Redes de Computadores - FTP e configuração da rede do laboratório

Miguel Rodrigues (up201906042@edu.fe.up.pt)
Nuno Castro (up202003324@edu.fe.up.pt)

Conteúdo

Sumário

O segundo trabalho prático da unidade curricular de Redes de Computadores coloca especial ênfase nas camadas de topo do modelo OSI, em particular as camadas de rede e de aplicação.

Numa primeira parte, abordar-se-á o desnvolvimento do cliente FTP proposto e onde se enquadra a camada de aplicação, referida acima. Depois, serão expostos todos os detalhes relevantes sobre a configuração da rede IP, onde se encaixa a camada de rede, também ela referida no parágrafo anterior.

Desenvolvimento do cliente FTP

O desenvolvimento do cliente FTP resultou numa aplicação simples para a descarga de ficheiros com base no protocolo FTP escrita na liguagem C.

Telnet

O inicio da interação com o protocolo FTP deu-se com o Telnet - um programa que funciona como uma interface para o protocolo com o mesmo nome e que possibilita a transferência de dados entre dispositivos diferentes numa mesma rede.

Establecimento da ligação

Em todo o caso, uma conexão com o servidor FTP inicia-se da seguinte forma:

- `$ telnet ftp.up.pt 21`.

Ou seja, estamos a iniciar uma ligação com o servidor `ftp.up.pt` na porta `21` - a padrão para o protocolo FTP.

Autenticação

Após o começo da ligação o servidor envia uma mensagem de boas-vindas. De seguida, o cliente deve autenticar-se fornecendo um nome de utilizador e a respetiva palavra-passe. Os comandos necessários ao processo de autenticação são:

- `USER <username>`
- `PASS <password>`

Uma nota importante é a de que este processo é ocorre para todas as ligações, no caso de se especificar o utilizador `anonymous` podemos depois fornecer qualquer palavra-passe e assim efetuar a autenticação, desde que suportado pelo servidor FTP. Além disso, observa-se que as palavra-passe no protocolo FTP circulam de forma não encriptada, o que imdiatamente levanta questões de segurança.

Estado da ligação

A conexão com o servidor FTP carateriza-se também pelo seu estado. Esse estado pode ser **ativo**, ou seja, o servidor toma a iniciativa e liga-se ao cliente, ou então **passivo**, onde é o cliente quem se liga ao servidor.

Por defeito, a ligação FTP está no modo ativo mas, para a finalidade pretendida, esta pode e deve ser alterada para o modo passivo com o comando `PASV` - uma vez que será pedido um recurso ao servidor, por iniciativa do cliente.

Pedido ao servidor

Quando o modo da ligação FTP é alterado o servidor fornece a informação ao cliente para que este se conecte numa nova porta. É a partir dessa nova ligação que os dados serão transferidos ao cliente.

Deste modo, o cliente deve conectar-se com o servidor nessa nova porta, sendo que, naquela onde efetuou a autenticação deve agora executar o comando `RETR <resource-path>`. Se tudo decorrer sem erros, o servidor deve responder com uma mensagem alertando para o fim da transferência dos dados.

Programa `download`

O programa `download` deve ser capaz de excutar os passos de acordo com o que foi discutido acima. Este programa deve ser invocado da seguinte forma:

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

Felizmente, para a comunicação entre diferentes dispositivos na *internet* existe uma abstração disponível - o *socket* - e que representa cada um dos canais de comunicação. Assim, o algoritmo a seguir deve ser algo como:

1. Leitura do URL fornecido;
2. Abertura do *socket* de autenticação;
3. Comandos `USER` e `PASS`;
4. Comando `PASV`;
5. Leitura das informações necessárias ao establecimento da nova ligação;
6. Abertura do *socket* de transferência;
7. Comando `RETR` via *socket* de autenticação;
8. Leitura dos dados via *socket* de transferência;
9. Fecho de ambos os *sockets* e por sua vez da ligação.

