Aplicação de download e configuração e estudo uma rede

Relatório

Universidade do Porto

Faculdade de Engenharia

FEUP

Redes de Computadores

3º ano

Mestrado Integrado em Engenharia Informática e Computação

Turma 4

Carolina Moreira	201303494	up201303494@fe.up.pt
Daniel Fazeres	201502846	up201502846@fe.up.pt
José Peixoto	200603103	ei12134@fe.up.pt

27 de Janeiro de 2017

Conteúdo

Resumo

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposto o desenvolvimento de uma aplicação que permitisse o download de ficheiros usando a especificação de FTP e a configuração e estudo de uma rede de computadores.

1 Aplicação de download

A primeira parte do segundo trabalho laboratorial consistiu no desenvolvimento de uma aplicação de download recorrendo ao protocolo de transferência de ficheiros FTP especificado pelo `RFC959`. Para este efeito foi também necessário resolver o endereço de IP para um dado URL de acordo com especificação RFC1738. Separaram-se as componentes do parsing do URL e do cliente de download usando o protocolo FTP respectivamente nos ficheiros `url.c` e `ftp.c`.

Casos de uso principais

O programa implementa uma versão básica de um cliente de FTP com suporte para download de ficheiros de forma anónima ou para um dado par utilizador e password, introduzidos antecipadamente ao caminho de URL do ficheiro. Apesar da introdução do nome de utilizador seguido de password serem facultativos, em alguns casos tornam-se obrigatórios na transferência com sucesso de um ficheiro por FTP caso este não esteja disponível de forma pública e requeira autenticação por parte do utilizador.

```
[user@localhost ftp-downloader]$ ./bin/ftp-downloader
Usage: ./bin/ftp-downloader ftp://[<user>:<password>@]<host>/<url-path>
```

Figura 1: Utilização do programa

Análise de URL

Após a leitura e compreensão do `RFC1738`, desenhou-se uma estrutura de dados com a finalidade de armazenar a informação extraída da análise de um link URL recebido pela linha de comandos. Quando fornecidos, são guardados na estrutura referida, o nome de utilizador, a password, o host, ip, path e nome do ficheiro em arrays de caracteres independentes. Para além disso, é também definida a porta 21 como a predefinida na ligação de controlo do protocolo de FTP. À semelhança do que é referido no `RFC1738`, caracteres capitalizados são interpretados como caracteres minúsculos, admitindo ftp da mesma forma que FTP.

Para se poder resolver o endereço de IP armazenado na estrutura de URL, é feita uma chamada à função `gethostbyname` que permite a obtenção do endereço de uma máquina a partir do nome e retorna uma estrutura do tipo `hostent` contendo o endereço na variável `h_addr` que é posteriormente convertido para um array de chars com o auxílio da função `inet_ntoa`.

Cliente de FTP

O cliente de FTP liga-se através de um socket TCP ao servidor de FTP identificado pelo endereço de IP acima mencionado e à porta 21 e estabelece uma ligação de controlo de comunicação, comunicando com uma sequência de comandos de transferência FTP que lhe são enviados ao estilo do protocolo Telnet:

USER user envio do nome de utilizador sob a forma de uma string que identifica o utilizador no servidor remoto.

PASS pass envio da palavra passe completando a identificação no sistema de identificação e controlo de acesso do servidor.

CWD path indicação do directório que contém o ficheiro requerido para download e sobre o qual se pretende trabalhar.

PASV comando que pede ao servidor para ficar à escuta numa porta de dados diferente da porta usada pelo serviço de controlo. Este comando recebe uma resposta que contém o endereço e a porta na qual o servidor ficou à escuta para poder estabelecer uma outra ligação TCP usada para transferência de dados.

RETR filename comando `retrieve` que pede ao servidor que inicie a transmissão de uma cópia do ficheiro especificado pelo campo `filename` usando a nova ligação estabelecida ao endereço e porta recebidos pelo comando anterior.

Uma vez estabelecida a ligação dedicada de transmissão de dados, estes vão sendo recebidos de forma ordenada pelo cliente de FTP e armazenados em disco.

Casos de uso

Um possível caso de uso pode ser o download de um ficheiro de forma anónima do URL `ftp://ftp.up.pt/pub/CentOS/filelist.gz`:

```
[user@localhost ftp-downloader]$ ./bin/ftp-downloader ftp://ftp.up.pt/pub/CentOS/filelist.gz

The IP received to ftp.up.pt was 193.136.37.8
220 Bem-vindo à Universidade do Porto
Bytes sent: 16
Info: USER anonymous

331 Please specify the password.
Bytes sent: 7
Info: PASS

230 Login successful.
Bytes sent: 17
Info: CWD pub/CentOS/

250 Directory successfully changed.
Bytes sent: 6
Info: PASV

227 Entering Passive Mode (193,136,37,8,169,170)
IP: 193.136.37.8
PORT: 43434
Bytes sent: 18
Info: RETR filelist.gz

150 Opening BINARY mode data connection for filelist.gz (3722620 bytes).
226 File send OK.
Bytes sent: 6
Info: QUIT
```

