço correspondente à rede privada inteira para a rede exterior. Isto transmite segurança adicional uma vez que esconde de forma eficaz os endereços da rede que está por detrás daquele endereço público. Adicionalmente, o NAT oferece também funções de segurança e é implementado em ambientes de acesso remoto.

Resumidamente, o NAT permite que os computadores de uma rede interna tenham acesso ao exterior, sendo que, um único endereço IP é exigido para representar um grupo de computadores fora da sua própria rede.

Conclusão

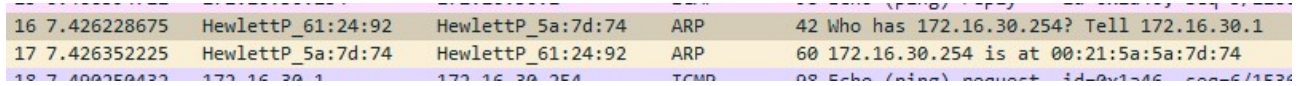
Conclusão aqui

Referências

Referências aqui

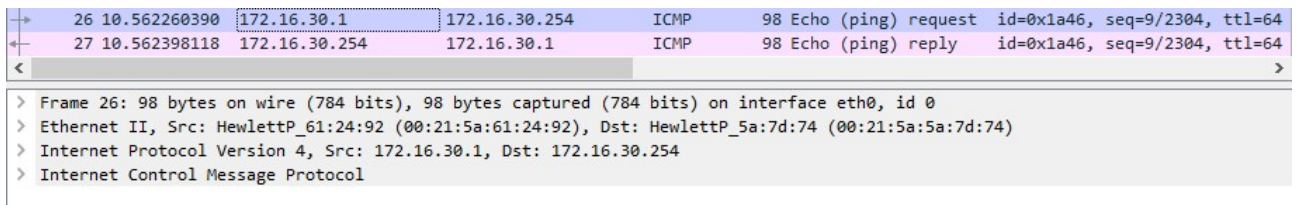
Anexos

Imagens



| | | | | | |
|----|-------------|-------------------|-------------------|------|--|
| 16 | 7.426228675 | HewlettP_61:24:92 | HewlettP_5a:7d:74 | ARP | 42 Who has 172.16.30.254? Tell 172.16.30.1 |
| 17 | 7.426352225 | HewlettP_5a:7d:74 | HewlettP_61:24:92 | ARP | 60 172.16.30.254 is at 00:21:5a:5a:7d:74 |
| 18 | 7.426352433 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) request id=0x1a46, seq=9/2304, ttl=64 |

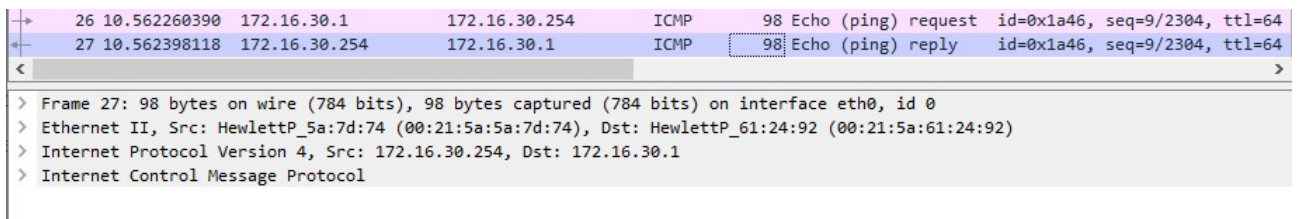
Figura 1: Pacotes ARP



| | | | | | |
|----|--------------|---------------|---------------|------|--|
| 26 | 10.562260390 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) request id=0x1a46, seq=9/2304, ttl=64 |
| 27 | 10.562398118 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x1a46, seq=9/2304, ttl=64 |

> Frame 26: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:24:92 (00:21:5a:61:24:92), Dst: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
> Internet Protocol Version 4, Src: 172.16.30.1, Dst: 172.16.30.254
> Internet Control Message Protocol