Arquitetura da aplicação e execução

A aplicação encontra-se estruturada de uma forma simples. No ficheiro `download.c` encontramos, essencialmente, o algoritmo descrito acima. Já nos ficheiros `connection.h` e `connection.c` encontramos uma interface com cada um dos procedimentos inerentes ao protocolo FTP, documentada e que pode ser consultada em detalhe nos anexos fornecidos.

A execução do programa apenas fornece *output* em caso de erro, terminando de forma imediata. Caso contrário, o ficheiro é transferido para o diretório com o mesmo nome do ficheiro no servidor FTP. Um exemplo de execução bem sucedido deve ser idêntico a algo como:

```
$ download ftp://ftp.up.pt/pub/kodi/timestamp.txt
$ ls timestamp.txt
-rw-r--r-- 1 user user 11 Jan 22 23:32 timestamp.txt
```

Também é possível fornecer os dados de autenticação pelo argumento da linha de comandos:

```
$ download ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4
$ ls -lh crab.mp4
-rw-r--r-- 1 user user 85M Jan 22 23:26 crab.mp4
```

Configuração da rede do laboratório

A seguinte secção demonstra os diferentes aspetos relativos à configuração da rede do laboratório. Um aspeto importante a ter em conta é o de que os comandos apresentados são válidos para a bancada 4 da sala I321.

Experiência 1

Nesta experiência, o objetivo é atribuir o endereço IP ao *tux3* e *tux4*.

Configuração

Em ambos os computadores é necessário efetuar um *reset* na interface em que se pretende configurar o endereço IP - por uma questão de simplicidade utiliza-se a interface `eth0`:

- `# ifconfig eth0 down`.

Em seguida, configura-se os endereços para cada um dos *tux* de acordo com o diagrama fornecido no enunciado:

- `# ifconfig eht0 up 172.16.40.1/24`, no *tux3*;
- `# ifconfig eht0 up 172.16.40.254/24`, no *tux4*.

Para testar a ligação entre os 2 computadores usa-se o comando `ping` - um utilitário que utiliza o protocolo ICMP e testar a conectividade entre equipamentos. O seguinte *output* indica que a configuração foi concretizada corretamente.

```
# ping 172.16.40.254
PING 172.16.40.254 (172.16.40.254) 56(84) bytes of data
64 bytes from 172.16.40.254: icmp_seq=1 ttl=64 time=0.315 ms
64 bytes from 172.16.40.254: icmp_seq=2 ttl=64 time=0.156 ms
```

Agora, é também momento de inspecionar as tabelas `route` e `arp` em cada *tux*. Os comandos são, respetivamente, `route -n` e `arp -a`.

Análise dos *logs*

Depois da sequência de comandos apresentada anteriormente, é importante apagar todos os registos das tabelas ARP para que no *wireshark* os pacotes ARP sejam capturados. Eis o comando necessário:

- `# arp -del 172.16.40.254`, no *tux3*.

  What are the ARP packets and what are they used for?

  What are the MAC and IP addresses of ARP packets and why?

Depois de apagadas as entradas da tabela ARP no *tux3*. Verifica-se que existem pacotes ARP a perguntar quem tem o endereço `172.40.16.1` logo após a primeira resposta ICMP. A resposta a

este pacote contém o endereço MAC do alvo dessa resposta, o que permite a entrega do pacote de resposta ICMP que se encontrava em espera.

Assim, é possível constatar que os pacotes ARP quando enviados, são enviados a todos os dispositivos conectados à mesma rede - *broadcast*. Como a sigla o indica, ARP (*Adress Resolution Protocol*) é um protocolo que permite reconhecer qual o dispositivo com um determinado endereço IP quando esse endereço faz parte da mesma rede. Como se referiu em cima, a resposta é o endereço MAC que distingue os dispositivos dentro da mesma rede e que, por isso, deve ser único.

What packets does the ping command generate?