Figura 2: Utilização anónima para download de um ficheiro

Também foi testado o download de um ficheiro quando é requerida a autenticação do utilizador no servidor de FTP:

```
[user@localhost ftp-downloader]$ ./bin/ftp-downloader ftp://ei12134[redacted]@tom.fe.up.pt/public_html/event_mana
ger/index.php

The IP received to tom.fe.up.pt was 192.168.50.138
220 FTP for Alf/Tom/Crazy/Pinguim
Bytes sent: 14
Info: USER ei12134

331 Please specify the password.
Bytes sent: 20
Info: PASS [redacted]

230 Login successful.
Bytes sent: 32
Info: CWD public_html/event_manager/

250 Directory successfully changed.
Bytes sent: 6
Info: PASV

227 Entering Passive Mode (192,168,50,138,106,139).
IP: 192.168.50.138
PORT: 27275
Bytes sent: 16
Info: RETR index.php

150 Opening BINARY mode data connection for index.php (48 bytes).
226 Transfer complete.
Bytes sent: 6
Info: QUIT
```

Figura 3: Utilização autenticada no download de um ficheiro

2 Experiência 1: Configuração de uma rede IP

Nesta experiência, configuraremos apenas uma rede simples de dois computadores ligados entre si por um switch. Esta rede usará o conjunto de protocolos que constituem, em parte, a arquitectura da mais conhecida rede de todas, a Internet. Como é sabido, o protocolo que gere a camada de rede da Internet é o IP (Internet Protocol). Para além do próprio protocolo IP, a Internet usa vários outros protocolos auxiliares para regular o seu funcionamento, nomeadamente ARP, ICMP, e DHCP. O protocolo DHCP é usado para negociar automaticamente a atribuição de um endereço IP a qualquer elemento de uma rede que permita uma identificação única perante outros membros. Este protocolo não será usado, e em vez disso atribuir-se-á manualmente e estaticamente endereços IP a cada um dos computadores das redes desta e subsequente experiências deste trabalho, o que é exequível visto tratarem-se de redes de pequena dimensão, o que não seria o caso noutras instalações de aplicação prática no mundo real.

Os endereços desta experiência serão divididos, como é feito neste protocolo de rede, numa parte respeitante à subrede (subnet), e outra que indicará o endereço de cada computador dentro desta rede. Um pequeno esquema do resultado pretendido é indicado na próxima imagem.

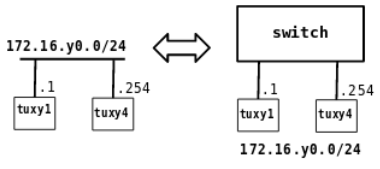

Figura 4: Rede da 1a Experiência

Configuração de interfaces

Na configuração desta rede, começamos por ligar apenas os computadores TUX1 e TUX4 ao switch Cisco Catalyst 3560. Cada um dos computadores tem duas interfaces expostas, para além da loopback. A interface loopback é uma interface apresentada pelo sistema operativo como qualquer outra interface, mas possuíndo o endereço especial 127.0.0.1 (IPv4) e :: 1 (IPv6). Pacotes dirigidos a este endereço são direccionados para o próprio computador. Cada interface pode ser configurada independentemente das outras, e o software de rede do sistema operativo encarregar-se-á de encaminhar o tráfego de rede para uma ou outra consoante a rede a que pertence e o destino do tráfego. Por esta razão, e uma vez que a configuração será manual, atribuiremos um endereço IP juntamente com uma máscara de rede à interface eth0 destes TUXs.

O primeiro passo é desactivar a interface eth1 com o comando

$$ifconfig\ eth1\ down$$

Numa nota à parte, este relatório foi completado depois de uma reconfiguração dos computadores por parte dos responsáveis pela manutenção do laboratório, pelo que, no caso de alguns TUXs, o programa usado para configurar as interfaces foi o *ip* e não o $ifconfig$, para o qual os comandos são diferentes. Para desactivar esta interface com o *ip* usámos o comando

$$ip\ link\ set\ eth1\ down$$

Seguidamente, removemos os IPs atribuídos ao TUX1 com:

$$ip\ addr\ flush\ dev\ eth0$$

após o que atribuímos o endereço pretendido:

$$ip\ addr\ add\ 172.16.10.1/24\ dev\ eth0$$

e confirmamos o resultado com:

$$ip\ addr\ show\ [dev\ eth0]$$

Nos resultados podemos ver a interface loopback já mencionada, o IP que configurámos na interface eth0, e que a MTU desta interface é 1500 bytes, que é o tamanho máximo de uma frame em ethernet, que é o tipo de rede (camada de rede) que estamos a utilizar. Realizamos o mesmo procedimento no TUX4, atribuindo o endereço IP com:

$$ifconfig\ eth0\ 172.16.10.254/24$$

Configuração do switch

O próximo passo é configurar o switch que liga os dois computadores. Para aceder à sua configuração, ligamos o seu terminal a um adaptador cuja saída é uma ligação em série, que depois é ligada à entrada de porta de série do TUX1, exposta no ficheiro $/dev/ttyS0$ do mesmo. Para termos a certeza que não existem configurações já definidas a interferir no nosso trabalho, fazemos reset do switch. Utilizando o programa *gtkterm* para comunicar com a sua linha de comandos, começamos por entrar em modo de configuração com o comando *enable*, para o qual precisamos de introduzir a password do switch. Escrevemos, depois, os comandos:

$$copy\ tftp : //192.168.109.1/2\ startup - config$$

$$delete\ flash : vlan.dat$$

$$reload$$

Rotas

O comando *reload* demora algum tempo a terminar. Durante este intervalo, podemos verificar as rotas de cada um dos computadores com o programa *route*. Podemos observar que temos uma única entrada para interface na sua tabela.) A coluna de destino (Destination) juntamente com a máscara de rede (Genmask), determinam para que endereços a sua linha será aplicada. Neste caso, visto a máscara de rede ser 255.255.255.0 (ou seja, usamos os primeiros três grupos de 8 bytes, ou 24 bits, conforme indicámos nos comandos anteriores), esta entrada será usada para todos os IPs que começam por 172.16.10. Um valor de Gateway nesta tabela indica que o pc enviará pacotes para o endereço de destino directamente em vez de um intermediário, que é o predendido, visto estarem ambos ligados entre si. O switch utilizado redireciona pacotes para as entradas apropriadas, verificando os IPs (é um switch activo ao invés de passivo), mas não recebe pacotes como se fosse um host normal – ou seja, a sua existência é transparente no processo de routing – por isso não aparece nesta tabela de routing.

VLAN

No entanto, para ter os TUXs a comunicar entre si, temos de nos assegurar que estão na mesma VLAN (a VLAN 1, default) do switch. Podemos inspeccionar isto com o comando *show vlan brief* no switch em modo normal.

Ligámos o TUX1 à entrada 1 do switch, que a designa por $Fa0/1$ (abreviatura de Fastethernet 1, visto ser uma entrada de Fastethernet) e o TUX2 à entrada dois. Nos resultados do switch, vemos que estão os dois na mesma rede.

2.1 Pacotes ARP e endereços MAC

Em qualquer rede deste tipo, o protocolo *ARP* (Address Resolution Protocol) desempenhará a função necessária de comunicar a atribuição de endereços IP entre vizinhos – NICs (Network Interface Cards), como placas Ethernet, não compreendem endereços IP, emitindo apenas tramas endereçadas pelo ao que se chamam endereços MAC (Media Access Control), atribuídos pela IEEE e garantidamente únicos em qualquer parte do mundo. Quando um computador numa rede deste tipo deseja saber a que endereço MAC corresponde um determinado endereço IP, emite (através de broadcast) um pacote ARP com a pergunta "a quem pertence este endereço IP?". De seguida, o dono do endereço referido responde à questão com outro pacote. O emissor do primeiro pedido de ARP também inclui o seu próprio endereço IP na mensagem, eliminando a necessidade do segundo membro da rede perguntar ele próprio para que endereço de IP deve responder. De notar que os pacotes utilizados nesta troca são pacotes IP, e não simplesmente tramas de Ethernet.

Cada membro da rede mantém em cache uma tabela de atribuições de endereços IP, mas esta deve ter uma duração finita, para lidar com reconfigurações na rede e na sua topologia. Quando um computador é configurado inicialmente, pode emitir um pedido ARP pelo seu próprio endereço. Não deverá ocorrer nenhuma resposta, mas todos os seus vizinhos deverão receber a informação relativa ao endereço IP do emissor.

Este protocolo é definido no RFC 826.

Para podermos observar pacotes ARP, apagaremos a tabela de ARP dos dois computadores com o comando $arp\ -d$.

2.2 Comandos PING

3 Experiência 2: Duas VLANs num switch

Nesta experiência foram configuradas duas VLAN's no switch. A primeira VLAN é constituída pelo TUX1 e pelo TUX4 e a segunda pelo TUX2. Criou-se no switch a vlan0 e adicionou-se as portas correspondentes ao

TUX1 e TUX4. Criou-se a vlan1 e adicionou-se a porta correspondente ao TUX2. Para adicionar uma nova VLAN ao switch acedeu-se pela consola de configuração e executou-se o comando `vlan [n]`. Após a criação da VLAN foi-se adicionando as portas do switch associadas com o comando `interface fastethernet 0/[i]`, seguido do comando `switchport access VLAN [n]`.