Figura 2: Pacote de Pedido



| | | | | | |
|----|--------------|---------------|---------------|------|--|
| 26 | 10.562260390 | 172.16.30.1 | 172.16.30.254 | ICMP | 98 Echo (ping) request id=0x1a46, seq=9/2304, ttl=64 |
| 27 | 10.562398118 | 172.16.30.254 | 172.16.30.1 | ICMP | 98 Echo (ping) reply id=0x1a46, seq=9/2304, ttl=64 |

> Frame 27: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74), Dst: HewlettP_61:24:92 (00:21:5a:61:24:92)
> Internet Protocol Version 4, Src: 172.16.30.254, Dst: 172.16.30.1
> Internet Control Message Protocol

Figura 3: Pacote de Resposta

```

▼ Ethernet II, Src: HewlettP_61:24:92 (00:21:5a:61:24:92), Dst: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Destination: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Source: HewlettP_61:24:92 (00:21:5a:61:24:92)
  Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 172.16.30.1, Dst: 172.16.30.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x9a46 (39494)
  > Flags: 0x40, Don't fragment
  Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)

```

Figura 4: Campo Type Pacote ICMP

```

▼ Ethernet II, Src: HewlettP_61:24:92 (00:21:5a:61:24:92), Dst: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Destination: HewlettP_5a:7d:74 (00:21:5a:5a:7d:74)
  > Source: HewlettP_61:24:92 (00:21:5a:61:24:92)
  Type: ARP (0x0806)

```

Figura 5: Campo Type Pacote ARP

```

▼ Frame 9: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
  > Interface id: 0 (eth0)
  Encapsulation type: Ethernet (1)
  Arrival Time: Nov 24, 2020 15:33:40.912061718 Hora padrão de GMT
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1606232020.912061718 seconds
  [Time delta from previous captured frame: 0.408640354 seconds]
  [Time delta from previous displayed frame: 0.408640354 seconds]
  [Time since reference or first frame: 4.418256468 seconds]
  Frame Number: 9
  Frame Length: 98 bytes (784 bits)
  Capture Length: 98 bytes (784 bits)

```

Figura 6: Tamanho de uma trama Recetora

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------------|-------------------|-------------------|----------|--------|--|
| 74 | 117.4924495... | HewlettP_a6:a4:f1 | Broadcast | ARP | 42 | Who has 172.16.21.1? Tell 172.16.21.253 |
| 75 | 117.4925883... | HewlettP_61:2b:72 | HewlettP_a6:a4:f1 | ARP | 60 | 172.16.21.1 is at 00:21:5a:61:2b:72 |
| 76 | 117.4925931... | 172.16.20.1 | 172.16.21.1 | ICMP | 98 | Echo (ping) request id=0x3316, seq=1/256, ttl=63 (reply in 77) |
| 77 | 117.4927290... | 172.16.21.1 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x3316, seq=1/256, ttl=64 (request in 76) |
| 91 | 122.6344745... | HewlettP_61:2b:72 | HewlettP_a6:a4:f1 | ARP | 60 | Who has 172.16.21.253? Tell 172.16.21.1 |
| 92 | 122.6344821... | HewlettP_a6:a4:f1 | HewlettP_61:2b:72 | ARP | 42 | 172.16.21.253 is at 00:22:64:a6:a4:f1 |

Figura 7: Pacotes ARP carta eth1 tux4

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------------|-------------------|-------------------|----------|--------|--|
| 75 | 119.6648582... | HewlettP_5a:7d:12 | Broadcast | ARP | 60 | Who has 172.16.20.254? Tell 172.16.20.1 |
| 76 | 119.6648850... | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 42 | 172.16.20.254 is at 00:08:54:50:3f:2c |
| 77 | 119.6649841... | 172.16.20.1 | 172.16.21.1 | ICMP | 98 | Echo (ping) request id=0x3316, seq=1/256, ttl=64 (reply in 78) |
| 78 | 119.6652819... | 172.16.21.1 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x3316, seq=1/256, ttl=63 (request in 77) |
| 92 | 124.9039676... | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 42 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 93 | 124.9040803... | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 60 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |

Figura 8: Pacotes ARP carta eth0 tux4

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|-------------------|-------------------|----------|--------|--|
| 5 | 5.015726702 | 172.16.20.1 | 172.16.20.254 | ICMP | 98 | Echo (ping) request id=0x31ce, seq=1/256, ttl=64 (reply in 6) |
| 6 | 5.015866604 | 172.16.20.254 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x31ce, seq=1/256, ttl=64 (request in 5) |
| 20 | 10.237511242 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 21 | 10.237532545 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |
| 40 | 25.016954288 | 172.16.20.1 | 172.16.21.253 | ICMP | 98 | Echo (ping) request id=0x31db, seq=1/256, ttl=64 (reply in 41) |
| 41 | 25.017093770 | 172.16.21.253 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x31db, seq=1/256, ttl=64 (request in 40) |
| 53 | 30.109973528 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | Who has 172.16.20.254? Tell 172.16.20.1 |
| 54 | 30.110059299 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | 172.16.20.254 is at 00:08:54:50:3f:2c |
| 75 | 47.522478676 | 172.16.20.1 | 172.16.21.1 | ICMP | 98 | Echo (ping) request id=0x31eb, seq=1/256, ttl=64 (reply in 76) |
| 76 | 47.522769445 | 172.16.21.1 | 172.16.20.1 | ICMP | 98 | Echo (ping) reply id=0x31eb, seq=1/256, ttl=63 (request in 75) |
| 91 | 52.735893146 | Netronix_50:3f:2c | HewlettP_5a:7d:12 | ARP | 60 | Who has 172.16.20.1? Tell 172.16.20.254 |
| 92 | 52.735913471 | HewlettP_5a:7d:12 | Netronix_50:3f:2c | ARP | 42 | 172.16.20.1 is at 00:21:5a:5a:7d:12 |

Figura 9: Pacotes ICMP com rotas definidas

```

1  Cisco NAT
2  http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.
   shtml
3
4  > conf t
5  > interface gigabitethernet 0/0 *
6  > ip address 172.16.y1.254 255.255.255.0
7  > no shutdown
8  > ip nat inside
9  > exit
10
11 > interface gigabitethernet 0/1*
12 > ip address 172.16.1.y9 255.255.255.0
13 > no shutdown
14 > ip nat outside
15 > exit
16
17 > ip nat pool ovrlld 172.16.1.y9 172.16.1.y9 prefix 24
18 > ip nat inside source list 1 pool ovrlld overload
19 > access-list 1 permit 172.16.y0.0 0.0.0.7
20 > access-list 1 permit 172.16.y1.0 0.0.0.7
21 > ip route 0.0.0.0 0.0.0.0 172.16.1.254
22 > ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
23 > end
24