O comando `ping` gera pacotes ICMP (*Internet Control Message Protocol*). Eis algumas das características destes pacotes:

- Comprimento de 98 *bytes*;
- Tipo (byte 34):
  - `0x00` no caso de um `reply`;
  - `0x08` no caso de um `request`;
- Número de sequência (bytes 40 e 41);
- *Checksum* (bytes 36 e 37).

What are the MAC and IP addresses of the ping packets?

Os endereços IP e MAC presentes nos pacotes ICMP identificam, respetivamente, as redes de origem e de destino e os dispositivos de origem e de destino dentro dessas redes.

How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

É possível determinar o tipo de *Ethernet frame* de acordo com os *bytes* 13 e 14:

- IPv4 (`0x0800`);
- ARP (`0x0806`).

Nota para o facto de que o *byte* 23, nos pacotes ICMP, identifica o protocolo e tem o valor `0x01`, contudo este valor já não faz da *Ethernet frame*.

How to determine the length of a receiving frame?

O comprimento de uma *frame* recebida é a soma do comprimento da *Ethernet frame* (14 *bytes*) e do comprimento da *IP frame* - valor que pode ser inspecionado nos *bytes* 16 e 17.

What is the loopback interface and why is it important?

A *loopback interface* é uma interface virtual e que está constantemente ativa. O protocolo IP reserva o endereços `127.0.0.0/8` para esta interface. Todavia, a maioria das implementações do protocolor utilizam o endereço `127.0.0.1` para IPv4 e `::1` para IPv6, sendo o nome padrão o `localhost`.

A principal finalidade desta interface é redirecionar os pacotes de volta para a sua origem. Apesar de aparentar ser uma tarefa simples a sua importância é determinante para testar a infraestrutura da rede e garantir que os diferentes pontos da rede transmitem e que consequentemente permite também testar se o transporte dos pacotes se faz da forma correta.

Experiência 2

Nesta experiência, o objetivo é configurar as 2 VLANs (*Virtual Local Area Network*) de acordo com o esquema fornecido pelo enunciado.

Configuração das VLAN

How to configure vlanY0?

A configuração das VLANs é extremamente simples. Eis os comandos necessários para criar a `vlan40`:

```
sw> enable
sw# configure terminal
sw# vlan 40
sw(config)# end
```

Agora, com a VLAN criada, é o momento de adicionar cada um dos *tux* à VLAN recentemente criada. Eis mais uma sequência de comandos a executar no switch:

```
sw# configure terminal
sw(config)# interface fastethernet 0/<port for tux3>
sw(config-if)# switchport mode access
sw(config-if)# switchport access vlan 40
sw(config-if)# end
sw(config)# interface fastethernet 0/<port for tux4>
sw(config-if)# switchport mode access
sw(config-if)# switchport access vlan 40
sw(config-if)# end
```

Depois da introdução dos comandos acima, a configuração da VLAN está terminada. A verificação da configuração pode ser inspecionada com o comando:

- `sw# show vlan brief`.

Com este comando são listadas as VLANs criadas no switch e ainda as respetivas portas para cada uma dessas VLANs.

Análise dos *logs*

How many broadcast domains are there? How can you conclude it from the logs?

Antes de testar a configuração feita na secção anterior, é importante entender o conceito de VLAN, pois desse modo a pergunta torna-se bem menos complicada.

Assim, uma VLAN define uma rede local virtual. Isto significa, que por exemplo podemos dividir uma rede física em sub-redes independentes entre si de uma maneira lógica.

Um exemplo prático é o de uma organização com vários departamentos. Como se pode facilmente concluir, cada departamento poderá ter uma ou mais VLANs associadas - dividindo os dispositivos nessas redes de uma forma lógica. Outra vantagem sobre as VLANs é a de que estas facilitam também a distribuição dos pacotes de rede - tornando-a mais eficiente.