Uma vez configuradas as VLANs, testou-se a nova configuração pingando o TUX2 quer da TUX1 quer do TUX4. Como o TUX2 faz parte de uma sub-rede diferente, o comando falhou como seria de esperar.

4 Experiência 3: Configurar um router em Linux

Nesta experiência configurou-se o TUX4 por forma a transformá-lo num router que faz de ponte entre as duas sub-redes atrás mencionadas. Configurou-se a segunda carta de rede `eth1` do TUX4 atribuindo-lhe o IP 172.16.11.253 pertencente à segunda sub-rede. Adicionaram-se as rotas apropriadas ao TUX1 e ao TUX2 para que conseguissem comunicar entre si. Para o efeito, no TUX1 executou-se o comando `route add -net 172.16.11.0/24 gw 172.16.10.254` em que o primeiro endereço identifica a gama de endereços para a qual se quer adicionar a rota e o segundo, é o endereço IP do TUX4 para o qual se deve reencaminhar o pacote. AO TUX2 adicionou-se a nova rota com o comando `route add -net 172.16.10.0/24 gw 172.16.11.253`, em que o IP 172.16.11.253 é o IP previamente atribuído ao TUX4 nesta segunda sub-rede. Por fim, pingou-se com sucesso o TUX2 a partir do TUX1 e vice-versa, confirmando a configuração como correcta, na qual os pacotes são reencaminhados da entre as sub-redes com o recurso ao TUX4 que opera como um router.

5 Experiência 4: Configurar um router comercial e implementar NAT

Nesta experiência pretendia-se configurar um router comercial e implementar NAT. Para o efeito, ligou-se o router a uma porta do switch e adicionou-se esta porta à configuração da segunda VLAN atrás referida. Acedeu-se ao router usando a consola de configuração e executou-se o comando `interface fastethernet 0/0` para configurar esta interface e atribuiu-se o IP para o router na VLAN1 de 172.16.11.254 com o comando `ip address 172.16.11.219 255.255.255.0`. De seguida, configurou-se a interface externa atribuindo-lhe o IP com o comando `ip address 172.16.1.19 255.255.255.0`. Para assegurar a atribuição de uma determinada gama de endereços executaram-se os comandos: `ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix 24` e `ip nat inside source list 1 pool vorld overload`. Definiram-se também rotas internas e externas com o comando `ip route 0.0.0.0 0.0.0.0 172.16.1.254` e `ip route 172.16.10.0 255.255.255.0`

172.16.11.253. Este comando redireciona os pacotes que tenho IP de destino dentro da gama 172.16.10.0-255 para o IP 172.16.11.253.

6 Experiência 5: DNS

Nesta experiência pretendia-se configurar o domain name system conhecido pela abreviatura DNS. Este serviço permite que um endereço de domínio seja traduzido num endereço IP válido correspondente. Por exexemplo www.google.pt pode ser traduzido para o endereço de IP 195.8.12.84.

6.1 Como configurar um serviço DNS?

Para configurar o serviço de DNS adicionou-se uma linha contendo o endereço de um servidor de DNS ao ficheiro de configuração das rotinas de resolução de endereços da internet da biblioteca de C localizado em /etc/resolv.conf.

nameserver 172.16.1.1

A palavra-chave nameserver é seguida do valor que neste caso é um endereço de IPv4 para um servidor de DNS.

6.2 Pacotes usados por DNS

Para testar a configuração atrás referida, usou-se o comando ping para o endereço de domínio www.google.pt e observou-se quer o output na linha de comandos quer os pacotes trocados com o servidor de DNS usando o wireshark. É feito um pedido ao servidor de DNS para que este lhe forneça informações que tenha acerca do endereço de domínio enviado, neste caso www.google.pt e o servidor responde com o tempo de vida, o tamanho dos dados e com o valor do endereço de IP neste caso com 4 byes e com o valor do endereço de IP.

DNS query:

$$www.google.pt:\ type\ A,\ class\ IN$$

DNS answer:

$$www.google.pt:\ type\ A,\ class\ IN,\ addr\ 195.8.11.212$$

7 Experiência 6: Ligações TCP

Nesta experiência compilou-se a aplicação apresentada no início do relatório com o fim de efetuar uma transferência de um ficheiro alojado num servidor de ftp e analisar o tráfego de rede usando o Wireshark. Uma vez finalizado o download do ficheiro, verificou-se a sua integridade, confirmando que continha os dados que eram expectáveis, confirmando deste modo que a transferência tinha sido feita sem problemas. Este download com sucesso também permitiu confirmar que era possível resolver um endereço de domínio e aceder a um IP externo a rede configurada. Analisando o tráfego, detetaram-se pelo menos duas ligações de TCP uma que corresponde à ligação de controlo e outra para transfência de dados FTP-DATA. Averiguou-se também que o TCP utiliza o Selective Repeat ARQ, em que o receptor continua a processar frames recebidos mesmo quando detecta algum erro.

