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

18 de Dezembro de 2016

Conteúdo

Resumo

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposto o desenvolvimento de uma aplicação que testasse um protocolo de ligação de dados criado de raiz, transferindo um ficheiro recorrendo à porta de série $RS-232$. O trabalho permitiu praticar conceitos teóricos no desenho de um protocolo de ligação de dados como o sincronismo e delimitação de tramas, controlo de erros, controlo de fluxo recurso a mecanismos de transparência de dados na transmissão assíncrona.

Findo o projeto, notou-se a importância dos mecanismos que asseguram tolerância a falhas fornecidos pela camada de ligação de dados, uma vez que a camada física não é realmente fiável.

1 Introdução

O objetivo do trabalho realizado nas aulas laboratoriais da disciplina de Redes de Computadores é a implementação de um protocolo de ligação de dados que permita praticar conhecimentos acerca de transmissões de dados entre computadores, programando em baixo nível as características comuns a este tipos de protocolos como a transparência na transmissão de dados de forma assíncrona e organização da informação sob a forma de tramas.

2 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 enderço 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 sêquencia 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 commando `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

3 Conclusões

O projeto pode ser sumariamente descrito pelo seu principal propósito que é o desenvolvimento de um protocolo de ligação de dados e o seu teste aquando da obtenção de sucesso na transferência de ficheiros entre dois computadores, com mecanismos de recuperação face a algumas situações de erros.

Findo o projeto, consideramos que atingimos os objetivos definidos tendo também implementado alguns dos elementos de valorização especificados no guião. Esta abordagem prática permitiu também uma melhor consciência do funcionamento e dos problemas inerentes às redes comunicações entre computadores, abordados nas aulas teóricas.