Com o conceito de VLAN discutido é agora momento de verificar a configuração feita acima. Portanto, a partir do *tux3* executamos o `ping` para o *tux4* e verificamos que ambos os computadores estão conectados. No entanto, o mesmo não se sucede quando o `ping` é executado para o *tux2* - surge a mensagem `Network unreachable`.

Agora, para um último teste executamos um `ping` com a opção *broadcast* (`-b`) nos computadores *tux3* e *tux2*. A opção *broadcast* envia um pacote para todos os dispositivos de uma determinada rede - pelo endereço próprio para o efeito o `.255` (para IPv4). Nota ainda, para o facto de que, por padrão, os *hosts* Linux não responderem aos pacotes de *broadcast* ICMP, ou seja, é preciso executar previamente o comando:

- `# echo 0 > /proc/sys/ipv4/icmp_echo_ignore_broadcasts`.

Analisando as capturas efetuadas, surgem 2 observações pertinentes:

1. Quando o `ping` tem origem no *tux3* (`172.16.40.1`) obtemos uma resposta do *tux4* (`172.16.40.254`);
2. Quando o `ping` tem origem no *tux2* (`172.16.41.1`) nunca obtemos qualquer resposta.

Deste modo, e tendo em conta o esquema fornecido, verifica-se que para cada VLAN está associado um domínio de *broadcast*. Portanto, com 2 VLANs (`vlan40` e `vlan41`) existem 2 domínios de *broadcast*.

Experiência 3

Nesta experiência, o objetivo é compreender a configuração do router CISCO e determinar os aspetos mais relevantes da mesma, bem como, o funcionamento do DNS (*Domain Server Name*).

Análise da configuração do router

How to configure a static route in a commercial router?

A configuração de uma rota estática no router apenas requer 1 único comando, pelo que é extremamente simples:

- `rtr# ip route 0.0.0.0 0.0.0.0 172.16.254.1`

Por exemplo, o comando acima define uma *default route* com a *gateway* em `172.16.254.1`, qualquer pacote sem rota definida na tabela do router seguirá para o endereço `172.16.254.1`.

How to configure NAT in a commercial router?

Outra vez, a configuração da NAT no router requer um par de comandos:

1. Em modo priveligiado, acedemos à interface pretendida, por exemplo:
   - `rtr(config)# interface FastEthernet0/1`;
2. Configuramos o tipo de NAT:
   - `rtr(config-if)# nat <inside | outside>`.

What does NAT do?

NAT, por extenso, *Network Addresss Translation* funciona como um intermediário entre uma rede interna e a internet traduzindo os endereços privados em endereços públicos - para onde os dispositivos externos a essa rede possam enviar os pacotes pretendidos.

De um lado da NAT gere-se os endereços privados (`nat inside`), do lado oposto (`nat outside`) lida-se com endereços públicos.

Contudo, para que a NAT funcione devidamente é necessário que os pacotes que por lá passam possuam um campo com o endereço MAC e outro com a porta que possa distinguir tanto a origem como o alvo dos pacotes. Isto acontece porque na internet os dispositivos na rede interna possuem o mesmo endereço IP - no caso do diagrama fornecido o `172.16.30.2`.

Configuração do DNS

DNS, sigla para *Domain Name System*, é um sistema hierárquico e distribuído para a resolução de nomes de domínio, por outras palavras, traduz um nome (`google.com`) num endereço IP (`142.250.200.142`).

How to configure the DNS service at an host?

A configuração do DNS pode ser feita de forma estática no ficheiro `/etc/hosts`, ou então no ficheiro `/etc/resolv.conf` onde é possível alterar o servidor DNS.

What packets are exchanged by DNS and what information is transported?

Os pacotes de DNS trocados na durante a resolução de um nome de domínio representam as interrogações efetuadas ao sistema DNS. Alguns dos componentes que fazem parte do pacote são:

- `NAME`: nome do domínio;
- `TYPE`: tipo de RR (*resource record*);
- `CLASS`: código de classe (`0x0001` para `IN` ou internet).