8 Conclusões

Findo o projeto, consideramos que atingimos os objetivos básicos estipulados. Esta abordagem prática permitiu uma melhor consciência do funcionamento e configuração de uma rede.

Referências

[1] Andrew S. Tanenbaum, David J. Wetherall, *Computer Networks*, Prentice Hall, 5th edition, 2011.

[2] Network Working Group, *Architectural Principles of the Internet*, June 1996.

A Código fonte

main.c

```
1  #include "ftp.h"
2  #include "url.h"
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define INVALID_PORT -1
8  #define USE_IPV6 0
9
10 void print_usage(char* program)
11 {
12     fprintf(stdout,
```

```
13              "Usage: %s ftp://[<user>:<password>@]<host>/<url
                    -path> \
14              [-p PORT]\n",
15              program);
16  }
17
18  int get_url_from_args(int argc, char** argv, url* dest_url)
19  {
20      int port_option = INVALID_PORT;
21      char* url_option = NULL;
22      if (argc == 1) {
23          print_usage(argv[0]);
24          return 1;
25      }
26
27      for (int i = 1; i < argc; ++i) {
28          if (strcasecmp(argv[i], "-p") == 0) {
29              if (argc != 4 || i == argc - 1) {
30                  print_usage(argv[0]);
31                  return 1;
32              }
33              i += 1;
34              for (char* s = argv[i]; *s != '\0'; ++s) {
35                  if (!isdigit(*s)) {
36                      print_usage(argv[0]);
37                      return 2;
38                  }
39              }
40              port_option = atoi(argv[i]);
41          } else {
42              url_option = argv[i];
43          }
44      }
45
46      // Uniform resource locator parser
47      url url;
48      init_url(&url);
49      if (parse_url(&url, url_option))
50          return -1;
51
52      if (USE_IPV6) {
53          if (get_host_ipv6(&url)) {
54              printf("Error: Cannot find ip to hostname %s.\n"
                        , url.host);
55              return -1;
56          }
57      } else {
58          if (get_host_ipv4_new(&url)) {
59              printf("Error: Cannot find ip to hostname %s.\n"
                        , url.host);
60              return -1;
61          }
62      }
63
```

```
64      if (port_option != INVALID_PORT) {
65          url.port = port_option;
66      }
67      *dest_url = url;
68
69      return 0;
70  }
71
72  void abort_connection(ftp* ftp, char* msg, int ret)
73  {
74      //fprintf(stderr,msg);
75      printf(msg);
76      disconnect_ftp(ftp);
77      exit(ret);
78  }
79
80  int main(int argc, char** argv)
81  {
82      url url;
83      int get_url_ret = get_url_from_args(argc, argv, &url);
84      if (get_url_ret != 0) {
85          return get_url_ret;
86      }
87      printf("The IP received to %s was %s:%d\n", url.host,
            url.ip, url.port);
88
89      // File transfer protocol client
90      ftp ftp;
91      if (connect_ftp(&ftp, url.ip, url.port)) {
92          return 2;
93      }
94
95      const char* user = strlen(url.user) ? url.user : "
            anonymous";
96      const char* password = strlen(url.password) ? url.
            password : "";
97
98      // Sending credentials to server
99      if (login_ftp(&ftp, user, password)) {
100         printf("Error: Cannot login user %s\n", user);
101         abort_connection(&ftp, "Exiting\n", -1);
102     }
103
104     // Changing directory
105     if (cwd_ftp(&ftp, url.path)) {
106         printf("Error: Cannot change directory to the folder
                of %s\n",
107                url.filename);
108         abort_connection(&ftp, "Exiting\n", 1);
109         return -1;
110     }
111
112     // Entry in passive mode
113     if (passive_ftp(&ftp)) {
```

```
114         abort_connection(&ftp, "Error: Cannot entry in
                passive mode\n", 1);
115     }
116
117     // Begins transmission of a file from the remote host
118     if (retr_ftp(&ftp, url.filename)) {
119         abort_connection(&ftp, "Exiting\n", 1);
120     }
121
122     // Starting file transfer
123     if (download_ftp(&ftp, url.filename)) {
124         abort_connection(&ftp, "Exiting\n", 1);
125     }
126
127     // Disconnecting from server
128     disconnect_ftp(&ftp);
129
130     return 0;
131 }
```