```

Figura 10: Configuração NAT router

Aplicação Download

download.c

```
1  /*      (C)2000 FEUP      */
2
3  #include "connection.h"
4  #include "getip.h"
5
6  #define SERVER_PORT 6000
7  #define SERVER_ADDR "192.168.28.96"
8
9
10 int main(int argc, char **argv) {
11
12     if (argc != 2) {
13         printf("usage: download ftp://[<user>:<password>@]<host>/<url-path>\n");
14         return 1;
15     }
16
17     char user[1024], password[1024], host[1024], url_path[1024], *ip;
18
19     parseArg(argv[1], user, password, host, url_path);
20     printArg(user, password, host, url_path);
21     ip = getIP(host);
22     printf("IP - %s\n", ip);
23     if (ftp_init_connection(ip) == -1) return -1;
24     if (ftp_login(user, password) == -1) return -1;
25     if (ftp_download(url_path) == -1) return -1;
26
27     return 0;
28 }
29
30 void printArg(char *user, char *password, char *host, char *url_path) {
31     printf("User - %s\n", user);
32     printf("Password - %s\n", password);
33     printf("Host - %s\n", host);
34     printf("URL - %s\n", url_path);
35 }
36
37 // ./download ftp://user:1234@sftp.up.pt/pub/ficheiro.zip
38 void parseArg(char *arg, char *user, char *password, char *host, char *url_path) {
39
40
41     char *args = strtok(arg, "/");
42     args = strtok(NULL, ":");
43     strcpy(user, args);
44
45     args = strtok(NULL, "@");
46     strcpy(password, args);
47
48     args = strtok(NULL, "/");
49     if (args == NULL) {
50         printf("No User\nSetting Default - anonymous\n");
51         strcpy(host, user);
52         strcpy(user, "anonymous");
53     } else {
54         strcpy(host, args);
55     }
56
57     args = strtok(NULL, "\\0");
58     if (args == NULL) {
```

```

59     printf("No Password\nSetting Default- 1234\n");
60     strcpy(url_path, password);
61     strcpy(password, "1234");
62 } else {
63     strcpy(url_path, args);
64 }
65
66 }

```

getip.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <netdb.h>
5 #include <sys/types.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8
9 char *getIP(char host[]);

```

getip.c

```

1
2 #include "getip.h"
3
4 char *getIP(char host[]) {
5     struct hostent *h;
6
7     /*
8     struct hostent {
9         char    *h_name;   Official name of the host.
10        char    **h_aliases; A NULL-terminated array of alternate names for the host.
11        int      h_addrtype; The type of address being returned; usually AF_INET.
12        int      h_length;   The length of the address in bytes.
13        char    **h_addr_list; A zero-terminated array of network addresses for the host.
14        Host addresses are in Network Byte Order.
15    };
16
17    #define h_addr h_addr_list[0] The first address in h_addr_list.
18    */
19    if ((h = gethostbyname(host)) == NULL) {
20        perror("gethostbyname");
21        exit(1);
22    }
23    char *IP = inet_ntoa(*((struct in_addr *) h->h_addr));
24    printf("Host name   : %s\n", h->h_name);
25    printf("IP Address  : %s\n", inet_ntoa(*((struct in_addr *) h->h_addr)));
26
27    return IP;
28 }

```

connection.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <netdb.h>
5 #include <sys/types.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8
9 char *getIP(char host[]);

```


connection.c

```
1
2 #include "getip.h"
3
4 char *getIP(char host[]) {
5     struct hostent *h;
6
7     /*
8     struct hostent {
9         char      *h_name;   Official name of the host.
10        char      **h_aliases; A NULL-terminated array of alternate names for the host.
11        int        h_addrtype; The type of address being returned; usually AF_INET.
12        int        h_length;   The length of the address in bytes.
13        char      **h_addr_list; A zero-terminated array of network addresses for the host.
14                                Host addresses are in Network Byte Order.
15    };
16
17    #define h_addr h_addr_list[0] The first address in h_addr_list.
18    */
19    if ((h = gethostbyname(host)) == NULL) {
20        perror("gethostbyname");
21        exit(1);
22    }
23    char *IP = inet_ntoa(*(struct in_addr *) h->h_addr);
24    printf("Host name   : %s\n", h->h_name);
25    printf("IP Address  : %s\n", inet_ntoa(*(struct in_addr *) h->h_addr));
26
27    return IP;
28 }
```

Makefile

```
1 CC = gcc
2 CFLAGS = -Wall -pthread -Wno-pointer-sign -g
3 DEPS = connection.h getip.h
4 OBJ = connection.o getip.o
5 TARGETS = download
6
7 all: download
8
9 %.o: %.c $(DEPS)
10     @$(CC) $(CFLAGS) -c -o $@ $<
11     @echo $@
12
13 download: $(OBJ)
14     @$(CC) $(CFLAGS) -o $@ $@.c $(OBJ) -lm
15     @echo $@
16
17 clean:
18     @rm *.o $(TARGETS)
```