Linux *routing*

What ICMP packets are observed and why?

A presença dos pacotes ICMP gerados pelo `traceroute` surge de uma maneira muito peculiar e particular. Essencialmente, a forma com que o `traceroute` executa a sua função é incrementando progressivamente o valor do campo TTL (*Time To Leave*), presente no datagrama UDP, e enviando o número de pacotes necessários até atingir o destino pretendido.

O valor TTL decrementa em cada *hop* até 0, quando atinge esse valor o pacote é descartado e uma mensagem é enviada de volta ao emissor com dados relativos a essa falha. No wireshark surge, também, a seguinte mensagem:

- `Time-to-live exceeded (Time to live exceeded in transit)`.

What are the IP and MAC addresses associated to ICMP packets and why?

Mais uma vez, a presença dos endereços IP e MAC, tanto da origem como do destino, nos pacotes ICMP acontece de modo a permitir a correta distribuição desses pacotes.

What routes are there in your machine? What are their meaning?

As rotas presentes numa máquina Linux podem ser visualizadas pelo comando:

- `route -n`.

Eis o *output* gerado:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
104.17.113.188  192.168.1.1     255.255.255.255 UGH   0      0        0 eno2
```

Neste caso em particular, todos os pacotes com destino a `104.17.113.188/32`, o que significa exatamente o endereço `104.17.113.188` deve seguir pela *gateway*, ou seja, pelo endereço `192.168.1.1`.

Experiência 4

Nesta experiência, ocorre o culminar de todas as experiências anteriores. O objetivo aqui é configurar totalmente a rede interna com as VLANs e a NAT para que os computadores dessa rede interna possar aceder à internet.

Linux router

What routes are there in the tuxes? What are their meaning?

What information does an entry of the forwarding table contain?

Após a configuração das VLANs e a introdução das rotas necessárias, fornecidas no enunciado, o comando `route -n` deve ser executado de modo a listar as rotas em cada um dos *tux*. Para o *tux3*, por exemplo as rotas devem ser as seguintes:

```
Kernel IP routing table
Destination     Gateway         Genmask       Flags Metric  Ref   Use Iface
0.0.0.0         172.16.40.254   0.0.0.0       U     0       0       0 eth0
172.16.40.0     0.0.0.0         255.255.255.0 U     0       0       0 eth0
172.16.41.0     172.16.40.254   255.255.255.0 U     0       0       0 eth0
```

Como se observa existem 3 rotas cada uma com o seu significado:

1. *Default route*. Rota usada caso o destino do pacote não encontre correspondência em nehuma outra - usada pelos pacotes com um destino fora da rede interna;
2. Rota para a própria rede, ou seja, para a mesma VLAN - vlan40;
3. Rota para vlan41, onde se encontra o *tux2*.

What ARP messages, and associated MAC addresses, are observed and why?

As mensagens ARP, como vimos na experiência 1, permitem determinar a quem um determinado pacote deve ser entregue, dentro de uma mesma rede.

Aqui esse tipo de mensagens surge, pois, as entradas das tabelas ARP foram previamente apagadas. Desse modo, os pacotes ARP surgem quando ocorre o encaminhamento de um pacote mas o próximo *hop* ainda não está em cache - não existe uma entrada na tabela ARP para esse *hop*. Finalmente, a presença dos endereços MAC nesses pacotes é discutida na experiência 1.

Cisco router

What are the paths followed by the packets in the experiments carried out and why?

Os caminhos seguidos pelos pacotes dependem da tabela de encaminhamento em cada um dos computadores.

Quando ambos os computadores encontram-se na mesma VLAN, o encaminhamento é direto - baseado no endereço MAC de destino. Quando esse não é o caso, é preciso fazer uma análise mais cuidada. Por exemplo, um pacote que tenha origem no *tux3* e cujo destino seja o *tux2* ou outro computador na internet precisa obrigatoriamente de sair pelo *tux4*. Depois as rotas do *tux4* ditam qual a próxima paragem - *tux2* ou o router.

Conclusão

Concluindo, este segundo trabalho prático foi um grande passo no que toca à aprendizagem sobre redes IP. O balanço é muito positivo, uma vez que, todos os objetivos propostos pelo corpo docente da unidade curricular - mesmo sendo desafiantes - foram alcançados com êxito.

Referências

- Modelo OSI
- Network Programming
- Address Resolution Protocol
- Virtual LAN
- Network Address Translation
- Domain Name System

Anexos

Código-fonte

download.c