ftp.c

```
1  #include "ftp.h"
2  #include <time.h>
3
4  #define NO_NUMBER 0
5
6  /* Call connect() on ip and port (which are in host byte
       order.) */
7  static int connect_socket(const char* ip, int port)
8  {
9      // open a TCP socket
10     int sockfd;
11     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
12         perror("socket()");
13         return -1;
14     }
15
16     // server address handling
17     struct sockaddr_in server_addr;
18     bzero((char*)&server_addr, sizeof(server_addr));
19     server_addr.sin_family = AF_INET;
20
21     // 32 bit Internet address network byte ordered:
22     server_addr.sin_addr.s_addr = inet_addr(ip);
23
24     // server TCP port must be network byte ordered:
25     server_addr.sin_port = htons(port);
26
27     // connect to the FTP server
28     if (connect(
29             sockfd,
30             (struct sockaddr*)&server_addr,
31             sizeof(server_addr)) < 0) {
```

```
32          perror("connect()");
33          return -1;
34      }
35      return sockfd;
36  }
37
38  int connect_ftp(ftp* ftp, const char* ip, int port)
39  {
40      printf("Connecting to %s:%d...\n", ip, port);
41      int socketfd;
42      if ((socketfd = connect_socket(ip, port)) < 0) {
43          fprintf(stderr, "Error: Cannot connect socket.\n");
44          return 1;
45      }
46
47      ftp->control_socket_fd = socketfd;
48      ftp->data_socket_fd = 0;
49
50      /* read first message */
51      char rd[1024];
52      if (read_ftp(ftp, rd, sizeof(rd))) {
53          fprintf(stderr, "Error: read_ftp failure.\n");
54          return 1;
55      }
56
57      return 0;
58  }
59
60  int ftp_command(ftp* ftp, const char* cmd, const char* args,
        char* rep, const int reply1, const int reply2)
61  {
62      char s[1024];
63
64      if (args != NULL) {
65          sprintf(s, "%s %s\r\n", cmd, args);
66      } else {
67          sprintf(s, "%s\r\n", cmd);
68      }
69      if (send_ftp(ftp, s, strlen(s))) {
70          fprintf(stderr, "Error: send_ftp failure.\n");
71          return 1;
72      }
73      if (read_ftp(ftp, s, sizeof(s))) {
74          fprintf(stderr, "Error: read_ftp failure.\n");
75          return 1;
76      }
77      if (rep != NULL) {
78          strcpy(rep, s);
79      }
80
81      char num[4];
82      strncpy(num, s, 3);
83      num[3] = '\0';
84      if (atoi(num) != reply1 && atoi(num) != reply2) {
```

```
85          fprintf(stderr, "Error: Reply code %s != %d || %d\n"
                , num, reply1, reply2);
86          return 2;
87      }
88      return 0;
89  }
90
91  int login_ftp(ftp* ftp, const char* user, const char*
        password)
92  {
93      // send the user
94      if (ftp_command(ftp, "USER", user, NULL, 331, NO_NUMBER)
            ) {
95          return 1;
96      }
97      // send the password
98      if (ftp_command(ftp, "PASS", password, NULL, 230,
            NO_NUMBER)) {
99          return 2;
100     }
101     return 0;
102 }
103
104 int list_ftp(ftp* ftp, char* path)
105 {
106     path = strlen(path) == 0 ? path : ".";
107     if (ftp_command(ftp, "LIST", NULL, NULL, 125, 150)) {
108         return 1;
109     }
110     return 0;
111 }
112
113 int cwd_ftp(ftp* ftp, const char* path)
114 {
115     if (strlen(path) == 0) {
116         fprintf(stderr, "CWD: Empty path.\n");
117         return 0;
118     }
119     if (ftp_command(ftp, "CWD", path, NULL, 250, NO_NUMBER))
            {
120         return 1;
121     }
122     return 0;
123 }
124
125 int passive_ftp(ftp* ftp)
126 {
127     char reply[256];
128
129     if (ftp_command(ftp, "PASV", "", reply, 227, NO_NUMBER))
            {
130         return 1;
131     }
132
```

```
133     // starting process information