Referências

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

A Código fonte

A.1 Camada de aplicação

netlink.c

```
1  void receiver_stats()
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5  #include <stdio.h>
6  #include <string.h>
7  #include <stdlib.h>
8  #include <termios.h>
9  #include <unistd.h>
10
11 #include "packets.h"
12 #include "file.h"
13 #include "netlink.h"
14 #include "serial_port.h"
15
16 struct file file_to_send;
17 int max_retries = 3;
18
19 void help(char **argv)
20 {
21   fprintf(stderr, "Usage: %s [OPTIONS] <serial port>\n",
       argv[0]);
22   fprintf(stderr, "\n Program options:\n");
23   fprintf(stderr, "  -t <FILEPATH>\t\ttransmit file over the
       serial port\n");
24   fprintf(stderr, "  -i\t\t\ttransmit data read from stdin\n
       ");
25   fprintf(stderr, "  -b <BAUDRATE>\t\tbaudrate of the serial
       port\n");
26   fprintf(stderr,
27       "  -p <DATASIZE>\t\tmaximum bytes of data transfered
           each frame\n");
28   fprintf(stderr, "  -r <RETRY>\t\tnumber of retry attempts\
       n");
29 }
30
31 int parse_serial_port_arg(int index, char **argv)
32 {
33   if ((strcmp("/dev/ttyS0", argv[index]) != 0)
34       && (strcmp("/dev/ttyS1", argv[index]) != 0)
35       && (strcmp("/dev/ttyS4", argv[index]) != 0)) {
36     fprintf(stderr, "Error: bad serial port value\n");
37     return -1;
38   }
39
40   return index;
41 }
42
```

```
43  int parse_baudrate_arg(int baurdate_index, char **argv)
44  {
45    if (strcmp("B50", argv[baurdate_index]) == 0) {
46      serial_port_baudrate = B50;
47      return 0;
48    } else if (strcmp("B75", argv[baurdate_index]) == 0) {
49      serial_port_baudrate = B75;
50      return 0;
51    } else if (strcmp("B110", argv[baurdate_index]) == 0) {
52      serial_port_baudrate = B110;
53      return 0;
54    } else if (strcmp("B134", argv[baurdate_index]) == 0) {
55      serial_port_baudrate = B134;
56      return 0;
57    } else if (strcmp("B150", argv[baurdate_index]) == 0) {
58      serial_port_baudrate = B150;
59      return 0;
60    } else if (strcmp("B200", argv[baurdate_index]) == 0) {
61      serial_port_baudrate = B200;
62      return 0;
63    } else if (strcmp("B300", argv[baurdate_index]) == 0) {
64      serial_port_baudrate = B300;
65      return 0;
66    } else if (strcmp("B600", argv[baurdate_index]) == 0) {
67      serial_port_baudrate = B600;
68      return 0;
69    } else if (strcmp("B1200", argv[baurdate_index]) == 0) {
70      serial_port_baudrate = B1200;
71      return 0;
72    } else if (strcmp("B1800", argv[baurdate_index]) == 0) {
73      serial_port_baudrate = B1800;
74      return 0;
75    } else if (strcmp("B2400", argv[baurdate_index]) == 0) {
76      serial_port_baudrate = B2400;
77      return 0;
78    } else if (strcmp("B4800", argv[baurdate_index]) == 0) {
79      serial_port_baudrate = B4800;
80      return 0;
81    } else if (strcmp("B9600", argv[baurdate_index]) == 0) {
82      serial_port_baudrate = B9600;
83      return 0;
84    } else if (strcmp("B19200", argv[baurdate_index]) == 0) {
85      serial_port_baudrate = B19200;
86      return 0;
87    } else if (strcmp("B38400", argv[baurdate_index]) == 0) {
88      serial_port_baudrate = B38400;
89      return 0;
90    }
91    fprintf(stderr, "Error: bad serial port baudrate value\n")
        ;
92    fprintf(stderr,
93        "Valid baudrates: B110, B134, B150, B200, B300, B600,
            B1200, B1800, B2400, B4800, B9600, B19200, B38400\
            n");
```

```
94    return -1;
95  }
96
97  void parse_max_packet_size(int packet_size_index, char **
        argv)
98  {
99    int val = atoi(argv[packet_size_index]);
100   if (val > FRAME_SIZE || val < 0)
101     max_data_transfer = FRAME_SIZE;
102   else
103     max_data_transfer = val;
104
105 #ifdef NETLINK_DEBUG_MODE
106   fprintf(stderr,"\nparse_max_packet_size:\n");
107   fprintf(stderr,"  max_packet_size=%d\n", max_data_transfer
          );
108 #endif
109 }
110
111 void parse_max_retries(int packet_size_index, char **argv)
112 {
113   int val = atoi(argv[packet_size_index]);
114   if (val <= 0)
115     max_retries = 4;
116   else
117     max_retries = 1 + val;
118
119 #ifdef NETLINK_DEBUG_MODE
120   fprintf(stderr,"\nmax_retries:\n");
121   fprintf(stderr,"  max_retries=%d\n", max_retries);
122 #endif
123 }
124
125 int parse_flags(int* t_index, int* i_index, int* b_index,
        int* p_index,
126     int* r_index, int argc, char **argv)
127 {
128   for (size_t i = 0; i < (argc - 1); i++) {
129     if ((strcmp("-t", argv[i]) == 0)) {
130       *t_index = i;
131     } else if ((strcmp("-i", argv[i]) == 0)) {
132       *i_index = i;
133     } else if ((strcmp("-b", argv[i]) == 0)) {
134       *b_index = i;
135     } else if ((strcmp("-p", argv[i]) == 0)) {
136       *p_index = i;
137     } else if ((strcmp("-r", argv[i]) == 0)) {
138       *r_index = i;
139     } else if ((argv[i][0] == '-')) {
140       return -1;
141     }
142   }
143 #ifdef NETLINK_DEBUG_MODE
144   fprintf(stderr,"\nparse_flags(): flag indexes\n");
```

```
145   fprintf(stderr,"  -t=%d\n  -i=%d\n  -b=%d\n  -p=%d\n  -r=%
          d\n", *t_index, *i_index, *b_index,
146       *p_index, *r_index);
147 #endif
148   return 0;
149 }
150
151 int parse_args(int argc, char **argv, int *is_transmitter)
152 {
153
154 #ifdef NETLINK_DEBUG_MODE
155   fprintf(stderr,"\nparse_args(): received arguments\n");
156   fprintf(stderr,"  argc=%d\n  argv=%s\n", argc, *argv);
157 #endif
158
159   if (argc < 2) {
160     return -1;
161   }
162
163   if (argc == 2)
164     return parse_serial_port_arg(1, argv);
165
166   int t_index = -1, i_index = -1, b_index = -1, p_index =
          -1, r_index = -1;
167
168   if (parse_flags(&t_index, &i_index, &b_index, &p_index, &
          r_index, argc,
169       argv)) {
170     fprintf(stderr, "Error: bad flag parameter\n");
171     return -1;
172   }
173
174   if (t_index > 0 && t_index < argc - 1) {
175     if (read_file_from_disk(argv[t_index + 1], &file_to_send
            ) < 0) {
176       return -1;
177     }
178     *is_transmitter = 1;
179   } else {
180     if (i_index > 0 && i_index < argc - 1) {
181       if (read_file_from_stdin(&file_to_send) < 0) {
182         return -1;
183       }
184       *is_transmitter = 1;
185     }
186   }
187
188   if (b_index > 0 && b_index < argc - 1) {
189     if (parse_baudrate_arg(b_index + 1, argv) != 0) {
190       return -1;
191     }
192   }
193
194   if (p_index > 0 && p_index < argc - 1) {
```

```
195       parse_max_packet_size(p_index + 1, argv);
196     }
197
198     if (r_index > 0 && r_index < argc - 1) {
199       parse_max_retries(r_index + 1, argv);
200     }
201
202     return parse_serial_port_arg(argc - 1, argv);
203   }
204
205   int main(int argc, char **argv)
206   {
207     int port_index = -1;
208     int is_transmitter = 0;
209
210     if ((port_index = parse_args(argc, argv, &is_transmitter))
          < 0) {
211       help(argv);
212       exit(EXIT_FAILURE);
213     }
214
215     if (is_transmitter) {
216       fprintf(stderr, "transmitting %s\n", file_to_send.name);
217       return send_file(argv[port_index], &file_to_send,
            max_retries);
218     } else {
219       fprintf(stderr, "receiving file\n");
220   #ifdef NETLINK_DEBUG_MODE
221       fprintf(stderr, "\tserial_port_baudrate:%d\n",
            serial_port_baudrate);
222       fprintf(stderr, "\tis_transmitter:%d\n", is_transmitter)
            ;
223   #endif
224       return receive_file(argv[port_index], max_retries);
225     }
226   }
```