```c
/**
* download.c
* ftp client
* Authors: Miguel Rodrigues & Nuno Castro
* RC @ L.EIC 2122
*/

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "connection.h"


int
main (int argc, char **argv)
{
        if (argc != 2) {
                fprintf(stderr,
                        "usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
                        argv[0]);
                return 1;
        }

        int sock_auth, sock_retr, auth, retr;
        URL *url_auth, *url_retr;
        FILE *f = NULL;

        url_auth = geturl(argv[1], FTP_PORT);
        sock_auth = start(url_auth);
        assert(sock_auth > 0);

        auth = login(sock_auth, url_auth);
        assert(auth == 0);

        url_retr = passive(sock_auth, url_auth);
        assert(url_retr != NULL);

        sock_retr = start(url_retr);
        assert(sock_retr > 0);
        free(url_retr);

```

```
        retr = retrieve(sock_auth, sock_retr, url_auth, f);
        assert(retr == 0);
        free(url_auth);

        stop(sock_retr);
        stop(sock_auth);

        return 0;
}
```

connection.h

```
/**
* connection.h
* ftp client
* Authors: Miguel Rodrigues & Nuno Castro
* RC @ L.EIC 2122
*/

#ifndef _CONNECTION_H_
#define _CONNECTION_H_

#include <sys/socket.h>
#include <sys/types.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <assert.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


/* macros */
#define FTP_PORT 21

/* typedefs */
typedef struct url URL;


/**
 * geturl - Parses the given url and port filling a URL structure
 * @url: an url string
 * @port: a port in which the connection should be made
 *
```

```
 * The main function should call this with port 21 aka FTP_PORT and
 * should call free on the returned pointer after using it
 * Return: URL*
 */
URL *geturl(const char *url, const unsigned short port);


/**
 * start - Starts the connection with the ftp server.
 * @u: a URL* for a struct given by geturl()
 *
 * Return: file descriptor for the opened socket
 */
int start(const URL *u);

/**
 * stop - Stops the connection with the ftp server.
 * @sockfd: file descriptor of the socket hosting the connection
 *
 * Must be called in order to finish a connection successfully
 */
void stop(int sockfd);


/**
 * login - Logins in the ftp server
 * @sockfd: file descriptor of the socket hosting the connection
 * @u: a URL* for a struct given by geturl()
 *
 * Return: 0 if successful <0 value otherwise
 */
int login(int sockfd, const URL *u);

/**
 * passive - Enters in passive mode
 * @sockfd: file descriptor of the socket hosting the connection
 * @u: a URL* for a struct given by geturl()
 *
 * The main function should call free on the returned pointer after using it
 * Return: a new URL* to allow open the retrieve connection of ftp (RFC 959)
 */
URL *passive(int sockfd, const URL *u);

/**
 * retrieve - Retrieves the desired file from the ftp server
 * @sockfd_auth: file descriptor of the socket hosting the connection
 * @sockfd_retr: file descriptor of the socket hosting the connection
 * @u: a URL* for a struct given by geturl()
```

```
 * @fp: file to be downloaded
 *
 * Return: 0 if successful <0 value otherwise
 */
int retrieve(int sockfd_auth, int sockfd_retr, const URL *u, FILE *fp);