134     int ipPart1, ipPart2, ipPart3, ipPart4;
135     int port1, port2;
136     if ((sscanf(reply, "227 Entering Passive Mode (%d,%d,%d
            ,%d,%d,%d)",
137                     &ipPart1, &ipPart2, &ipPart3, &ipPart4, &
                            port1, &port2))
138         < 0) {
139         fprintf(stderr,
140                 "Error: Cannot process information to
                        calculating port.\n");
141         return 1;
142     }
143
144     char ipstr[256];
145
146     // format IP address
147     if ((sprintf(ipstr, "%d.%d.%d.%d", ipPart1, ipPart2,
            ipPart3, ipPart4))
148         < 0) {
149         fprintf(stderr, "Error: Cannot form IP address.\n");
150         return 1;
151     }
152
153     // calculating new port
154     int portResult = port1 * 256 + port2;
155
156     fprintf(stderr, "IP: %s\n", ipstr);
157     fprintf(stderr, "PORT: %d\n", portResult);
158
159     if ((ftp->data_socket_fd = connect_socket(ipstr,
            portResult)) < 0) {
160         fprintf(stderr,
161                 "Error: Incorrect file descriptor associated
                        to ftp data socket fd.\n");
162         return 1;
163     }
164
165     return 0;
166 }
167
168 int retr_ftp(ftp* ftp, const char* filename)
169 {
170     return ftp_command(ftp, "RETR", filename, NULL, 125,
            150);
171 }
172
173 int download_ftp(ftp* ftp, const char* filename)
174 {
175     FILE* file;
176     if (!(file = fopen(filename, "w"))) {
177         fprintf(stderr, "Error: Cannot open file.\n");
178         return 1;
179     }
```

```
180
181     char buf[1024];
182     int bytes;
183     long progress = 0;
184     time_t t = time(NULL);
185     while ((bytes = read(ftp->data_socket_fd, buf, sizeof(
            buf)))) {
186         progress += bytes;
187         if (time(NULL) - t > 1) {
188             printf("Downloaded %ld B\n", progress);
189             t = time(NULL);
190         }
191         if (bytes < 0) {
192             fprintf(stderr,
193                     "Error: Nothing was received from data
                            socket fd.\n");
194             return 1;
195         }
196
197         if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
198             fprintf(stderr, "Error: Cannot write data in
                        file.\n");
199             return 2;
200         }
201     }
202     fclose(file);
203     close(ftp->data_socket_fd);
204
205     char s[1024];
206     if (read_ftp(ftp, s, sizeof(s))) {
207         fprintf(stderr, "Error: read_ftp failure.\n");
208         return 1;
209     }
210
211     return 0;
212 }
213
214 int disconnect_ftp(ftp* ftp)
215 {
216     if (ftp_command(ftp, "QUIT", NULL, NULL, 221, 226)) {
217         return 1;
218     }
219     if (ftp->control_socket_fd) {
220         return close(ftp->control_socket_fd);
221     }
222     return 0;
223 }
224
225 int send_ftp(ftp* ftp, const char* msg, size_t size)
226 {
227     int bytes;
228
229     if ((bytes = write(ftp->control_socket_fd, msg, size))
            <= 0) {
```

```
230         fprintf(stderr, "Warning: Nothing was sent.\n");
231         return 1;
232     }
233
234     fprintf(stderr, "Message (%d bytes): %.*s\n", bytes, (
        int)size - 1, msg);
235
236     return 0;
237 }
238
239 int read_ftp(ftp* ftp, char* str, size_t size)
240 {
241     FILE* fp = fdopen(ftp->control_socket_fd, "r");
242
243     do {
244         memset(str, 0, size);
245         str = fgets(str, size, fp);
246         if (str == NULL) {
247             return 1;
248         }
249         fprintf(stdout, "%s\n", str);
250     } while (!('1' <= str[0] && str[0] <= '5') || str[3] !=
        ' ');
251
252     return 0;
253 }
```

url.c

```
1  #include "url.h"
2
3  static char* process_until_char(char* str, char chr);
4
5  void init_url(url* url)
6  {
7      // fill with zero
8      memset(url->user, 0, sizeof(url_content));
9      memset(url->password, 0, sizeof(url_content));
10     memset(url->host, 0, sizeof(url_content));
11     memset(url->path, 0, sizeof(url_content));
12     memset(url->filename, 0, sizeof(url_content));
13     // default port
14     url->port = 21;
15 }
16
17 const char* USER_PW_REGEX = "ftp://[A-Za-z0-9]+:([A-Za-z0
       -9])+@([A-Za-z0-9.~-])+/([[A-Za-z0-9/~._-])+";
18 const char* ANONYMOUS_REGEX = "ftp://([A-Za-z0-9.~-])+/([[A-
       Za-z0-9/~._-])+";
19
20 int parse_url(url* url, const char* URLSTR)
21 {
22     const char USER_SEPARATOR = '@';
23
```

```
24      /* copy url string to temporary */
25      char* tempURL = (char*)malloc(strlen(URLSTR) + 1 +
            strlen("ftp://"));
26      tempURL[0] = '\0';
27      if (strncmp(URLSTR,"ftp://",strlen("ftp://")) != 0) {
28          strcpy(tempURL,"ftp://");
29      }
30      memcpy(tempURL + strlen(tempURL), URLSTR, strlen(URLSTR)
            + 1);
31
32      /* Use password? */
33      int use_password;
34      char* active_regex;
35      if (strchr(tempURL, USER_SEPARATOR) != NULL) { // find
            separator
36          printf("URL: Using password\n");
37          use_password = 1;
38          active_regex = (char*)USER_PW_REGEX;
39      } else {
40          use_password = 0;
41          active_regex = (char*)ANONYMOUS_REGEX;
42      }
43
44      /* Check validity of URL against regex */
45      regex_t* regex = (regex_t*)malloc(sizeof(regex_t));
46      int reti = regcomp(regex, active_regex, REG_EXTENDED);
            // compile regex
47      if (reti) {
48          perror("URL regex error");
49          return 1;
50      }
51      size_t nmatch = strlen(URLSTR);
52      regmatch_t pmatch[nmatch];
53      if ((reti = regexec(regex, tempURL, nmatch, pmatch,
            REG_EXTENDED)) != 0) {
54          perror("URL regex mismatch");
55          fprintf(stderr, "URL: %s\n", tempURL);
56          return 1;
57      }
58      free(regex);
59
60      // removing ftp:// from string
61      char* s = malloc(sizeof(char) * (strlen(tempURL) + 1));
62      strcpy(s, tempURL + 6);
63      strcpy(tempURL, s);
64      free(s);
65
66      /*
67       * Write to URL struct:
68       */
69
70      char* element = (char*)malloc(strlen(URLSTR) + 1);
71      if (use_password) {
72          // saving username
```

```
73          strcpy(element, process_until_char(tempURL, ':'));
74          memcpy(url->user, element, strlen(element) + 1);
75
76          // saving password
77          strcpy(element, process_until_char(tempURL, '@'));
78          memcpy(url->password, element, strlen(element) + 1);
79      }
80
81      // Setting host
82      strcpy(element, process_until_char(tempURL, '/'));
83      memcpy(url->host, element, strlen(element) + 1);
84
85      // Setting URL path
86      char* path = (char*)malloc(strlen(tempURL) + 1);
87      path[0] = '\0';
88      int startPath = 1;
89      while (strchr(tempURL, '/')) {
90          element = process_until_char(tempURL, '/');
91
92          if (startPath) {
93              startPath = 0;
94              strcpy(path, element);
95          } else {
96              strcat(path, element);
97          }
98
99          strcat(path, "/");
100     }
101
102     strcpy(url->path, path);
103
104     // Setting filename
105     strcpy(url->filename, tempURL);
106
107     free(tempURL);
108     free(element);
109
110     //  fprintf(stdout, "\n%s\n%s\n%s\n%s\n%s\n", url->user,
            url->password,
111     //      url->host, url->path, url->filename);
112
113     return 0;
114 }
115
116 int get_host_ipv4(url* url)
117 {
118     struct hostent* h;
119
120     if ((h = gethostbyname(url->host)) == NULL) {
121         perror("get_host_ip");
122         return 1;
123     }
124
125     //printf(stdout, "Host name  : %s\n", h->h_name);
```

```
126     //fprintf(stdout, "IP Address : %s\n", inet_ntoa(*((
            struct in_addr *) h->h_addr)));
127
128     /* "inet_ntoa()" converts a numeric address (in network
            byte order) to the
129      * IPv4 numbers-and-dots representation.  */
130     char* ip = inet_ntoa(*((struct in_addr*)h->h_addr));
131     strcpy(url->ip, ip);
132     return 0;
133 }
134
135 int get_host_ipv4_new(url* url)
136 {
137     struct addrinfo hints;
138     memset(&hints, 0, sizeof(hints));
139     hints.ai_family = AF_INET;
140     hints.ai_socktype = SOCK_STREAM;
141     hints.ai_protocol = 0;
142     hints.ai_flags = AI_PASSIVE;
143
144     if (url->host == NULL) {
145         printf("url host is NULL\n");
146     }
147
148     struct addrinfo* res;
149     if (getaddrinfo(url->host, "ftp", &hints, &res) != 0) {
150         printf("Error in getaddrinfo; %s\n",url->host);
151     }
152
153     struct sockaddr* sockaddr_var = res->ai_addr;
154     struct sockaddr_in* sockaddr_in_var = (struct
            sockaddr_in*)sockaddr_var;
155     struct in_addr in_addr_var = sockaddr_in_var->sin_addr;
156     char* ip = inet_ntoa(in_addr_var);
157     printf("ip: %s\n", ip);
158     strcpy(url->ip, ip);
159     return 0;
160 }
161
162 int get_host_ipv6(url* url)
163 {
164     struct hostent* h;
165
166     if ((h = gethostbyname(url->host)) == NULL) {
167         perror("get_host_ip");
168         return 1;
169     }
170
171     //  fprintf(stdout, "Host name  : %s\n", h->h_name);
172     //  fprintf(stdout, "IP Address : %s\n", inet_ntoa(*((
            struct in_addr *) h->h_addr)));
173
174     /* "inet_ntoa()" converts a numeric address (in network
            byte order) to the
```

```
175         * IPv4 numbers-and-dots representation.  */
176         char* ip = inet_ntoa(*((struct in_addr*)h->h_addr));
177         strcpy(url->ip, ip);
178         return 0;
179     }
180
181     static char* process_until_char(char* str, char chr)
182     {
183         // using temporary string to process substrings
184         char* tempStr = (char*)malloc(strlen(str));
185
186         // calculating length to copy element
187         // eg, copy @pass/abc, compute length, subtract from
                length of string
188         int index = strlen(str) - strlen(strcpy(tempStr, strchr(
                str, chr)));
189
190         tempStr[index] = '\0'; // termination char in the end of
                 string
191         strncpy(tempStr, str, index);
192
193         // delete from the beginning of string
194         strcpy(str, str + strlen(tempStr) + 1);
195
196         return tempStr;
197     }
```