#endif /* _CONNECTION_H_ */
```

connection.c

```
/**
* connection.c
* ftp client
* Authors: Miguel Rodrigues & Nuno Castro
* RC @ L.EIC 2122
*/

#include "connection.h"

/* macros */
#define MAX_LINE_LEN 1024

/* enums */
typedef enum {
        USER,
        PASS,
        PASV,
        RETR,
        QUIT
} CMD;

typedef enum {
        OPEN        = 150,
        ACCEPT      = 220,
        TRANSFER    = 226,
        PASSIVE     = 227,
        LOGIN       = 230,
        PASS_SPEC   = 331
} CODE;

/* structs */
struct url {
        char           user[32];
        char           pass[64];
        char           host[32];
        unsigned short port;
        char           path[512];
};
```

```
39
40  /* globals */
41  static const char cmds[][5] = { "USER", "PASS", "PASV", "RETR", "QUIT" };
42  static const char anon[] = "anonymous";
43
44
45  URL *
46  geturl(const char *url, const unsigned short port)
47  {
48          URL *u;
49          u = calloc(1, sizeof(URL));
50          assert(u != NULL);
51
52          char l[128];
53          sscanf(url, "ftp://%128[^/]/%512s", l, u->path);
54
55          strncpy(u->host, l, strlen(l));
56          strncpy(u->user, anon, 10);
57          strncpy(u->pass, "", 1);
58          u->port = port;
59
60          if (strchr(l, '@'))
61                  sscanf(l, "%32[^:]:%64[^@]@%32s", u->user, u->pass, u->host);
62
63          return u;
64  }
65
66  static unsigned short
67  response(int sockfd, char *info, size_t infolen)
68  {
69          unsigned short code;
70          int s;
71          FILE *fp;
72          char line[MAX_LINE_LEN];
73
74  /*
75   *  Here we really need to kkep track of the file descriptor open
76   *  by the connect(). This means that is needs to be duplicated here
77   *  just because of the response, otherwise when this function is done
78   *  and calls fclose() the file descriptor will be closed as well.
79   *
80   *  More info is available in fclose() man page:
81   *  The fclose() function shall perform the equivalent of a close() on
82   *  the file descriptor that is associated with the stream pointed to
83   *  by stream.
84   */
85          s = dup(sockfd);
86          fp = fdopen(s, "r");
```

```
87          assert(fp != NULL);
88
89          do {
90                  char *buffer;
91                  buffer = fgets(line, sizeof(line), fp);
92                  assert(buffer == line);
93          } while(line[3] != ' ');
94
95          sscanf(line, "%hu [^\r\n]\r\n", &code);
96          if (info != NULL)
97                  strncpy(info, line, infolen);
98
99          fclose(fp);
100         return code;
101 }
102
103 static void
104 command(int sockfd, CMD cmd, const char *arg)
105 {
106         char fmt[strlen(arg)+8];
107         ssize_t wb;
108
109         snprintf(fmt, sizeof(fmt), "%s %s\r\n", cmds[cmd], arg);
110         wb = send(sockfd, fmt, strlen(fmt), 0);
111         assert(wb >= 0);
112 }
113
114
115 int
116 start(const URL *u)
117 {
118         struct addrinfo hints, *res, *r;
119         int s, connection = 1;
120         char port[6];
121
122         memset(&hints, 0, sizeof(hints));
123         hints.ai_family = AF_UNSPEC;
124         hints.ai_socktype = SOCK_STREAM;
125         snprintf(port, sizeof(port), "%hu", u->port);
126
127         getaddrinfo(u->host, port, &hints, &res);
128         for (r = res; r != NULL; r = res->ai_next) {
129                 s = socket(r->ai_family, r->ai_socktype, r->ai_protocol);
130                 if (s < 0)
131                         continue;
132
133                 connection = connect(s, r->ai_addr, r->ai_addrlen);
134                 if (!connection)
```

```
135                         break;
136                 close(s);
137         }
138
139         return !connection ? s : -1;
140 }
141
142 void
143 stop(int sockfd)
144 {
145         command(sockfd, QUIT, "");
146         close(sockfd);
147 }
148
149
150 int
151 login(int sockfd, const URL *u) {
152         unsigned short resp;
153
154         resp = response(sockfd, NULL, 0);
155         if (resp != ACCEPT)
156                 return -ACCEPT;
157
158         command(sockfd, USER, u->user);
159         resp = response(sockfd, NULL, 0);
160         if (resp == LOGIN)
161                 return 0;
162         if (resp != PASS_SPEC)
163                 return -PASS_SPEC;
164
165         command(sockfd, PASS, u->pass);
166         if (response(sockfd, NULL, 0) != LOGIN)
167                 return -LOGIN;
168
169         return 0;
170 }
171
172 URL *
173 passive(int sockfd, const URL *u)
174 {
175         char info[MAX_LINE_LEN], data[24], addr[INET_ADDRSTRLEN+6];
176         unsigned char ip[6];
177
178         URL *url;
179
180         command(sockfd, PASV, "");
181         if (response(sockfd, info, sizeof(info)) != PASSIVE)
182                 return NULL;
```

```
183
184         sscanf(info, "%*[^(](%24[^)]).\r\n", data);
185         sscanf(data, "%hhu,%hhu,%hhu,%hhu,%hhu,%hhu",
186                     &ip[0], &ip[1], &ip[2], &ip[3], &ip[4], &ip[5]);
187         snprintf(addr, sizeof(addr), "ftp://%hhu.%hhu.%hhu.%hhu/",
188                     ip[0], ip[1], ip[2], ip[3]);
189
190         url = geturl(addr, ip[4] * 256 + ip[5]);
191         if (strncmp(u->user, anon, strlen(anon)) != 0) {
192                 strncpy(url->user, u->user, strlen(u->user));
193                 strncpy(url->pass, u->pass, strlen(u->pass));
194         }
195
196         return url;
197 }
198
199 int
200 retrieve(int sockfd_auth, int sockfd_retr, const URL *u, FILE *fp)
201 {
202         char *filename, info[MAX_LINE_LEN];
203         unsigned char fragment[1024];
204
205         ssize_t rb;
206         size_t fsize;
207
208         filename = strrchr(u->path, '/') + 1;
209         fp = fopen(filename, "wb");
210         assert(fp != NULL);
211
212         command(sockfd_auth, RETR, u->path);
213         if (response(sockfd_auth, info, sizeof(info)) != OPEN) {
214                 fclose(fp);
215                 return -OPEN;
216         }
217
218         sscanf(info, "%*[^(](%zu bytes).\r\n", &fsize);
219
220         do {
221                 rb = recv(sockfd_retr, fragment, sizeof(fragment), 0);
222                 fwrite(fragment, 1, rb, fp);
223         } while (rb > 0);
224
225         fclose(fp);
226         if (response(sockfd_auth, NULL, 0) != TRANSFER)
227                 return -TRANSFER;
228
229         return 0;
230 }
```

setup.sh

```
#!/bin/sh

ifconfig eth0 down
ifconfig eth1 down

# tux 2
if [[ $1 -eq 2 ]]
    ifconfig eth0 up 172.16.41.1/24
    route add -net 172.16.40.0/24 gw 172.16.41.253
    route add default gw 172.16.41.254
fi

# tux 3
if [[ $1 -eq 3 ]]
    ifconfig eth0 up 172.16.40.1/24
    route add -net 172.16.41.0/24 gw 172.16.40.254
    route add default gw 172.16.40.254
fi

# tux 4
if [[ $1 -eq 4 ]]
    ifconfig eth0 up 172.16.40.254/24
    ifconfig eth1 up 172.16.41.253/24
    echo 1 > /proc/sys/net/ipv4/ip_forward
    echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
    route add default gw 172.16.41.254
fi
```